Digital Twin and Machine Learning Frameworks for Control of Data Center Energy Management Systems

Thesis by

Thomas Hosmer

In Partial Fulfillment of the Requirements

For the Degree of

Bachelors of Science

University of California, Berkeley

Berkeley, California

ABSTRACT

Digital Twin and Machine Learning Frameworks for Control of Data Center Energy Management Systems

Thomas Hosmer

As society continues to expand its digital footprint, data centers are needed to handle the growing input and storage of information. Data centers have grown to require 1% of the world's energy supply; meanwhile climate change has driven research and policy to adapt our power grid for renewable sources to reduce CO_2 emissions. With energy intensive data centers clashing with a grid that is struggling to modernize and replace fossil fuels, it is necessary that we develop methods to optimize both grid and data center efficiency. Advanced grids that rely on renewable energy will feature multiple sources of power that span the grid (e.g. solar panels) rather than a single, centralized supplier (e.g. a coal fired power plant). A grid dependent on renewable power will require control for energy flow between varying generators and loads. Energy management systems (EMS) are frameworks that achieve this and can be scaled from single devices to national power grids. Pairing an EMS with a data center campus can reduce energy consumption. In this work, a digital twin inspired by the work of Dr. Tarek Zohdi for an EMS is developed. To mimic reality while saving computational expense, device behavior in the EMS is modeled from a neural network that was trained on synthetic data. Energy levels for the synthetic data were designed to match the order of magnitude for energy consumption from Dr. Zohdi's simulation of heat transfer in a data center. This enables a more accurate demonstration of the EMS capabilities before it is deployed. The EMS digital twin will be optimized at each time step by a genetic algorithm (GA) to determine the optimal power flow that minimizes line losses and overall demand. The architecture of the deep neural network was optimized to reduce mean absolute error for both the training and

validation data by a GA that tested different neuron counts, dropout levels for 3 layers, and bounds for the randomly generated weights of the neural network.

ACKNOWLEDGEMENTS

To my Mom and Dad for their never ending love and support.

Special thanks to Dr. Tarek Zohdi for his guidance and teaching, as well as sharing his work so I could explore my own interests. E150 is why I am here today. I would like to also extend gratitude to Dr. Van Carey whose course, Machine Learning Tools for Modeling Energy Transport and Conversion Processes, provided me with the tools necessary for the deep learning section of this thesis.

Finally thank you to whomever is taking the time to read this. This paper was a challenge as I enter the realm of computational engineering with an unfamiliar background, but I am eager for what lies ahead.

Contents

LIST OF FIGURES

LIST OF TABLES

“‘

Chapter 1

Introduction

With internet usage accelerating in recent years [1], the need for 'data centers' and the energy to support them has grown with it. Following Zohdi [2], we define 'data centers' as building spaces dedicated to housing computing systems to process data, telecommunications, high performance computing devices, and supporting equipment. Cooling for these centers is extremely energy intensive. However, the volume of data centers is only increasing, creating a need for more efficient methods of cooling as well as energy usage. Energy management systems (EMS) present one possible solution to managing the energy supply for a hub of data centers. EMS refer to frameworks that control the energy generation, transmission, and storage for multiple coupled devices [3]. These systems are scalable, capable of controlling from electric vehicles to national grids. The goal for an EMS is to fulfill the power needs of every device while minimizing line losses and demand from fossil fuels dependent suppliers of energy. By pairing a digital twin of an EMS with multiple data center digital twins, we can simulate and demonstrate the effectiveness of an EMS in reducing energy consumption. We define digital twins as physically accurate models of real systems that can be utilized for real time control and optimization. Modeling an EMS with devices that accurately model the behaviour of data centers is challenging, given that the models for each data center require their own simulations, which can be computationally expensive.

To address the computational needs of the data center model, an artificial neural network (ANN) will be written to provide a reduced order model for the data center's energy consumption. This will allow for the devices in the EMS to accurately mimic the data centers without drastically slowing the EMS simulation.

1.1 Motivation

Data centers, which are growing in energy consumption with each passing year [3], are hindering the goal to scale renewable energy technology to support the entire national grid. Renewable energy power sources are still unable to fully support the U.S. power grid, and increasing the nation's power requirements pushes us further from reaching this standard. Data centers are essential to the massive amounts of information and data processing required across all areas of society, from national security to medicine. However, it is also essential that we address climate change, which has grown to be one of the most pressing issues for U.S. security as rising temperatures drive stronger, deadlier weather events. The two issues presented:

1. A need for increased computing capacity through more and/or more powerful data centers

2. A necessity to drastically reduce greenhouse gas emissions

seem to clash, where efforts to be more efficient could reduce computing capabilities and methods to expand data center capabilities can draw even more energy from the grid. This work will demonstrate an approach to address both issues through computational methods.

Constructing, expanding, and improving the performance of data centers is necessary to keep up with the global increase of digital information. The electronics operate constantly and their cooling systems require large inputs of energy to prevent overheating. Data centers represent about 1.5% of total U.S. energy consumption, and that number is still growing [4]. This creates a need to:

1. improve the efficiency of data center cooling systems and reduce required energy

2. develop control methods for energy flow through power networks to minimize demand.

With the second item, we consider an EMS model, and the opportunity it poses to incorporate renewable technologies. The EMS allows for the control of power flow in a system of interconnected devices, where devices are capable of acting as generators, loads, or both. Many data centers are buildings with flat, large roof spaces and built on large plots of land where solar and wind could potentially act as significant sources of the total required energy [5]. This allows for data centers to be treated as both generators and loads in an EMS model. With an EMS that accounts for line losses, the most efficient flow of power can be determined heuristically through a genetic algorithm that serves as a control for the system.

The power generation and consumption of data centers can be determined at each timestep through a thermofluids model. A model that utilizes the Navier-Stokes equations and first law of thermodynamics for voxel-based, iterative solutions was developed in [2]. However, running these calculations for multiple data centers is computationally expensive. To accelerate the speed of the EMS simulation, an ANN will be used to reduce the order of the thermofluids model while maintaining its accuracy.

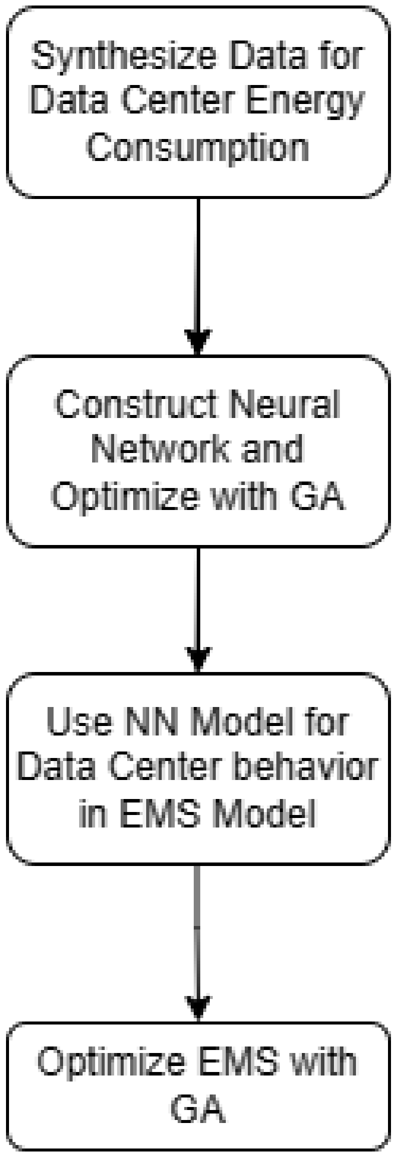
Synthesize Data for Data Center Energy Consumption
Construct Neural Network and Optimize with GA
Use NN Model for Data Center behavior in EMS Model
Optimize EMS with GA

Figure 1.1: General Outline of framework.

1.2 Machine Learning Challenges

ANNs are a great tool for nonlinear multivariate regression, but require careful design and can be difficult to comprehend, even for the experts [6]. They are also vulnerable to overfitting the data they are trained on and producing models that cannot be applied across the entire realm of data they are tested for. To address this, a genetic algorithm (GA) will be coded to optimze the aid in the design of the ANN so that it properly fits to the data and produces an accurate yet robust model.

Chapter 2

Digital Twins for Energy Management System and Heat Transfer in a Data Center

2.1 Energy Management System

Following Zohdi [3], we develop a model for a multi-device system where each device is connected and capable of transferring energy. A single supplier is geographically centered and is capable of delivering and receiving energy to and from any device. For the original model, devices randomly consume and generate electricity. Power is measured and then distributed amongst devices before the system turns to the supplier.
Energy states of each device are denoted D_i, $i = 1, 2, ..., N$. The energy state (Joules) of each device at time $t + \Delta t$ is equal to the energy state of the device at time t plus the total flux of energy from time t to $t + \Delta t$, denoted as $\Delta E_i^{tot}(t \to t + \Delta t)$

$$D_i(t + \Delta t) = D_i(t) + \Delta E_i^{tot}(t \to t + \Delta t) \tag{2.1}$$

We then subtract $D_i(t)$ and divide by Δt for both sides

$$\frac{D_i(t + \Delta t) - D_i(t)}{\Delta t} = \frac{\Delta E_i^{tot}(t \to t + \Delta t)}{\Delta t} \tag{2.2}$$

Taking the limit as $\Delta t \to 0$ yields the "power equation" for each device

$$\frac{dD_i}{dt} = \frac{dE_i^{tot}}{dt} = P_i^{tot} = TotalPower \tag{2.3}$$

2.1.1 Components of Power

P_i^{tot} is governed by the following parameters:

- G_i is the power generated
- C_i is the power consumed
- $F_{i\leftrightarrow j}$ is the net flux between devices i and j, with the following conditions:
- If $F_{i\leftrightarrow j} > 0$, then power is sent to device j and $\alpha_{ij} = 1$, with a loss at j
- If $F_{i\leftrightarrow j} \leq 0$, then power is sent from the device j and with a loss at receiver i, $\alpha_{ij} = e^{-\kappa d_{ij}}$, where κ is power loss per unit length and d_{ij} is the distance between device i and device j.

Putting these factors together, we derive

$$\frac{dD_i}{dt} = P_i^{tot} = G_i - C_i - \sum_{j=1}^{N} F_{i\leftrightarrow j}\alpha_{ij} + P_{s\leftrightarrow i} \tag{2.4}$$

integrating with time

$$D_i(t + \Delta t) = D_i(t) + \Delta t(G_i - C_i - \sum_{j=1}^{N} F_{i\leftrightarrow j}\alpha_{ij} + P_{s\leftrightarrow i}) \tag{2.5}$$

Loss, regardless of where it occurs, is governed by

$$\mathcal{L}_{ij} = \|F_{i\leftrightarrow j}\|(1 - e^{-\kappa d_{ij}}) \tag{2.6}$$

$P_{s\leftrightarrow i}$, the power sent between supplier and device i, is dependent on the needs of the device after exchanges of energy with other devices

$$\gamma_{si} \overset{\text{def}}{=} \frac{dD_i}{dt} - (G_i - C_i - \sum_{j=1}^{N} F_{i\leftrightarrow j}\alpha_{ij}) \tag{2.7}$$

where

- If $\gamma_{si} > 0$, then power needs to be sent to device i from the supplier
- If $\gamma_{si} \leq 0$, then power will be sent from device i to the supplier

2.1.2 Targeted Energy Supply

At each time step the target value is randomly set to $D_i^*(t + \Delta t)$. Therefore:

- If device i is in an energy deficit that must be fulfilled by the supplier, S, then, accounting for the losses, we may write

$$P^*_{s\leftrightarrow i} \overset{\text{def}}{=} P^{*,+}_{s\leftrightarrow i} e^{-\kappa d_{si}} = \frac{D_i(t+\Delta t) - D_i(t)}{\Delta t} - (G_i - C_i - \sum_{j=1}^{N} F_{i\leftrightarrow j}\alpha_{ij}) \tag{2.8}$$

  with line losses of

$$\mathcal{L}_{ij} = \|P^{*,+}_{s\leftrightarrow i}\|(1 - e^{-\kappa d_{si}}) \tag{2.9}$$

- If there is excess energy in unit i, its transfer to the supplier can be denoted as

$$P^*_{s\leftrightarrow i} \overset{\text{def}}{=} P^{*,-}_{s\leftrightarrow i} e^{-\kappa d_{si}} = \frac{D_i(t+\Delta t) - D_i(t)}{\Delta t} - (G_i - C_i - \sum_{j=1}^{N} F_{i\leftrightarrow j}\alpha_{ij}) \tag{2.10}$$

  with line losses of

$$\mathcal{L}_{ij} = \|P^{*,-}_{s\leftrightarrow i}\|(1 - e^{-\kappa d_{si}}) \tag{2.11}$$

  where $P^{*,-}_{s\leftrightarrow i}$ leaves device i, then arrives at the supplier as $P^{*,-}_{s\leftrightarrow i} e^{-\kappa d_{si}}$.

This is computed for each device at every time step. Energy exchange with the supplier is dependent on exchanges with surroundings.
These simulations can be computed faster than running in real time, making them ideal as a design tool that can be optimized to determine ideal parameters for the system.

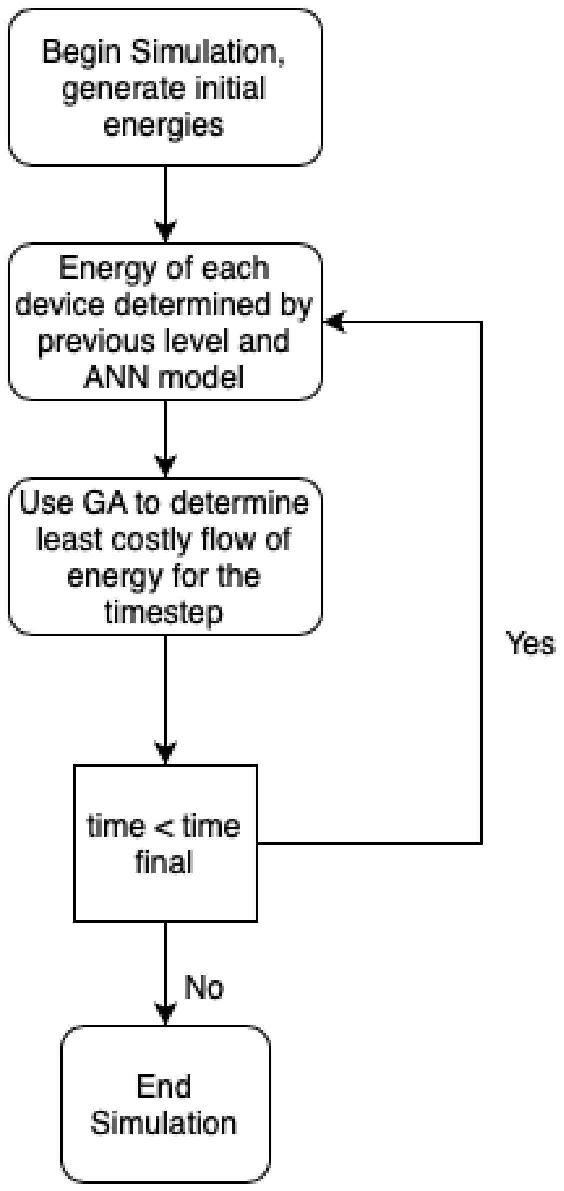
Begin Simulation, generate initial energies
Energy of each device determined by previous level and ANN model
Use GA to determine least costly flow of energy for the timestep
time < time final
Yes
No
End Simulation

Figure 2.1: Overall outline for the EMS simulation and optimization.

2.2 Governing Equations for Heat Transfer in Data Center

We refer the reader to Zohdi [2] for the steps and derivations of the heat transfer model for the data center. Synthetic data for the neural network is based upon results from this model.

Chapter 3

Deep Learning for Reduced Order Modeling of Data Centers

In this section we explore the application of a neural network constructed using the Keras and TensorFlow libraries in Python to model data center behavior to make for a more realistic simulation of the EMS. Since the thermofluids model coded by Dr. Zohdi to determine data center energy consumption is too computationally expensive to run at every time step of the EMS simulation, an artificial neural network (ANN) is trained to replicate the data center behavior. The ANN was trained on synthetic data since the thermofluids model was not coded to account for varying levels of energy input. However, the synthetic data was scaled to match the energy output of the thermofluids simulation. A GA was also coded to determine optimal neuron counts, dropout levels, and weight boundaries for each layer of the ANN.

3.1 Artificial Neural Network

Following Zohdi [cite 193], ANNs are defined as layered input output frameworks for adaptive nonlinear regressions of the form:

$$\mathcal{O} = \mathcal{B}(I_1, I_2, ..., I_M, b_1, b_2, ..., b_N, w_1, w_2, ..., w_N), \tag{3.1}$$

where $\mathcal{B}$ is the neural network, I_N represents inputs, b_N represents biases, and w_N represents the weight of each synapse. $\mathcal{O}$ is the output of the neural network and can be a single or multi-variable vector. We define:

- Synapses: linear functions that multiply incoming inputs by weights and add biases to calculate relevance of the input to the final output

- Neurons: neurons accept results from all incoming synapses and employ activation functions to scale the results within a certain boundary (typically 0 to 1 or -1 to 1)
- Training: the calibration of weights and biases to achieve the desired output
- Dropout: a regularization method where neurons are randomly dropped from each layer based on the ”dropout” probability.
- MinVal and MaxVal: inputs to the Keras.initializers [7] function that, respectively, determine lower and upper boundaries for random generation of weights as the ANN is trained
- Input and Output layer: the first and final layers to the neural network. In this model there are 2 inputs in the input layer and 1 output
- Hidden Layers:layers in between the input and output layers. This model features 3 hidden layers.

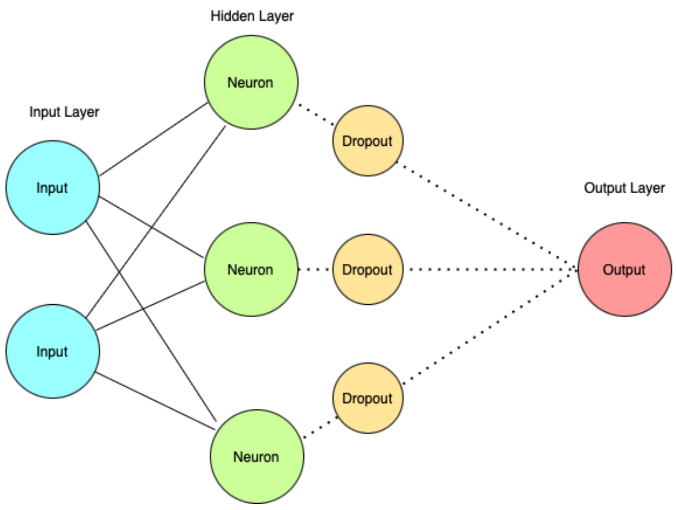

Figure 3.1: Basic ANN structure. Lines are dashed from the hidden layer to the output because there could be multiple hidden layers. Only one was illustrated for simplicity.

3.1.1 Synthetic Data

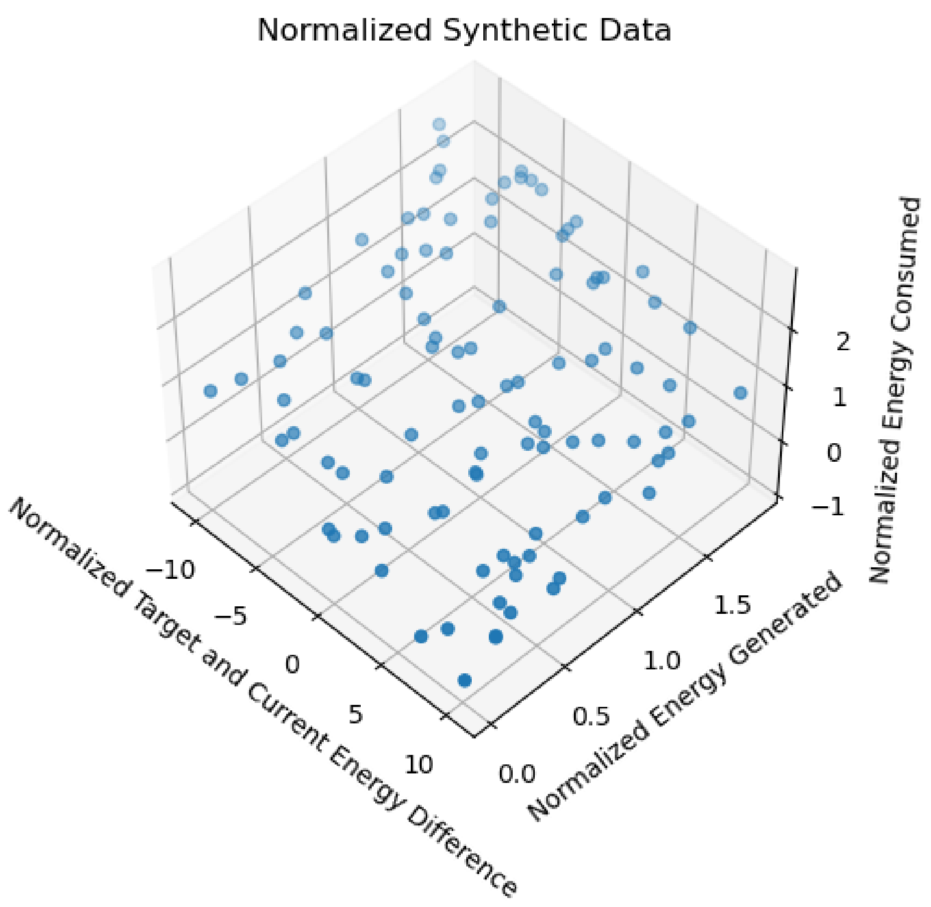

Figure 3.2: Synthetic Data normalized by the median for each input and output.

100 data points for the inputs were randomly generated. Synthetic data for the difference in target energy level and current customer state was subtracted from the synthetic data for the energy generated to produce the output of energy consumed. This was done so that there would be some pattern/linearity that the ANN could attempt to track down and minimize its mean absolute error.

3.1.2 Overfitting and Underfitting

When training ANNs, large sets of data are required. An oversimplified way to think of an ANN is a nonlinear regression. We define the mean absolute error (MAE) as:

$$\text{MAE} = \frac{\sum_i^n ||\mathcal{O}_{Des,i} - \mathcal{O}_i||}{n} \tag{3.2}$$

for a set of n data points where $\mathcal{O}_{Des,i}$ is the desired output for data point i and $\mathcal{O}_i$ is the output of the ANN. To produce a regression that minimizes the MAE, it would be ideal to make a large ANN that fits tightly to the data it trains on. However, this ANN would not be robust and likely produce high MAE when introduced to new data that it was not trained on. To better understand this, it is important to comprehend the notion of "noise" in data and the problem it poses in machine learning, and we refer the reader to Gupta and Gupta [8]. This is called overfitting the data, making an ANN that is too big and complex, focuses excessively on and noise, and is unable to function outside of its training data. Underfitting is the opposite, where the ANN is too small and overgeneralizes the behavior of the data. A critical step to ensure that an ANN is not over or underfitting is to split the data into a training set and a validation set. Common practice is to use $\frac{2}{3}$ to $\frac{3}{4}$ of the data for the training set and then the rest for the validation set. The ANN is trained with the training data and then confirmed to not be overfitting with the validation set. The goal is to minimize the MAE for both sets and for them to be approximately equal. Root mean squared error (RMSE) Dropout aids this by allowing the construction of a large ANN that can be "thinned" and have a multiple architectures calibrated to determine the most desirable structure [9].
The Keras and TensorFlow libraries utilize gradient descent methods to optimize the weights and biases of the ANN. However, we are still left to determine the optimal structure for the ANN, in terms of the neurons per layer and dropout levels following each layer.

3.2 Keras and TensorFlow Library

The Keras and TensorFlow motivated the decision to code this project in Python for their machine learning capabilities. The code for the ANN was based on the code provided by Dr. Van Carey in his machine learning course at UC Berkeley. Parameters for the Keras model of the ANN that remained constant throughout the project are are described and their values listed:

- RMSprop: Root mean squared propagation was the method of gradient descent used for optimizing the neural network to fit the training data. Set to .02 for all runs.

- Epochs: The maximum number of steps taken by the RMSprop when searching for the global minimum of the loss function. Set to 500.
- Activation function: Set to exponential linear unit (ELU), the activation function scales the outputs of the neurons and effectively turns a neuron ”on” or ”off”. For more on ELUs we refer the reader to [10].
- Initializer: method for selecting the initial weights before they are adjusted through RMSprop. Here the initializer is a random uniform distribution between MinVal and MaxVal.

Chapter 4

Genetic-Based Machine Learning Optimization

4.1 Genetic Algorithm

For functions that are non-convex and non-smooth, it becomes challenging to employ gradient based methods for optimization. These methods are vulnerable to converging on local minima for non-convex functions. This leads to search methods that are heuristic. Genetic algorithms (GAs), which follow a "survival of the fittest principal" for creating and testing designs, are well suited for this approach when paired with a cost function that can be rapidly computed.

Following Zohdi [2],[3],[11] we outline the general structure for a GA:

1. **Populate:** Generate a parameter population of genetic strings that represent design variables: $\mathbf{\Lambda}^i$
2. **Simulate:** Determine performance of each genetic string by plugging strings into a function or model that outputs a cost: $\Pi = \Pi(\mathbf{\Lambda}^i)$
3. **Evaluate:** Rank the strings $\mathbf{\Lambda}^i, i = 1, ..., S$ from lowest to highest based on determined cost, $\Pi(\mathbf{\Lambda}^k)$
4. **Mate:** Numerically combine the top performing pairs of strings ($i = 1, ..., P$),

obtaining 2 children per string:

$$\lambda^i \overset{\text{def}}{=} \mathbf{\Phi} \circ \Lambda^i + (1 - \mathbf{\Phi}) \circ \Lambda^{i+1}$$

$$\overset{\text{def}}{=} \left\{ \begin{matrix} \phi_1 \Lambda_1^i \\ \phi_2 \Lambda_2^i \\ \phi_3 \Lambda_3^i \\ \cdots \\ \phi_N \Lambda_N^i \end{matrix} \right\} + \left\{ \begin{matrix} (1-\phi_1)\Lambda_1^{i+1} \\ (1-\phi_2)\Lambda_2^{i+1} \\ (1-\phi_3)\Lambda_3^{i+1} \\ \cdots \\ (1-\phi_N)\Lambda_N^{i+1} \end{matrix} \right\}$$

and

$$\lambda^{i+1} \overset{\text{def}}{=} \mathbf{\Psi} \circ \Lambda^i + (1 - \mathbf{\Psi}) \circ \Lambda^{i+1}$$

$$\overset{\text{def}}{=} \left\{ \begin{matrix} \psi_1 \Lambda_1^i \\ \psi_2 \Lambda_2^i \\ \psi_3 \Lambda_3^i \\ \cdots \\ \psi_N \Lambda_N^i \end{matrix} \right\} + \left\{ \begin{matrix} (1-\psi_1)\Lambda_1^{i+1} \\ (1-\psi_2)\Lambda_2^{i+1} \\ (1-\psi_3)\Lambda_3^{i+1} \\ \cdots \\ (1-\psi_N)\Lambda_N^{i+1} \end{matrix} \right\}$$

where ϕ_i and ψ_i are random numbers such that $0 \leq \phi_i \leq 1$, $0 \leq \psi_i \leq 1$, and $\phi \neq \psi$.

5. **Eliminate:** Eliminate M poorest performing genetic strings but keep the P parents and P children for the next generation (M + 2P = S).

6. **Repeat:** Repeat process with updated gene pool and new M random genetic strings until lowest cost is within a predetermined tolerance or limit of generations is reached.

Remark: ϕ and ψ are bounded between 0 and 1 to prevent "mutations" in the child strings. If we exceed these bounds then it becomes possible for the child strings to inherit traits that would not be expected from combining the parent strings. However, since the algorithm outlined already produces random strings to fill the remaining spots of the population for each generation, inducing mutations in the child strings is redundant.

Parents are maintained because it is possible for child strings to be inferior to their parents. Maintaining the best performing strings allows us to reduce computational expense since their costs are already known and ensure that the optimal cost never increases (if new strings can not outperform the best string then the best cost will simply remain constant). One option, that was not employed in this project but is under investigation by the author, is to re-adapt boundaries of the design parameters around the top performing string. Doing this narrows the realm of focus for the algorithm, allowing for less computation to find the minimum adjacent to the top performing string. However, if this minimum is not the global minimum and the search is narrowed, then it can become more challenging for new strings to find the global minimum.

4.2 Energy Management System

The objective of the optimization scheme is to minimize total line losses between devices, losses between supplier and devices, and demand on the supplier. The optimization occurs at each step within the simulation to determine the optimal flow of power for the time step. First we consider the losses incurred sending power between supplier and devices. After power distribution among devices, the device states are calculated and, if they are lacking energy relative to a target level, the supplier sends them the difference. If the the device has an excess of energy relative to a target level, then the device sends power back to the supplier. The total power either sent or received by the supplier is the third cost:

$$\Pi_1(\Lambda_{1,1}, ..., \Lambda_{N,N}) \stackrel{\text{def}}{=} \sum_{i=1}^{N} P^*_{s\leftrightarrow i}(\boldsymbol{\Lambda}_k, t)\Delta t \tag{4.1}$$

where

$$\boldsymbol{\Lambda}^k \stackrel{\text{def}}{=} \begin{bmatrix} \Lambda_{1,1} & ... & \Lambda_{1,N} \\ \vdots & \ddots & \vdots \\ \Lambda_{N,1} & ... & \Lambda_{N,N} \end{bmatrix}$$

with $\mathbf{\Lambda}^k$ representing a second order design tensor for the k^{th} design string. $\Lambda^k_{i,j}$ represents flow from device i to device j for design tensor k. Note: $\Lambda^k_{i,j} = -\Lambda^k_{i,j}$. Power sent between supplier and device i is determined by energy levels in each unit following the exchange amongst all devices.

For total losses incurred sending power between supplier and devices and inter-device interactions, we sum the total losses for the time step:

$$\Pi_2(\Lambda_{1,1}, ..., \Lambda_{N,N}) \overset{\text{def}}{=} \sum_{i=1}^{N} \mathcal{L}_{ij}(\mathbf{\Lambda}^k, t)\Delta t. \tag{4.2}$$

We also consider a penalty for any devices whose energy level falls below 0. For any device to have a measured negative energy:

$$\text{Penalty} \overset{\text{def}}{=} \sum_{i=1}^{n} |C_i| \tag{4.3}$$

Total cost is then

$$\Pi_{EMS} = w_1\Pi_1 + w_2\Pi_2 + \text{Penalty} \tag{4.4}$$

where w_1 and w_2 are user adjustable weights that can tune the value of the demand versus losses in the algorithm.

4.2.1 Genetic Algorithm for EMS

Π_{EMS} is a non-convex, non-smooth function in the design parameter space. These functions are challenging to optimize directly through gradient based methods. This leads to search methods that are heuristic, trading accuracy of the final solution for speed. Genetic algorithms (GAs) are well suited for this approach when paired with a function that can rapidly compute its cost. Since the simulation for the EMS is faster than real time, the GA is utilized. Following Zohdi [2],[3],[11], a general approach is followed as described in 4.2.

4.3 Artificial Neural Network

The objective of the optimization scheme for the ANN is to reduce mean absolute error (MAE) between the ANN and the training and validation data. 75% of the synthetic data was randomly assigned to the training set and 25% was randomly assigned to the validation set. I considered the MAE for both sets to be equally important with the intention to avoid overfitting the data to the training set. Low MAEs for both sets is indicative of a robust and accurate neural network model. The cost function for the ANN was:

$$\Pi_{ANN} = w_1\Pi_1 + w_2\Pi_2 + w_3\Pi_3 \tag{4.5}$$

where Π_1 and Π_2 are the MAE for the ANN against the training and validation data, respectively. To place an additional constraint that prevents the algorithm from focusing on the training MAE and ignoring the validation MAE, Π_3 is defined:

$$\Pi_3 = \frac{\Pi_2}{\Pi_1}. \tag{4.6}$$

This restrains Π_1 from becoming even an order of magnitude less than Π_2. w_1, w_2, and w_3 are user tunable weights that can be adjusted. However, they are equal for this problem as I want the model to achieve low MAEs on the same order of magnitude for both training and validation data.

4.3.1 Genetic Algorithm for ANN

Π_{ANN} is a non-convex, non-smooth function in the design parameter space. This leads us to once again employ a GA for the optimization of the ANN.

Chapter 5

Results and Discussion

All simulations discussed below are for 8 devices or "customers" that are randomly assigned connectivtiy values (i.e. which customers are able to exchange power and which are not) and bounded to a 20x20 meter grid.

5.1 Numerical Example of Energy Management System

A GA was used for the optimization of the EMS at each time step. An evolution for the average, parent average, and optimal cost for power flow in a single time step is shown in Fig 5.1. The parameters for the GA are listed:

- Number of design variables: 28
- Population size per generation: 12
- Number of parents kept in each generation: 4
- Number of children created in each generation: 4
- Number of randomly generated new genes in each generation: 4
- Number of generations: 10

Energy flux between devices represented the design parameters. All fluxes were equally bounded:

$$0 \leq \lambda_i \leq 10000. \tag{5.1}$$

The small size of the GA (few generations and small population size) was to reduce computation and enable the algorithm to effectively optimize at each timestep. While most GAs are used to optimize entire simulations, this one optimized the simulation at each timestep, which can significantly increase the computational expense. Although the GA was small, it was still effective at reducing cost.

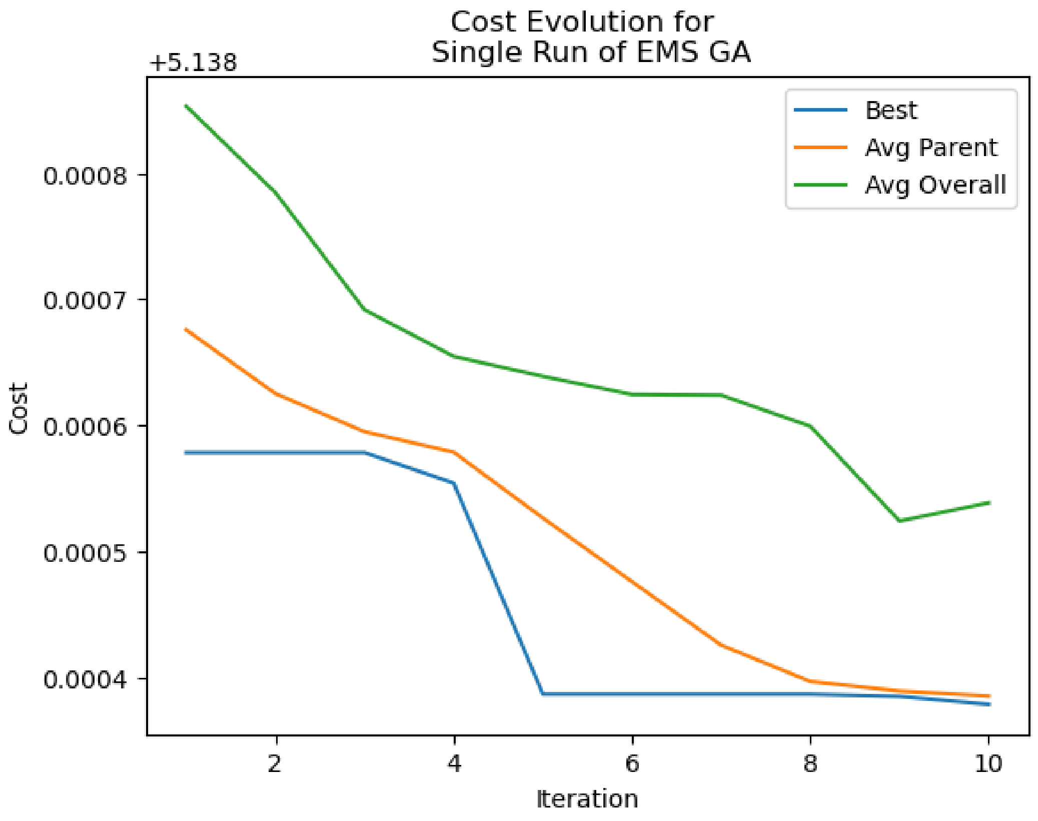

Figure 5.1: Cost evolution for the optimization of power flow in the EMS at a single time step.

This evolution occurred on every timestep of the final EMS simulation. The difference may at first seem small, but when added up for 100 timesteps, the cost becomes remarkable.

Time and Cost for EMS with a GA versus EMS without GA

Run	Time [s]	Cost	Time [s]	Cost
1	3.2	9.6933	0.1	28.3668
2	3.3	11.1842	0.0	30.4891
3	3.2	10.0059	0.0	34.1462
4	3.2	7.9022	0.0	33.1212
5	3.1	13.4517	0.0	32.3451

Table 5.1: Time and cost values for 100 time steps (time step = 1 second) of the EMS with a GA (second column to the left and middle column) and EMS simulation without a GA (right two columns).

5.1.1 Penalty

Initially, the EMS simulations were producing penalties on the order of 1000000 and dominating the cost function. Ultimately, it was found that the random method for sending power as used in [3] did not account for the energy difference among devices nor the physical impossibility of negative energy. One modification that helped the EMS achieve these significantly lower costs was an algorithm that prevented the power fluxes from devices that were low on energy or to devices of higher energy.

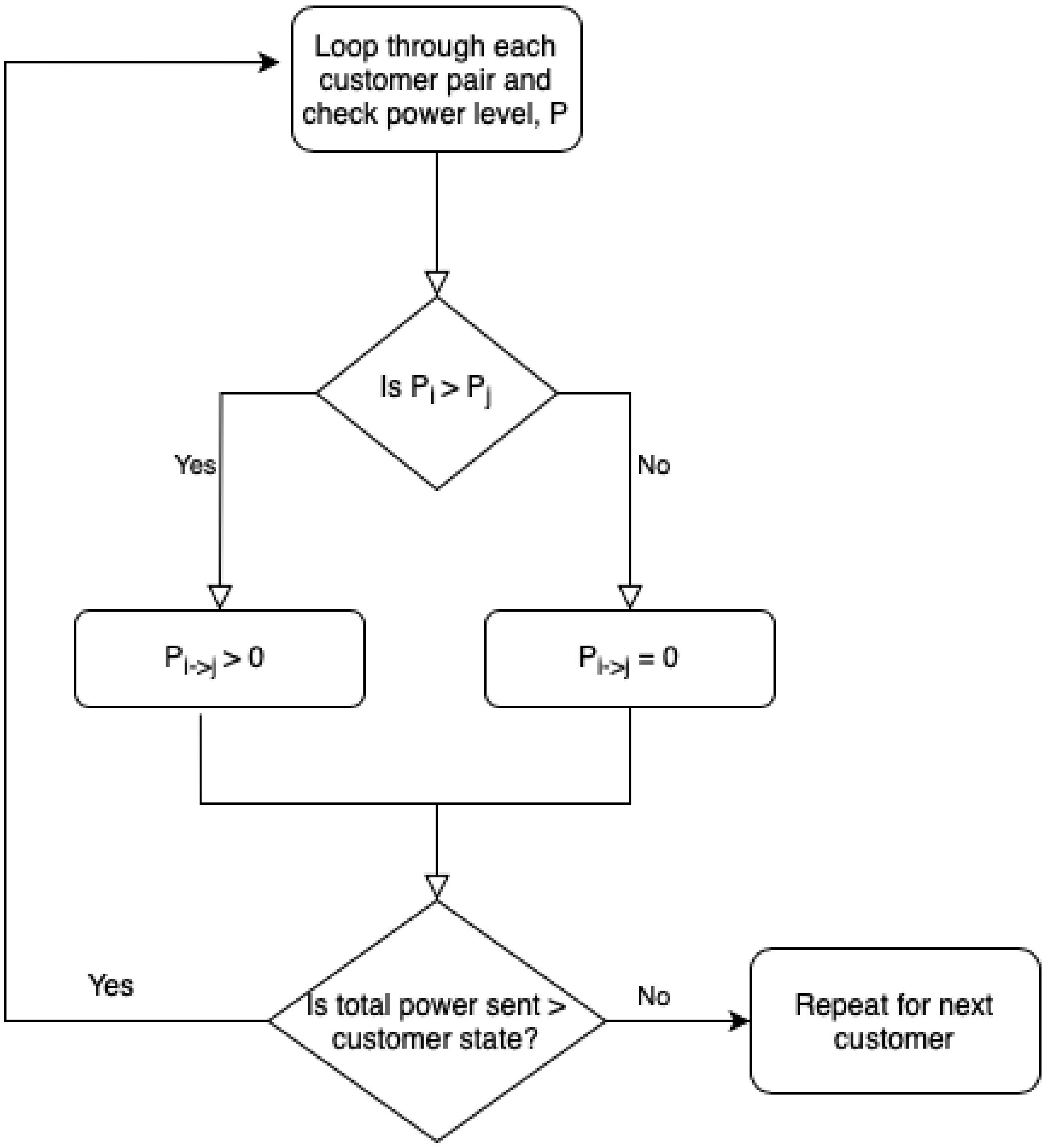
Loop through each customer pair and check power level, P
Is $P_i > P_j$
Yes
No
$P_{i\text{->}j} > 0$
$P_{i\text{->}j} = 0$
Yes
Is total power sent > customer state?
No
Repeat for next customer

Figure 5.2: Algorithm to reduce penalty for devices in the EMS.

The optimization scheme for the EMS proves to be orders of magnitude more beneficial to device health rather than random assignment of inter-device power fluxes. Energy states of devices are compared before power is sent which helps the EMS with the GA ensure it does not extract too much power from one device. The penalty tends to dominate the cost function, however, I believe it to be necessary as excessive discharging of an energy storage unit can increase fatigue and shorten its lifespan. While this paper focuses on reducing energy consumption, it is also important to consider materials, as many renewable technologies are coming to rely on critical metals [12].

5.2 Artificial Neural Network Optimization

The structure of the ANN that we are able to adjust numerically through Keras include the MinVal and MaxVal for initial weight selection, neuron counts in the hidden layers, and dropout levels following the hidden layers. These parameters make up the design string. The same method of optimization is applied for the neural network with some slight modifications to the specific parameters.

- Number of design variables: 8
- Population size per generation: 18
- Number of parents kept in each generation: 6
- Number of children created in each generation: 6
- Number of randomly generated new genes in each generation: 6
- Number of generations: 15

and the parameter ranges:

- MinVal: $-0.7 \leq \lambda_1 \leq 0$
- MaxVal: $0.1 \leq \lambda_2 \leq 0.7$
- Neurons: $2 \leq \lambda_3, \lambda_4, \lambda_5 \leq 16$
- Dropout: $0 \leq \lambda_6, \lambda_7, \lambda_8 \leq \frac{1}{3}$.

Note: Neurons are limited to integers and rounded to nearest whole number during mating.

The size of this GA was once again limited by the computational expense. Eventhough this GA is relatively small, it took over 2 hours to run on a MacBook Pro. The entirety of this project was coded in Python which, while one of the best performing languages for machine learning given its many libraries, is quite slow. It becomes very slow when compared to languages like C and Fortran. Besides other languages, another step to improving this issue is identifying areas of the code that can be parallelized.

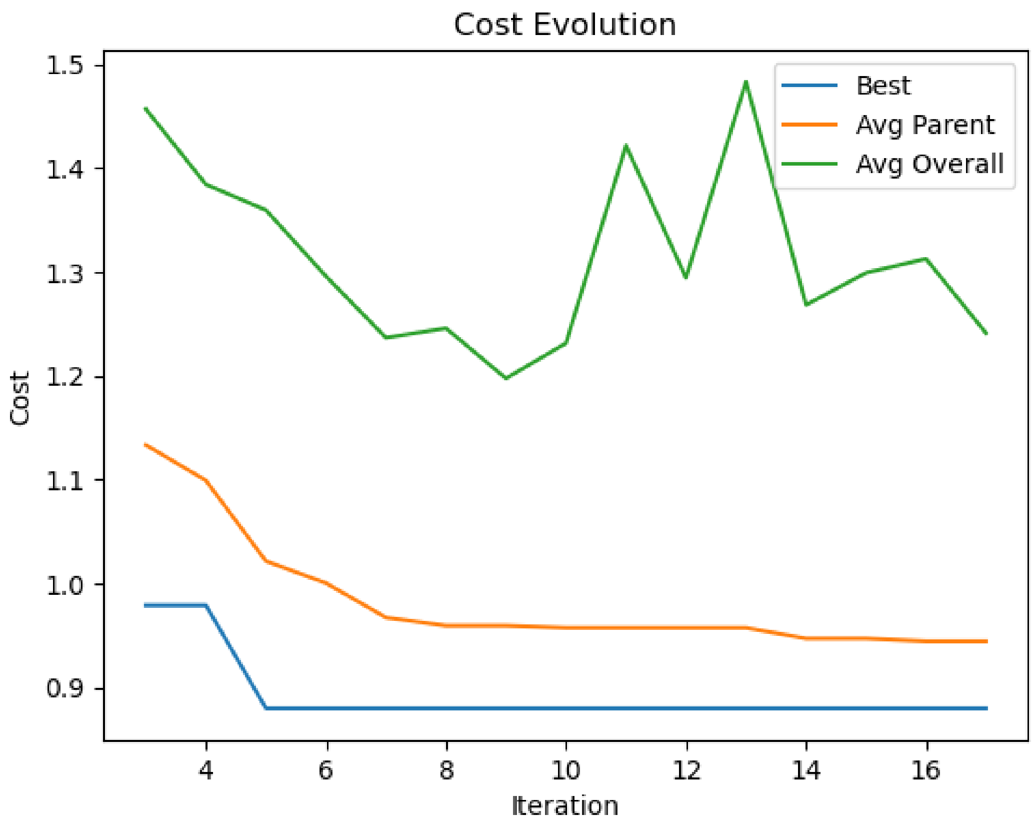

Figure 5.3: Cost evolution for the optimization of the ANN structure.

Given the cost function for the ANN 4.5, it was challenging for the cost to fall below 0.88. Π_3 in the cost function limits the cost from falling far below 1, as it is extremely unlikely for there to be a significant difference in the MAEs of the training and validation data that favors the validation set. In summary, the GA started off with an extremely optimal set of parameters, and there was not much room for improvement.

Design	MinVal	MaxVal	Dropout	Dropout	Dropout	Neuron	Neuron	Neuron	Cost
1	-3.187e-1	4.314e-1	6.281e-2	1.103e-1	9.933e-3	5	2	6	8.799e-1
2	-2.018e-1	5.267e-1	2.644e-1	2.634e-1	3.253e-1	13	10	8	9.164e-1
3	-1.650e-1	4.395e-1	3.174e-1	2.927e-1	2.645e-1	12	4	2	9.362e-1
4	-1.335e-1	2.330e-1	1.839e-1	1.770e-1	1.884e-1	5	5	8	9.716e-1

Table 5.2: Table of best performing strings and costs for neural network structure. Dropout and neuron counts are ordered for layers 1-3 from left to right.

After the parameters of the optimal design string were recorded, they were plugged back into the Keras model to develop the final working ANN that was applied in the EMS simulation. After four runs of the RMSprop Keras Optimizer, the following MAEs were achieved:

Training MAE	.06436
Validation MAE	.04871

Table 5.3: Mean Absolute Error values of the optimized neural network structure

Remark: While the cost function indicates an MAE for the validation data is less than the training, the exact model was not saved in the GA. It was recreated with the same boundaries for weights, neuron counts per layer, and dropout. Therefore it is possible for a validation MAE greater than the training MAE.

Below we include plots of predicted output by the model against training data and validation data:

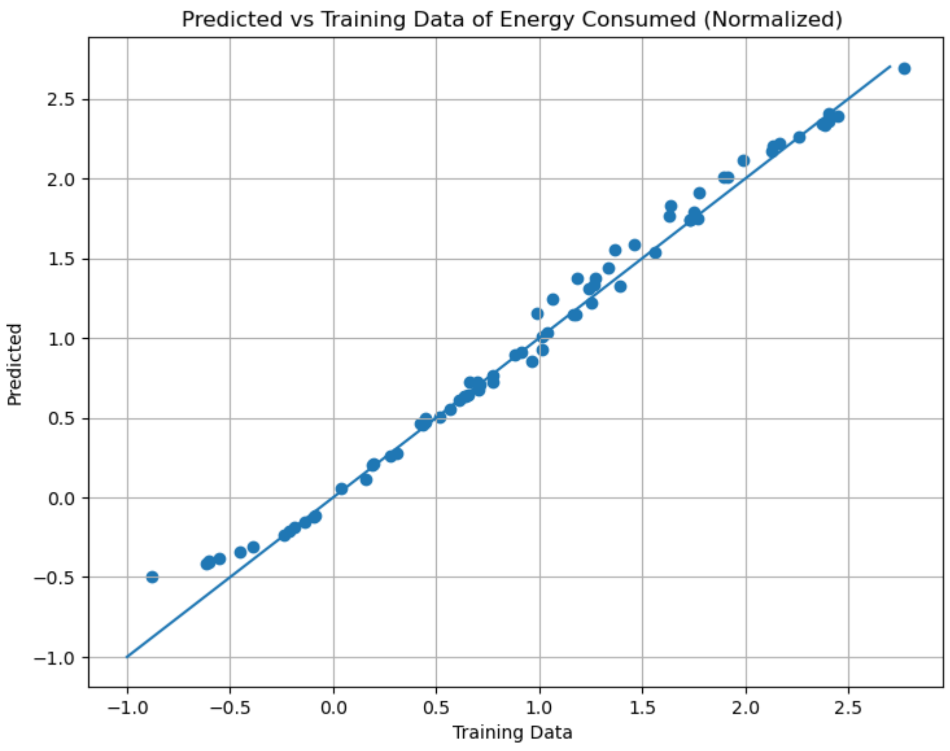

Figure 5.4: Energy consumed as predicted by the model against training data. Line through points is y = x.

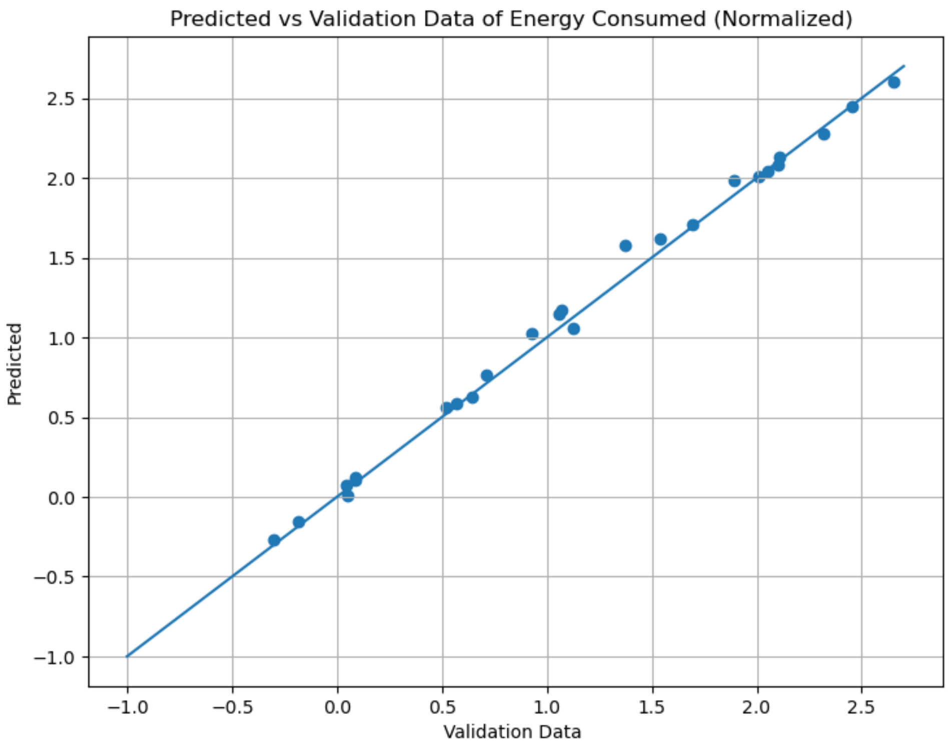

Figure 5.5: Energy consumed as predicted by the model against validation data. Line through points is y = x.

It is important to note the straying of data near the lower limit from the y = x line for the training data. This indicates the model is not well trained for the lowest levels of energy consumption.

5.3 Coupled EMS with ANN

Finally, we bring the optimized EMS and ANN together. The ANN serves as a predictor for energy consumption by the data centers to bring the digital twin closer to reality. For the purpose of real time control, it would be ultimately best to feed data live to the EMS digital twin to mitigate the need of excess computation. However, if simulations are necessary

Time and Cost for EMS with a GA versus EMS without GA

Run	Time [s]	Cost
1	43.2	11.0694
2	41.6	15.4782
3	44.2	11.0129
4	40.3	11.3520
5	40.0	15.6779

Table 5.4: Time and cost values for 100 time steps (time step = 1 second) of the EMS with a GA and ANN for reduced ordered model of data center behavior.

Note the drastically longer run times the ANN brings onto the simulation. While still orders of magnitude faster than an actual multiphysics model, this slow down in run time prevents this digital twin from being useful as a controller. However, for testing that does not require real time speed, this model becomes more desirable as there is some logic to the generation of energy; all that is left is to acquire real data to replace the synthetic set used here. It would become more applicable for simulation and testing outside of real time if we consider other data driven models, such as renewable power generation. It is hypothesized that the slight increase to the cost is a result of the ANN producing abnormal levels of consumed energy after receiving data that was at the bounds of its training set.

Chapter 6

Concluding Remarks

6.1 Summary

In this paper the control of an EMS through optimization with a genetic algorithm is explored as an option for reducing the energy consumption of a data center campus. Since real data was unavailable and the available model for a data center was too slow for optimization of an entire campus, an artificial neural network took its place as a reduced order model. This method is well-suited for the rapid optimization of complex multivariate designs, so long as sufficient data is available to train the neural network. Synthetic data was used in this paper because the data center model was not designed for the exact input/output of energy production that was best suited for the EMS, but adjusting this model accordingly is being considered for future work. However, a more urgent priority is to convert this code from Python to a low-level language, such as C or Fortran, and search for opportunities to parallelize the code. This is necessary as these changes will speed up the code by several orders of magnitude. If a digital twin-optimization framework, such as this, is to be used for an EMS, then the framework must run as fast as possible to find the most desirable control scheme and manage the flow of power in real time. One lacking portion in the paper is the size of the GA for the optimized EMS simulation. It was small, which allowed for the simulation to run faster than real time, but prevented the GA from fully exploring the design space. Even as it was faster than real time, it was still not sufficiently quick enough for application. For a real EMS, energy control needs to be instant, and time steps of a second are far too large for a digital twin approach. One intriguing control scheme for EMS is reinforcement learning (RL) and we refer the reader to this comprehensive review of RL for control by Buşoniu, Bruin, Tolić, Kober, and Palunko [13]. RL does not rely on a model, simply feedback in the form of ”rewards” or ”punishments” from the system. In an EMS, rewards would include reduced power drawn from the supplier and punishments may include excessive line losses.

6.2 Advancing the Digital Twin

6.2.1 Power Generation

Energy consumption was modeled based on the power required for the electronics cooling necessary to mitigate thermal stresses for data processing units. However, energy generation was completely random for the EMS simulation. This is an unrealistic approach that can be improved by importing data for local weather patterns, particularly wind speed, cloud cover, and sun path. Sensors would be especially helpful for collecting and feeding this information so the EMS can predict what power generation sources to rely or whether it is time to tap into an energy storage unit. Energy storage methods, both modeling and optimization, are currently under investigation by Dr. Zohdi.

6.2.2 Energy Storage

Another key element to the advancement of renewable power is energy storage. From fuel cells to thermal energy storage, commercial to lab scale, it is a topic that spans science and industry. For renewable energy sources, like solar and wind, that do not provide a constant, reliable supply of power, storage is critical to making that source capable of supporting the grid [14]. Storage is also crucial when one considers the daily patterns in energy consumption. It is typically for their to be a spike in the late afternoon as people begin cooking, using HVAC, turning on lights, etc. [15]. These fluctuations were not accounted for in my model as I was only considering a data center campus that relies on a fairly steady power flux to ensure all electronics remain a within a narrow temperature range. For grid level management, it is necessary to be prepared for times when there is high power generation and low demand or vice versa. Most renewables are not capable of a steady supply or being able to exactly match the needs of the grid. While some forms of constant renewable power are being researched, such as nuclear fusion, they are years, if not decades, away from being commercialized. The climate crisis requires an immediate transition from fossil fuels, and energy storage will allow the technology that has already been commercialized to scale even further. Adding a component of storage to the EMS digital twin is a crucial next step for modeling behavior in more dynamic regions, such as metropolitan areas.

REFERENCES

[1] R. Price, "Internet growth accelerates and social continues to climb, but there are changes brewing," *Digital Content Next*, April 2019.

[2] T. Zohdi, "A digital-twin and machine-learning framework for precise heat and energy management of data-centers," *Computational Mechanics*, vol. 69, March 2022.

[3] ——, "An adaptive digital framework for energy management of complex multi-device systems," *Computational Mechanics*, vol. 70, 2022.

[4] G. Kamiya, "Data centres and data transmission networks," *IEA*, September 2022.

[5] A. Olivo, "Va. officials weigh letting loudoun data centers use diesel backup power," *The Washington Post*, March 2023.

[6] D. Beer, "Why humans will never understand ai," *BBC*, April 2023.

[7] "Layer weight initializers," https://keras.io/api/layers/initializers/, accessed: 2023-05-12.

[8] A. G. Shivani Gupta, "Dealing with noise problem in machine learning data-sets: A systematic review," *Procedia Computer Science*, vol. 161, p. 466.

[9] A. K. I. S. Nitish Srivastava, Geoffrey Hinton and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, 2014.

[10] T. U. . S. H. Djork-Arne Clevert, "Fast and accurate deep network learning by exponential linear units (elus)," in *International Conference on Learning Representations*, San Juan, Puerto Rico, May 2016.

[11] T. Zohdi, "A note on the rapid genetic calibration of artificial neural networks," *Computational Mechanics*, vol. 70, August 2022.

[12] L. Leruth and A. Mazarei, "Who controls the world's minerals needed for green energy?" *Peterson Institute for International Economics*, August 2022.

[13] D. T. J. K. I. P. Lucian Buşoniu, Tim de Bruin, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.

[14] A. D. Max Schoenfisch, "Grid-scale storage," *IEA*, 2022.

[15] "Hourly electricity consumption varies throughout the day and across seasons," https://www.eia.gov/todayinenergy/detail.php?id=42915#, accessed: 2023-05-12.