325 South 1000 East

Salt Lake City, UT 84102

April 3, 2018

Dr. Mikhail Skliar

Beehive State Engineers

Salt Lake City, UT 84112

Dear Dr. Skliar:

On February 1, 2018, you tasked Dainger Adams, Malcolm Bailey, and Nathan Le with predicting a power demand using a neural network and then controlling a hypothetical generator to meet the demand. Our project is described in detail in the attached report entitled "Predictive Control using Neural Network Systems."

We were able to create a neural network that could predict power demand using historical data. It was found that the best algorithm was Resilient Backpropagation using 6 nodes. Using the forecast generated by the network a set point generator was able to adequately control a simulation of a shell and tube heat exchanger. We believe that this strategy could be applied to a power generator.

Our recommendation for the future would be to use more historical data to train the neural network. Using 3+ years' worth of data would likely make the network accurate enough for industrial usage.

Sincerely,

Dainger Adams Nathan Le Malcolm Bailey

Dainger Adams Nathan Le Malcolm Bailey

PREDICTIVE CONTROL USING NEURAL NETWORK SYSTEMS

By

Dainger Adams

Malcolm Bailey

Nathan Le

Project #11

Shell and Tube Heat Exchanger Neural Network

Assigned: February 1, 2018

Due: April 3, 2018

Submitted: April 3, 2018

Project Team Members for Group 11:

Dainger Adams, Group Leader

Dainger Adams

Malcolm Bailey

Malcolm Bailey

Nathan Le

Nathan Le

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUMMARY

Predictive Control using Neural Network Systems

Group 11

Dainger Adams, Malcolm Bailey, Nathan Le

Report Date: April 3, 2018

The primary interest of our group was whether it was possible to smoothly and accurately meet power demand, an inherently noisy and hard to predict variable. Our approach uses a predictive neural network in order to manipulate the set point of a heat exchanger, which was our substitute for a power plant generator. The neural network was trained on a year's worth of historical power demand data, and several inputs of our choice. Operation of the heat exchanger generated data for a computer model by varying the shell-and-tube heat exchangers valve % opening and plotting the resulting temperature of the tube side coming out. The transfer function and control of the heat exchanger was modeled in Matlab and Simulink. The research led to two important conclusions: First, neural networks can accurately predict future power demand using several years of training data and the appropriate inputs, only having a mean squared error of around 500 on a graph with an average Y value of 6000. Secondly, the set point function can attain satisfactory results when controlling a heat exchanger.

Moving forwards from this project, there are numerous additional tasks that could improve the performance of our system. Some examples include new neural inputs such as weekly average power usage term, a term that accounts for how many hours of daylight there are in a day, and a very intricate one could be noting days of public events that would draw large crowds from their homes. There is also room for improvement in the fitness function part of our set point function. The sole goal during our project was meeting the power demand at any point with ramp rate constraints, but perhaps there could be some component that accounts for the second derivative to minimize aggressive slope changes. It would also be very interesting to be able to use actual generators for control instead of simulating a heat exchanger.

I. INTRODUCTION

Modern day power plants must be very versatile to meet fluctuating demand. Often times, rapidly adjusting to this changing power demand leads to accelerated equipment degradation, which in turn hurts company profit. There is a lot of ongoing research into methods of effectively meeting power demand without damaging generators. The inability to rapidly change to the needs of a region without damaging equipment results in an optimization issue between producing the right amount of power without hurting equipment excessively.

By training a neural network to aid in power demand prediction over a 24 hour period, our project can identify the most effective way to generate power for the entire day. This will lead to minimizing equipment damage and optimally supplying power to the populace. This, in turn, results in a better bottom line for any company that employs such an approach. Combining a trained neural network with an optimization algorithm will generate a smooth graph of varying setpoints of power generation. The smooth curves mean a softer demand on generators while still supplying an ample amount of power when it is needed.

The project will be divided into three phases. Phase one entails acquiring historical weather and power demand information in order to train a neural network for future demand prediction. Historical weather data for Utah will be acquired via the National Oceanic and Atmospheric Administration's (NOAA) accessible programmable interface (API). Historical power demand data will be supplied by Rocky Mountain Power (RMP). The data will be divided into three subsets, one for training purposes, one for validating the performance of our trained neural network, and a final one for testing the final algorithm on. Once we have a functional neural network, phase two begins with utilizing MATLAB's algorithm packages in order to optimize an array of setpoints representing power generated for each 24 hour day of power usage. The first step in using any optimization algorithm is to identify both costs (measures of efficiency) and constraints (bounds of our system). Once we have defined these in a way that can be understood in MATLAB, then it's just a matter of executing the algorithm and observing its efficacy. Once these two phases are fully complete, the project team can fully focus its efforts into controlling a heat exchanger. The concept of using the heat exchanger is to relate the energy transfer between the cold and hot stream to a power generated as a means of simulating a large power plant that cannot quickly change set points. In order to apply our algorithm to controlling the heat exchanger (by manipulating one stream's valve opening), we need to derive or find a transfer function in research literature that relates valve opening to the energy transfer between the hot and cold streams. Once we have a transfer function that can be incorporated into our optimization algorithm, the team will be able to control the valve opening with our algorithm. Then the performance and error of the algorithm will be observed for the third subset of meeting an estimated power demand.

When the project is finished, there will be a concrete proof of concept for employing neural networks and optimization algorithms for controlling a power plant. With this technology in place, power companies across the country will be able to effectively reduce the strain on their generators while continuing to provide adequate power for the areas they operate in.

II. Theory

Neural networks are a computational model that allow for prediction and modelling of systems where the relationships between the input and output are not necessarily obvious. They are also a great approach for any sort of task that is difficult to program by hand. The basic anatomy of a neural network includes at least 3 layers (as shown above with a single hidden layer), and at least one input node, one hidden node, and one output node. Of course, most real systems have many more input and hidden nodes, to represent the complex interactions between each input that occur to produce some output. The name neural network comes from the fact that each node is similar to how a neuron in the brain might work. Each connection between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

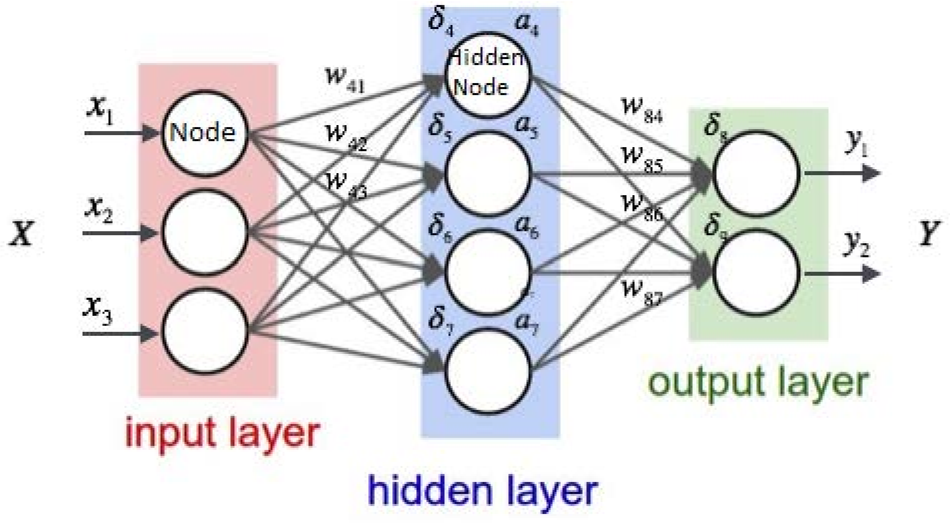


Figure 1: A basic diagram that illustrates artificial neural networks. X represents the inputs, Y represents the outputs, and w are the weights on each interaction. [1]

Most neural networks spend a significant time learning. The general idea of learning with neural networks entails giving the network some task to solve, a set of functions, and some cost function that allows it to measure progress and accuracy. The three primary types of learning are supervised learning, unsupervised learning, and reinforcement learning. “In supervised learning, the environment provides, for each input, an explicit desired output or target. The goal of the learning system is to learn the mapping from inputs to outputs specified by this teaching signal from the environment.” Supervised learning is the type of learning that is employed specifically in our project, where we have inputs such as time of day and month compared with the output of power demand that we seek to find a relationship between. “[The] environment provides inputs but gives neither desired targets nor any measure of reward or punishment.” Surprisingly, as directionless as this method sounds, it’s actually used quite frequently. This type of learning could be used for clustering, or dividing large groups of information into smaller groups. The final type of learning, reinforcement learning works as follows, “for each input to and output from the learning system, the environment provides feedback in the form of either reward or punishment. The overall performance measure that the system tries to maximize is the sum of total future rewards, which can be weighted to favor immediate gain over longer-term gain.”[2]

There is also even more variation in neural networks by the type of algorithm. Some examples of common and effective algorithms include: Bayesian Regularized, Resilient Backpropagation, and Scaled Conjugate Gradient. Matlab’s documentation says the following for each algorithm, “[The Bayesian regularization] minimizes a combination of squared errors and weights, and then determines the correct combination so as to produce a network that generalizes well.” The Bayesian is known as a network that is fairly accurate in general, which in turn makes this algorithm more robust against overfitting data. [3]

The resilient Backpropagation (Rprop) algorithm operates on the same general idea as a gradient descent, which involves taking steps in the derivative of the performance function until it finally reaches zero. The unique feature of the (Rprop) is that it follows the slope of the land in a slower more controlled fashion. “The adaptation-rule works as follows: Every time the partial derivative of the corresponding weight w_{ij} changes its sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update-value A_{ij} is decreased by the factor η-. If the derivative retains its sign, the update-value is slightly increased in order to accelerate convergence in shallow regions.” [4]

The final method of scaled conjugate gradient follows optimizes in the same general way of descending down the gradient like Rprop, where it picks a search direction, and then it picks a step size thereafter. They key difference is that this method has a unique approach for step direction and step size by using the second order approximation of the error. The details of this algorithm are very abstruse, so any further interest should refer to the research. [5]

In this project, we wanted to model and simulate a generator that would be present at a power plant. Since the team did not have access to any generator that could directly be modelled or used, a heat exchanger was used as a substitute. The shell & tube heat exchanger functions as a substitute since both a generator and the shell & tube heat exchanger take a considerable time to change from one steady state to another after a step change. In order to utilize the heat exchanger for simulations, a mathematical equation that describes the systems behavior after a step change or disturbance needed to be derived.

The process of developing a mathematical relationship between the inputs and outputs of a system or process based on input-output data is referred to as system identification. After obtaining an ample amount of raw input-output data, a program such as Matlab’s system identification toolbox can be utilized. Inside of system identification software, there are many options for fitting and retrieving constants. The two primary modelling types are black-box and grey-box.

In black box modelling, the main goal is to fit the data without a specific mathematical structure of the model. “Black-box modeling is usually a trial-and-error process, where you estimate the parameters of various structures and compare the results. Typically, you start with the simple linear model structure and progress to more complex structures. You might also choose a model structure because you are more familiar with this structure or because you have specific application needs.” This approach is very effective when you’re not entirely certain what sort of transfer function the system may have, as you can apply many different models simultaneously and see their respective accuracies.

Grey (or gray) box modelling, on the other hand, starts off with a specified mathematical model structure. “In the grey-box approach, you use the data to estimate the values of the unknown parameters of your model structure. You specify the model structure by a set of differential or difference equations in MATLAB and provide some initial guess for the unknown parameters specified.” This method is most appropriate for systems that are already well understood, and have obvious model structures. An example of this could be a basic mass-spring-damper system. It is usually quite obvious which type of modelling method is most appropriate for a project. In this project’s case, the black box method was employed.

After choosing either method, it is essential that the quality of the model be evaluated. Inside the Matlab toolbox, you can plot each of your potential models over the actual output. Furthermore, Matlab will display the fit in terms of 0% to 100%, making it quite obvious which models perform the best and how well it performs. [6]

Deriving an accurate model for the shell & tube heat exchanger is essential so that it can be modelled by computer programs such as Matlab’s Simulink. In Simulink, a user can create blocks that represent transfer functions (one that could represent a heat exchanger), inputs, disturbances, and controllers. With all of these features combined, it becomes quite straight forwards to accurately model any step change or disturbance for a piece of equipment. In this project’s case, this would allow for modelling the heat exchanger as it attempts to shift its set point to meet a theoretical energy transfer/demand. By employing many different apps inside Matlab, this team is capable of simulating a heat exchanger (as an estimation of a generator) as it is given many different set points based on the predictions from our neural network. This process allows us to effectively meet a predicted power demand, which semi-accurately matches the real power demand, as shown in our results section later.

III. Apparatus and Procedure

1) Gathering Data With the Shell & Tube Heat Exchanger

The experimental apparatus used is a shell-and-tube heat exchanger located in Room 3290 of the Merrill Engineering Building (MEB). Heat transfer occurs by running hot fluid through the tubes, and cold fluid through the shell-side in opposing directions. Tube and shell flow rates are controlled by changing valve controls on the Opto software. The data from the heat exchanger is used to determine the ramp up and ramp down rates in our model. A schematic of the heat exchanger is shown in Figure 4.

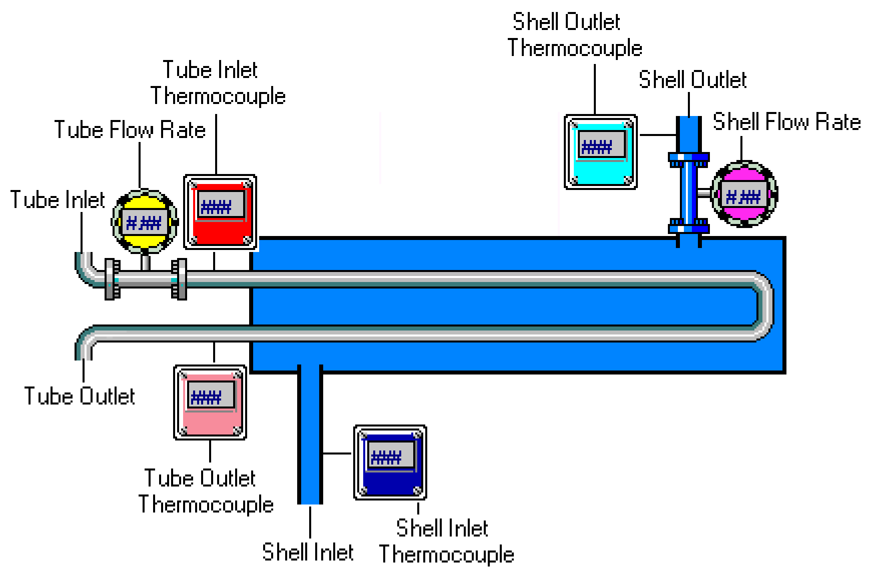


Figure 2: Schematic of Shell-and-Tube Heat Exchanger. Shows different devices attached to the heat exchanger. Acquired from Shell-and-Tube Heat Exchanger Laboratory Manual [7].

Design specifications of major components are listed in Appendix C. It should be noted that many of the procedures listed are from the Shell-and-Tube Heat Exchanger Laboratory Manual [7]. The procedure is performed as followed:

A. Cleaning Out The System

When the heat exchanger is unused for extended periods of time, rust can build up, this may prevent us from acquiring accurate data. In order to prevent this the heat exchanger needs to be cleaned out before acquiring our experimental data. Procedure for cleaning out the system is as follows:

1. Open both the shell-and-tube computerized valves to 100%.
2. Open valve 10, the gate valve on the shell inlet.
3. Open valve nine, the ball valve on the shell outlet.
4. Allow the tank to fill approximately 2/3 full.
5. Close valve nine manually.
6. Check to see that valve 3 is also open.
7. Turn on the pump. The pump switch is located on the wall directly to the west of the heat exchanger.

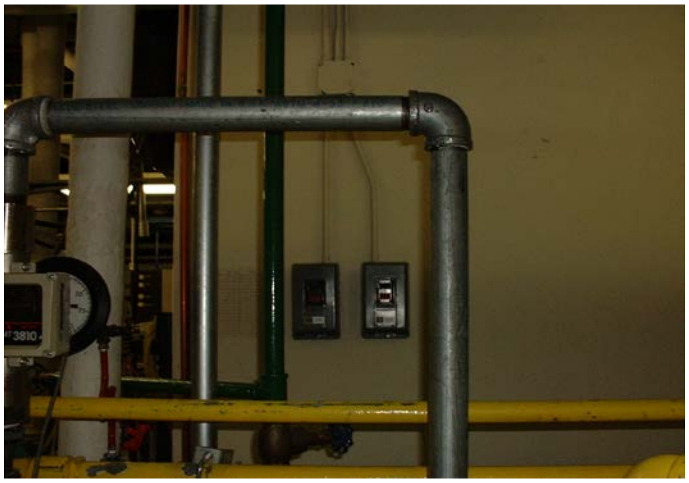


Figure 3: The Pump Switches. Located on the west side wall of the heat exchanger. Acquired from Shell-and-Tube Heat Exchanger Manual [7].

8. Allow the water to run for one to two minutes. When the water has been run through the whole system (water should be cleaner), turn the pump off.
9. Open the tank discharge valve (a valve near the floor on the west side of the tank) to drain the tank completely.
10. Close the tank discharge valve.
11. Open the shell side ball valve (valve nine) and refill the tank, again approximately 2/3 full. Close valve nine.
12. Turn on the pump and allow the water to circulate for approximately five minutes, or as long as the operators deem necessary for the system to be cleaned out. This can be determined when the color of the tube discharge water is constant.
13. Turn the pump off.
14. Drain the tank and close the tank discharge valve
15. Repeat steps 12-14 at least once more to replace the water in the tubes with clean water. The steps can be repeated multiple times until the heat exchanger is deemed to be clean.

B. MAKING MEASUREMENTS

To make the measurements, following steps are followed:

1. Start the software by opening the Opto Control Shell and Tube program.
2. On the software, set the shell valve control and tube valve control to 100%.
3. Open the shell flow tank feed valve located above the lip of the tank.
4. Open the recycle valve 3-4 turns.
5. Open the pump outlet valve 100%.
6. Close the shell flow drain valve and verify that the hose connected to the shell flow drain valve is inserted into the drain pipe.
7. Close the tank drain valve.
8. Open the main water supply valve.
9. Water should now flow into the tank, fill the tank to approximately 65% full.
10. Open the shell flow drain valve
11. On the software, set the shell valve to 0%.
12. Verify that the recycle valve and pump outlet valve are open.
13. Verify the tube valve is set to 100% on the software.
14. Start the pump and ensure the pump output pressure does not exceed 50 PSI by adjusting the recycle valve.
15. Allow the system to reach steady state.
16. On the software, adjust the shell valve control to desired levels.
17. Monitor the pump outlet pressure.
18. Allow the system to reach steady state.
19. Repeat steps 16 through 19 until the data acquired is satisfactory.

C. USING HEAT EXCHANGER DATA TO FIND OUR RAMP UP AND DOWN RATES

After acquiring our data from the heat exchanger we will find the transfer functions (ramp up and down rates) of our system by:

1. Acquire the following data obtained from the heat exchanger: tube outlet temperature and shell valve percent opening.
2. Import the data into Matlab 2017b
3. Using Matlab's "systemIdentification" toolbox and the newly acquired data, model the rate of temperature change as a second order transfer function plus dead time (SOPDT).

D. SHELL-AND-TUBE HEAT EXCHANGER SAFETY

Operators should wear safety glasses when working with the heat exchanger. When opening or closing the steam valves, the operator should be wearing heat resistant gloves. The pump should never be ran dry due to wear and tear on the valves and seals as well as causing the pump to overheat. The level of the tank should be continuously monitored for spills and leakage. Finally, operators should avoid touching warm metal.

2) Acquiring Historical Weather Data with the NOAA API

A) Request a Token For Access

In order to utilize the NOAA's API, the user needs to first acquire a token. The token requires an individual account to prevent an excessive number of access attempts (10,000 requests per second or 5 requests per second). Once a token is acquired the API can now be utilized.

B) Making a Request

At this point, a user can find the station ID of a weather station that provides relevant data for all the time points the user is interested in. Then, they can plug in additional specifications, such as data type (average temperature, precipitation, etc.), range of dates, and units (metric or imperial). Once the request has been made, Matlab will receive a matrix with the relevant information.

C) In this experiment, the date was acquired along with the average temperature and precipitation.

3) Formatting Data To Train the Neural Network

A) A neural network has to be trained on a number of inputs and outputs. For our system the sole output was the predicted power demand. Inputs include the time of day, the date, the average temperature for that day, the precipitation on that day, whether that day was a working day or not, and the previous day's power load.

B) The data was organized into a single table with all of the inputs in columns, then another column in a separate table with the outputs.

4) Using the Matlab Neural Network Toolbox

A) Open MATLAB 2017b

B) In order to open this toolbox in Matlab, type "nnstart" in the console. This will pull up a GUI for the Neural Net Toolbox

C) At this point a fitting app was selected as it yields the best performance as compared to the time series app. Then, the input columns and output columns are provided for the toolbox.

D) The next step is to select how much of the dataset should be used for training, validation, and testing. The default division of 70%, 15%, and 15% is useful, but for a complicated system such as ours we needed to use as much training data as we could use to increase accuracy.

E) After everything else is specified, the number of hidden neurons (nodes) and the method for training need to be selected. Picking the ideal number and method requires some trial and error. The only reliable rule of thumb is to start with a number of nodes equal to the number of inputs and then work from there.

5) Modeling the Heat Exchanger Using Matlab's System Identification

A) Open MATLAB 2017b

B) Type "systemIdentification" in the command window and press enter

C) Click "import data" and select "time domain data"

D) Select "Shell Valve Control" and "Tube Outlet Temperature" as your input and output data. The values are acquired from the heat exchanger data.

E) Click "estimate" and select "Process Models"

F) Select 2 poles, check that the initial condition is set to "auto".

G) For the ramp up rate, select "estimate." The Transfer function should now be obtained for this rate.

H) For the ramp down rate, under disturbance model, select "first order." We use a disturbance model for the ramp down rate because the cold fluid is coming from other sources in the building and may fluctuate.

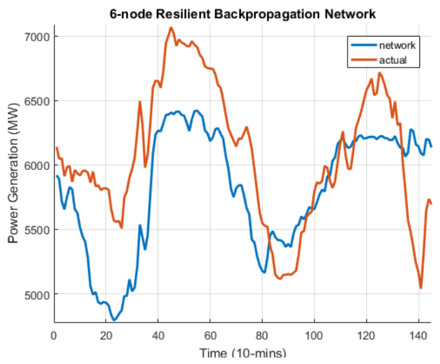
I) Click "estimate", at this point you should now have transfer functions for both the ramp up and ramp down rate.

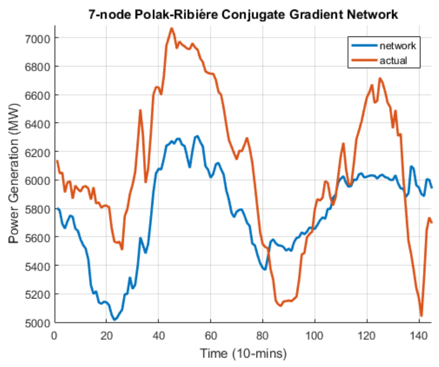
IV. Results and Discussion

The first section of this project entailed training a neural network to predict future power demand. The inputs included the daily average temperature, daily precipitation, if it was a weekday, and the previous day's power demand at the same time. This network was trained using a year's worth of power generation data at 10-minute intervals.

To train the neural network Matlab's neural network toolbox was used. This toolbox comes with many different algorithms and each algorithm has different results at different nodes, even differing on unique training attempts for the same number of nodes. The best training algorithms were found to be the Resilient Backpropagation, Polak-Ribiére Conjugate Gradient, and Fletcher-Powell Conjugate Gradient.

Each algorithm was run 25 times with a varying number of nodes from 2 to 7. The entire compilation of errors is present in Appendix A, while the most accurate results are discussed just below.





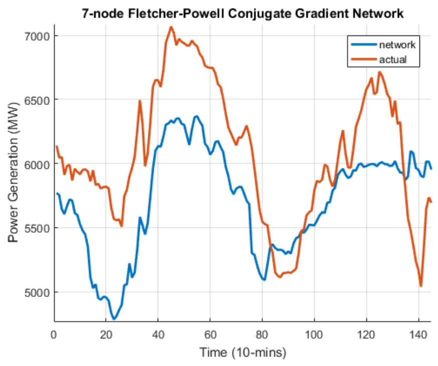


Figure 4: Neural Network Algorithm Plots. The plots display the performance of the most close fit network types.

In order to validate the models, each trained network is plotted with data that was not used in the training phase. The resulting graphs are encouraging, the general trends present in the actual data are mimicked with moderate precision by the trained networks. The top three networks (which are the same ones plotted) are shown below with a mean squared error for comparison. The error term may initially seem surprisingly large, but the average magnitude of the Y values around 6000.

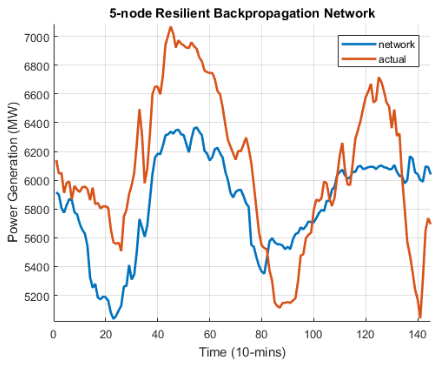
Table 1: Mean Squared Error of Top Three Neural Network Training Algorithms with Standard Deviation

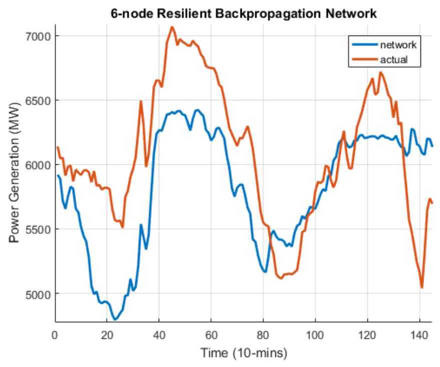
	Algorithm	nodes	Root Mean Squared Error (± SD)
First	Resilient Backpropagation	6	560.2 ± 59.2
Second	Polak-Ribiére Conjugate Gradient	7	598.3 ± 80.3
Third	Fletcher-Powell Conjugate Gradient	7	606.2 ± 89.6

It is worth noting that although these were the top performers over 25 trails, they were not necessarily the highest performers on each individual trial. This volatility could be due to the size of our training data, and it is suspected that 2 to 3 years of training data would yield better results. Upon initial observation, these errors may seem to condemn the ability of the networks, but in truth these errors are fairly small when considering the magnitude of the Y values are around 6000 on average. In the next table (table 2), there are more results for the most accurate training algorithm (Resilient Backpropagation), showing some other node counts other than the most accurate.

Table 2: Mean Squared Error of Resilient Backpropagation With Varying Nodes

	Nodes	Root Mean Squared Error (± SD)
First	6	560.2 ± 59.2
Second	7	588.2 ± 85.2
Third	5	623 ± 80.1





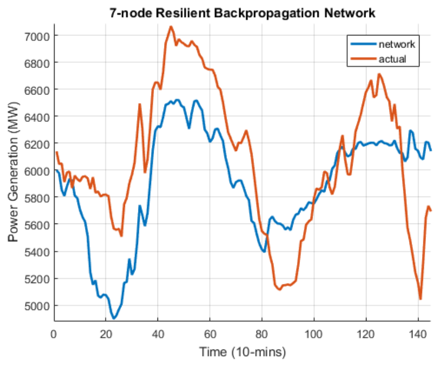
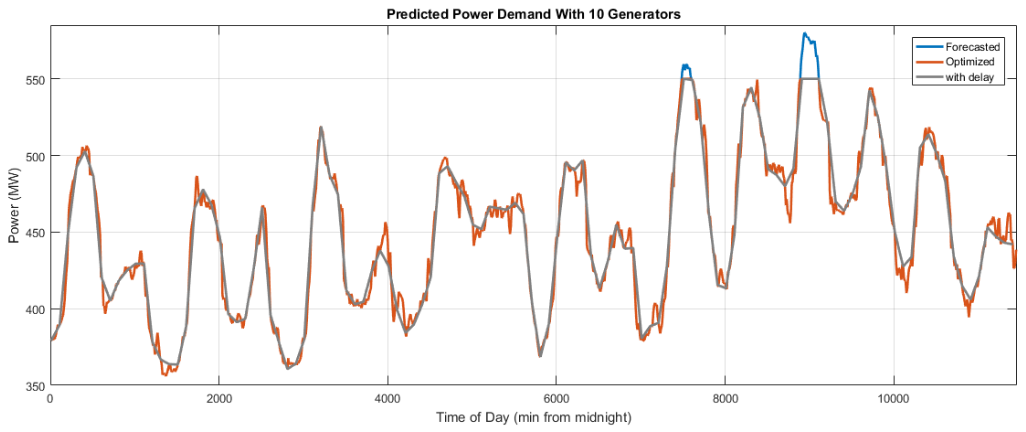


Figure 5: Node Plots using Resilient Backpropagation. Since Resilient did the best, extra information is provided for it.

The correct number of nodes is important because there needs to be enough to accurately predict the system without overfitting the data. A good rule of thumb for selecting the number of nodes is to initially use the number of inputs you are working with, and then vary it from there. In our case, we had 7 inputs initially, and the best result was attained at 6 nodes.

The second section of this project was to devise a set point function that would follow the predicted demand, but would also consider the functional limitations of the theoretical generator. The function that was created would attempt to follow the power demand as closely as possible while including a limit on the rate that it could ramp up or down, as well as a maximum power that each generator is capable of. In order to account for the fact that we're assuming there's an array of generators, we chose to split the power demand evenly (by dividing the total power demand by the number of generators). The function was created in a Matlab script that basically told the set point to always be on the demand line, unless doing so would exceed the ramp up/down rates. In the case of exceeding the ramp rates, the function would just move towards the demand at whatever the ramp rates are specified as. The results of our optimal set point function are illustrated in the plots below:



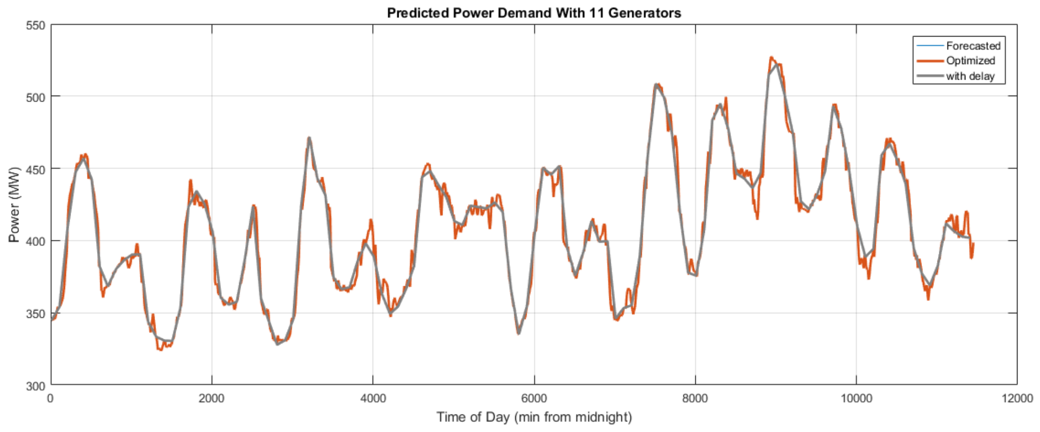


Figure 6: Power Demand Set Point Response. This graph shows the forecasted power demand, with the set points of the generator array overlaying it. Optimized changes the set point every 10 minutes, while "with delay" changes it every 100 minutes.

In the top plot the optimizer handles the forecast as if there are only 10 generators, this results in two sections where the total power demand is actually above what the array of generators is even capable of producing. In the case of 11 generators, the power demand is attainable at all points. Another aspect worth commenting on is the presence of the "optimized" line and the "with delay" line. The difference between these two plots is the frequency at which they respond to the forecasted power demand. The "optimized" line will adjust its set point every 10 minutes towards the newest demand. On the other hand, the "with delay" graph only checks every 100 minutes before changing its target value. Overall, both functions are very accurate at meeting demand, and the 100 minute period allows for clearer controllability since it does not have to factor in a dead time every 10 minutes.

The final section of this project was to simulate controlling a real heat exchanger system. This was done in the Matlab simulation environment, Simulink. Using real values from the shell and tube heat exchanger and the system identification toolbox the following transfer function was found:

$$\frac{Y(s)}{U(s)} = -\frac{0.41183}{(432.57s+1)(42.351s+1)}e^{-.5s}$$

During the system identification phase of this project, the previous transfer function was found solely for the step down response. When attempting to model the step up the team encountered serious troubles in creating an equation that actually modelled the system accurately. In a future continuation of this experiment, finding a transfer function for the step up response would lead to a significant step in performance and accuracy for the simulated heat exchanger.

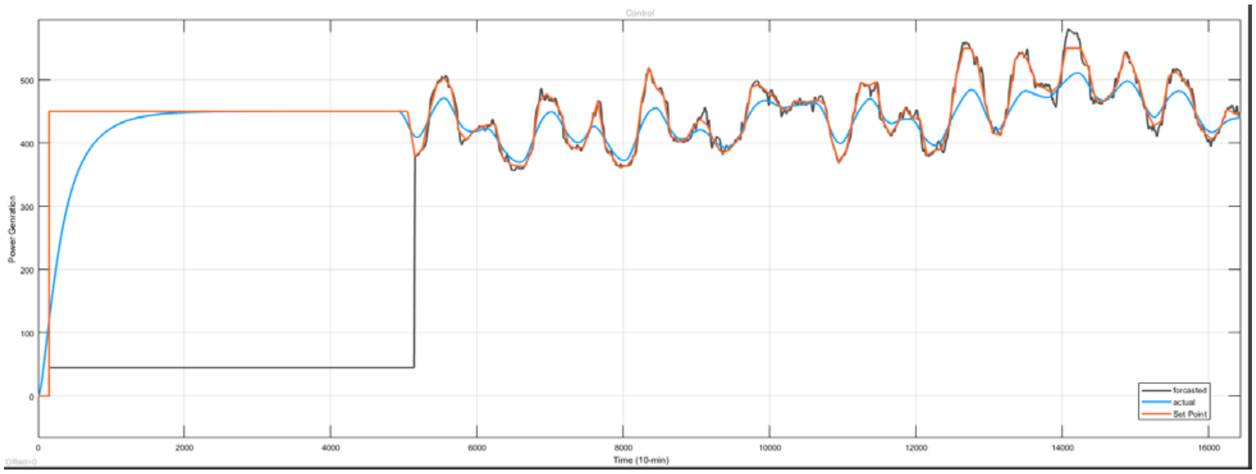


Figure 7: Power Demand Heat Exchanger Set Point Response. This graph shows how the simulated heat exchanger reacts to the changing set points.

Observing the graph, it becomes clear that the transfer function derived is a powerful model for step downs, but does not have the ability to step up correctly. It seems likely that with an improved model of the step up response, the ability to meet the demand would increases even further. For future experimentation two models should be used one for ramp up and a second for ramp down.

The initial step on the graph above is for “warming up” the heat exchanger, as it would be unlikely that the generators would start from 0 power every day. There is also a 150 min delay between the simulation and the setpoint. This means the setpoint would need to be fed into the simulation 150 mins before the setpoint needed to be reached this is mostly caused by dead time in the system. This is an adequate simulation that needs a few modifications to be perfected.

V. Conclusions and Recommendations

This project has given our team a lot of insight into the ability of neural networks and the difficulty of simulating a generator meeting power demand. In our first phase of training the network, we attained good results with many different training algorithms, but the best performer overall was the Resilient Backpropagation method. We suspect that the main source of inaccuracy that still remains in the neural networks approximation is due to a lack of training data. Rocky Mountain Power provided us with one year of training data for power demand, but it is likely that 3+ years would prove very beneficial. In the second part of our trials, we created a very simple function for manipulating the set point of a potential generator in order to meet power demand predictions. The part of this section that deserves more attention is characterizing generator behavior near its maximum power output. This would entail adding more terms to our optimizer function that would account for a decreasing ramp rate near the upper end of performance, and accounting for the possibility of raising a generator above 100% power output for a short period. In the final phase of our experiment, we successfully simulated a heat exchanger in order to roughly model a generator. This resulted in a graph with the actual power output meeting demand when trending downwards, but having it fall short on step ups. The clearest path to fixing this is constructing two separate transfer functions that individually describe the behavior of the heat exchanger stepping up and stepping down.

Nomenclature

Symbol	Definition	Units
X	Combination of inputs into the neural network	N/A
x_i	Input into the neural network	N/A
Y	Combination of outputs of the neural network	N/A
y_i	Individual outputs of the neural network	N/A
SD	Standard Deviation	N/A
U	Input into the Heat Exchanger	N/A

REFERENCES

[1] V. Valkov, “Creating a Neural Network from Scratch - TensorFlow for Hackers (Part IV),” Medium, 19-May-2017. [Online]. Available: https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8. [Accessed: 03-Apr-2018].

[2] D. Wolpert, Z. Ghahramani and J. Flanagan, "Perspectives and problems in motor learning", Trends in Cognitive Sciences, vol. 5, no. 11, pp. 487-494, 2001.

[3] D. McKay. "Bayesian interpolation." Neural computation. Vol. 4, No. 3, 1992, pp. 415–447.

[4] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm", IEEE International Conference on Neural Networks, 1993.

[5] M. Møller, "A scaled conjugate gradient algorithm for fast supervised learning", Neural Networks, vol. 6, no. 4, pp. 525-533, 1993.

[6] "System Identification Overview- MATLAB & Simulink", Mathworks.com, 2018. [Online]. Available: https://www.mathworks.com/help/ident/gs/about-system-identification.html. [Accessed: 30- Mar- 2018].

[7] A. M. Redding, Shell-and-Tube Heat Exchanger Laboratory Manual. Not Published, 2001.

VI. APPENDICES

APPENDIX A

RAW DATA

Table 3: All Training Algorithm Root Mean Squared Errors , Standard Deviations, and Minimum RMSE.

	two-nodes			three-nodes			four-nodes			five-nodes			six-nodes			seven-nodes		
	mean	sd	min	mean	sd	min	mean	sd	min	mean	sd	min	mean	sd	min	mean	sd	min
trainlm	694.2	92.1	471.9	632.1	101.8	472.5	620.9	88.4	494.1	690.1	139.0	468.3	654.9	115.9	502.9	764.8	157.3	489.9
trainbr	683.3	91.9	480.1	609.8	86.9	475.8	610.1	129.8	453.0	663.7	144.0	479.2	760.0	158.3	482.0	812.7	136.9	518.7
trainbfg	731.2	132.4	555.6	722.2	114.6	546.3	672.4	95.6	510.6	682.0	95.0	507.0	693.6	116.1	424.9	676.5	108.1	537.0
trainrp	716.9	103.6	481.3	647.1	86.9	527.9	619.8	89.4	483.3	623.1	80.1	492.1	560.2	59.2	441.4	588.2	85.2	456.5
trainscg	694.8	90.3	508.4	662.7	105.6	483.3	659.2	108.8	448.9	650.7	88.8	537.4	613.6	85.1	504.7	626.0	85.6	487.3
traincgb	719.7	149.7	521.9	655.8	89.9	542.3	653.2	104.2	455.1	607.2	61.6	513.0	624.0	76.4	514.3	617.6	72.5	445.0
traincgf	716.7	76.4	526.0	709.4	99.0	527.1	631.9	81.7	536.0	640.0	107.7	480.0	620.1	82.0	483.2	606.2	89.6	419.8
traincgp	705.8	88.5	545.1	673.7	96.0	481.2	630.9	111.6	473.5	631.3	98.4	497.0	623.9	86.0	519.8	598.3	80.3	463.0
trainoss	740.9	71.6	542.4	657.0	97.8	478.0	675.1	114.2	444.6	646.2	105.8	443.6	645.6	110.9	484.6	647.2	86.7	464.1
traingdx	893.7	342.7	563.4	983.8	397.9	611.2	959.4	355.8	427.3	953.6	384.6	591.6	830.0	196.1	450.0	831.5	237.7	536.3

Table 4: Highest Performing Algorithms

	mean		sd		min	
	al	node	al	node	al	node
first	trainrp	6	trainrp	6	traincgf	7
second	traincgp	7	traincgb	5	trainbfg	6
third	traincgf	7	trainoss	2	traingdx	4

APPENDIX B

ITEMS OF EQUIPMENT

1) Shell-and-Tube Heat Exchanger

- Shell

    Material: Carbon Steel

    Length: 6.25 ft

    Inside Diameter: 6.0 in

    Outside Diameter: 6.60 in

    Incoming pipe diameter: ¾ in, Schedule 40 stainless steel

    Outlet pipe diameter: 1 ¼ in, Schedule 40 stainless steel

- Tubes

    Number of Tubes: 28

    Number of Passes: 2, U-bend configuration

    Material: Copper, Schedule 40

    Length: 6.0 ft

    Inside Diameter: 5/8 in

    Outside Diameter: ¾ in

    Pitch: 7/8 in, Triangular pitch

    Incoming pipe diameter: 1 ¼ in, Schedule 40 stainless steel

    Outlet pipe diameter: 1 ¼ in, Schedule 40 stainless steel

- Baffling

    Number of Baffles: 2

    Baffle spacing: 30 5/8 in from tube side

    50 ¾ in from tube side

- Holding Tank

2) Pump

Manufactured by: Worthington Pump Incorporated

Model: Worthington D520

Size: 1.2X1X5

Impeller Diameter: 5.25 in

Operating Pressure: 21 psi

Incoming pipe diameter: 1 ½ in, Schedule 40 stainless steel

Outlet pipe diameter: 1 ¼ in, Schedule 40 stainless steel

3) Double Pipe Heat Exchanger

Material: Schedule 40 stainless steel

Length: 14 ft

Inside Pipe Diameter: 1 ¼ in

Outside Pipe Diameter: 2 ½ in

4) Valves

- Gate Valves

Manufactured by: Stockham

- Ball Valves

Manufactured by: Watts Regulator

- Computer Controlled Valves

Manufactured by: Honeywell

Model: Modulating Valve Actuator ML 7984

Operating Temperature: 0 to $55^{o}C$

Tube Valve Discharge Coefficient: 29.3

Shell Valve Discharge Coefficient: 11.7

P (for shell-and-tube): 5.0

I (for shell-and-tube): 2.0

D (for shell-and-tube): 0.000

5) Flow meters

Manufactured by: Brooks Instruments

Model: MT 3810

Accuracy: + or – 5% full scale from 100% to 10% of scale reading

Repeatability: 0.25% full scale

Operating Temperature: -29 to $215^{o}\mathrm{C}$

Flow Range: 3.52 to 35.2 gpm for shell side flow meter

8.80 to 88.0 gpm for tube side flow meter

6) Thermocouples

Manufactured by: Omega

Model: Type T

Range: -60 to $100^{o}\mathrm{C}$

Accuracy: 1.0C or 0.75% above $0^{o}\mathrm{C}$ (whichever is greater)

1.0C or 1.5% below $0^{o}\mathrm{C}$ (whichever is greater)

7) Low Pressure Steam

Pressure: 15 psi

Temperature: $100^{o}\mathrm{C}$

8) Computer

Manufactured by: Dell Systems

Operating System: Windows NT

Software:

a. Opto-22 electronics and computer based software, Version R3.16. Copyright 1996-2000 Opto-22
b. MATLAB 2017b
    i. Neural Network Toolbox
    ii. System Identification Toolbox
    iii. Simulink