The Role of Data Science in Numerical Modeling of Lithium Ion Batteries

Neal Dawson-Elli

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Venkat R. Subramanian, Chair

Daniel T. Schwartz

David A. C. Beck

Program Authorized to Offer Degree:
Chemical Engineering

University of Washington

Abstract

The Role of Data Science in Numerical Modeling of Lithium Ion Batteries

Neal Dawson-Elli

Chair of the Supervisory Committee:
Professor Venkat R. Subramanian
Chemical Engineering

Batteries are complex electrochemical devices which are nearly ubiquitous in today's society. As the energy demands of mobile devices increase, the performance of batteries must also improve to keep pace. One of the key elements in iterative battery design is the application of numerical models which can predict the properties of potential batteries at significantly reduced cost compared to cell development, enabling high-throughput screening of potential materials and geometries. These models can vary in complexity from simple empirical fits, through continuum-scale models, up to molecular dynamics simulations, which offer increased fidelity, but at an extremely high computational cost.

In addition to first-principles models, data-driven models have become popular as the available computational resources and amount of available data have grown astronomically. These models use self-tuning algorithms which form highly accurate nonlinear mappings from inputs to outputs, providing excellent accuracy for relatively low computational cost at runtime. While traditional data-driven models can achieve impressive results given a large amount of data, the acquisition of data at the proper scale is expensive, does not generalize well to other use conditions or battery chemistries, and offers little guidance in the form of physical interpretability. In this work, combinations of physical models and data-driven models are utilized in order to provide highly accurate, flexible applications of the information contained within the first-principles models, while also significantly reducing computational

cost at runtime. While design applications of equivalent techniques are conceivable, this work focuses on applications for the calibration of first-principles models for the purposes of improved control of existing electrochemical cells.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I'd like to thank the many, many people who have helped make Seattle feel like home over the past four years. I'd like to start by thanking my committee members Venkat Subramanian (Chair), Dan Schwartz, and David Beck, whose advice and feedback have been invaluable.

An extra thank you to Venkat Subramanian for the mentorship and the opportunity to pursue my passions for embedded systems, data science, electrochemistry, and numerical methods.

Thank you to Magda Balazinska, David Beck, and other members of the eScience Institute, without whom the DIRECT and Advanced Data Science Options wouldn't have been possible. These programs, and Dave's teachings, helped me discover a love for software engineering.

Another huge thank you to all the friends who helped celebrate the ups and commiserate the downs of graduate school. To those who came before me (Matt, Barry, Honorio, Manan, Chintan, Seongbeom, Surya, Elena, Trevor, Yanbo, Josh, Wes, Evan), thank you for your guidance, friendship, and example. To those in my cohort (Ben, Ian, Chad, Chris, Caitlyn, Jon, Brian, Kelly, Kyle), those who joined after me (Victor, Erica, Caitlin, Luke, Mike, Luke, Akshay, Tae Jin, Linette), and those from back home (Bart, Joe, Kenny, Stevie, Lou), thank you for making the last few years great!

I'd like to thank my parents, Dave and Patty, and my sister Megan – thank you for the phone calls, the family trips, and the constant reminders of love.

To my twin brother, Alex – thank you for always having time to hear me vent about research, to talk about purpose, and to get excited about each others' projects. Especially in the trenches of graduate school, it's great to know I'm never alone.

To Sheila – I couldn't have done it without your constant support, sense of adventure, and love. Thank you for simultaneously inspiring me to work harder and reminding me that life happens outside of the lab. Your jokes, goofiness, and positivity are always a ray of sunshine, even on the cloudiest of Seattle days.

DEDICATION

To my parents, whose unending support and encouragement made this all possible.

Thank you for everything.

Chapter 1

INTRODUCTION

Lithium-ion batteries are complex electrochemical devices whose behavior is governed by a combination of thermodynamic, mass-transport, and kinetic properties. These properties can vary based upon the materials used, cell construction, and transient operating conditions. This sensitivity has led to the development of a variety of numerical models of varying physical relevance, from simple empirical fits to continuum-scale models and continuing to molecular scale simulations. These models offer varying abilities to predict the future performance of batteries, analyze the performance of existing batteries, and recommend property modification for improved performance in future batteries. In general, the usefulness of the model varies inversely with the computational resources needed to solve it - that is, the added fidelity associated with more sophisticated models comes at an increased computational cost which must be paid at runtime.

Two popular physics-based lithium-ion battery models include the single particle model[1] and the pseudo two-dimensional model[2,3,4,5,6,7], which will be more thoroughly discussed in chapters 5 and 2, respectively. These models aim to mathematically replicate the physical processes which occur during the charge and discharge of a lithium-ion battery, including volume-averaged Fickian diffusion, Butler-Volmer kinetics, and localized overpotentials. As a lithium-ion battery charges, the lithium ions must physically intercalate into the anode particles, and then penetrate deeper into the particles through diffusion. The lithium ions which are consumed at the surface of the particles must be replenished by the bulk electrolyte, again through diffusion. Batteries are ohmic devices, meaning that the voltage or current can be specified, but not both.

When seeking to optimize existing battery performance, the cell is treated as a black

box the voltage, current, and temperature can be monitored, but little else can be inferred directly. Constraints on these external states can enhance the performance of the battery allowing the voltage to extend below 2.5 V can damage the cell through irreversible reactions, and elevating the voltage too far above 4.2 V can degrade the electrolyte and risks setting fire to the cell. While these guidelines can prove useful, the localized overpotentials and thermodynamics of the cell dictate the rate of degradation mechanisms. Unfortunately, these metrics cannot be directly measured in pre-built cells. However, using these sophisticated models, it may be possible to predict these internal states and use them to inform control behavior.

1.1 Using Models for Design and Control

In collaboration with NREL, the MAPLE lab[3,8] experimentally calibrated a variant of the pseudo two-dimensional model with lithium precipitation kinetics to an existing cell. Lithium precipitation, or lithium plating, occurs at the negative electrode during charging when the plating overpotential becomes negative[9]. In order to minimize this effect, a charge paradigm was created which stored the maximum amount of charge in an allotted time while relaxing the 4.2V constraint and avoiding a negative plating potential. When this charge protocol was compared to a traditional constant-current, constant-voltage charge paradigm, which was applied across an identical charge window, the number of useful cycles was more than doubled. In essence, using the model to create control schema based on modeled internal variables was experimentally validated. However, the process of extracting these model constants experimentally was costly and time consuming, taking one year and requiring $300,000 in funding. These steps would need to be repeated for each new chemistry or geometry, limiting the viability to single-production-run cells.

Experimental parameter extraction is not the only method available sophisticated optimization algorithms exist which endeavor to minimize the error between two curves, which can be used to estimate the model parameters via fitting to experimental data. These physics-based models are nonlinear, stiff, and have parameter sensitivity which varies throughout

the parameter space[10], which makes fitting challenging. The three main concerns of the optimization problem are as follows: convexity of the problem, solution time of the model, robustness of the model within the parameter bounds. Given the accuracy and convergence constraints on the model, there are performance ceilings which can limit the speed with which an optimization problem can reach convergence. It is with these three components that data science may be able to help.

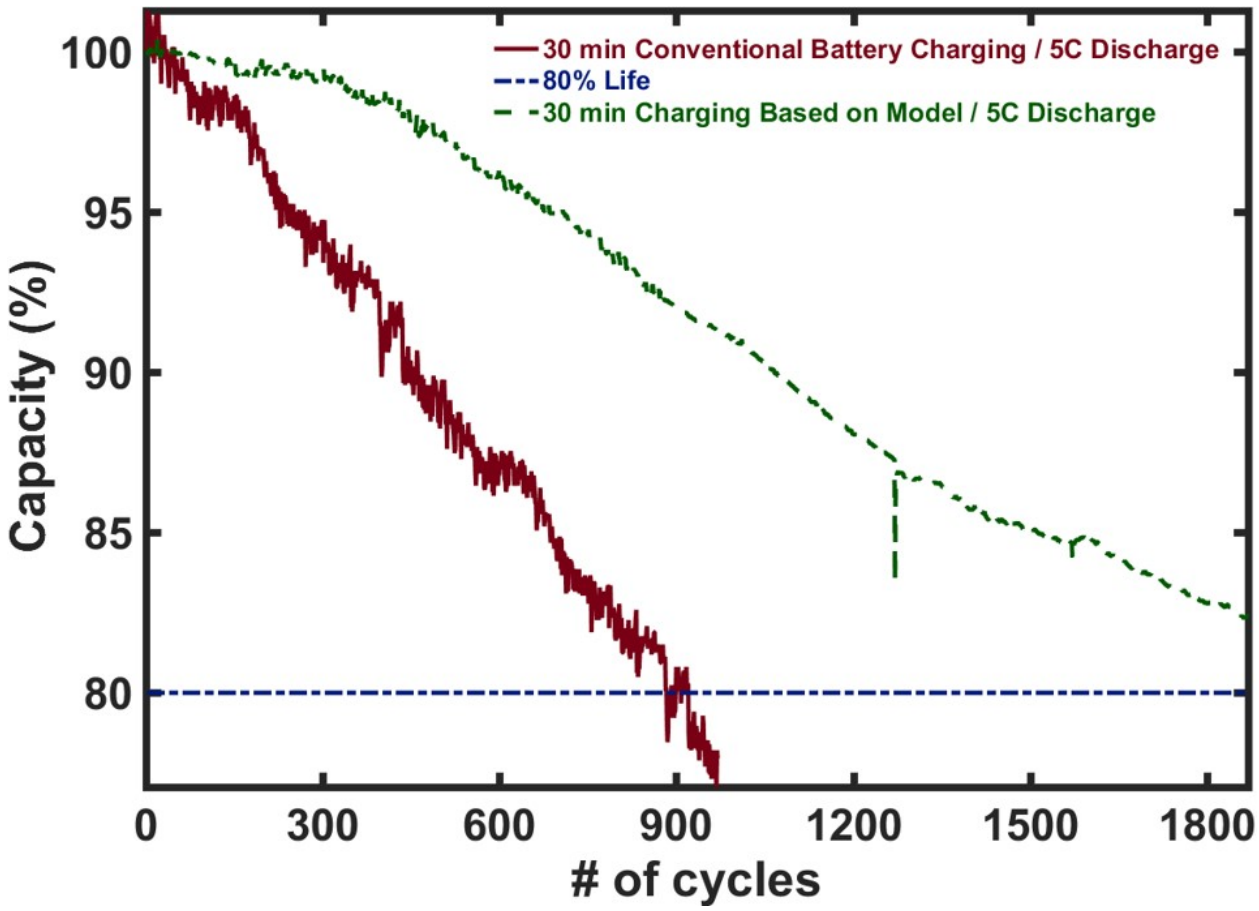


Figure 1.1: Model-based charging paradigm can store the same amount of energy in the same time constraints, but can extend cycle life by 2-fold. Reproduced here for clarity, originally sourced from previous works[3,8]

1.2 Electrochemistry + Data Science

Data science, hailed as the fourth paradigm of science[11], is a broad field which encompasses data visualization, data management, and machine learning. As data sets grow, it becomes increasingly more important that the management schema is scalable. While comma separated variable (CSV) and text files may be reasonable for files on the order of megabytes, they would present a significant challenge to analysis pipelines on the scale of hundreds of gigabytes or more. The intentional management of data in anticipation of reaching the limits of naive implementations of storage is one of the key aspects of data science. Towards this end, many tools have been developed, including the Hierarchical Data Format version 5 (HDF5), which offers an elegant way to store large quantities of data which can be represented by arrays. All of the projects in this thesis rely on HDF5 files in order to efficiently store the training data.

Machine learning algorithms, named for the ability to learn relationships between given inputs and outputs without explicit user manipulation, are powerful estimators which can be leveraged to create predictive models from simulated or experimentally collected data. Many works combine data science and experimental data in order to create data-driven models, capable of estimating state of charge[12,13,14,15,16], forecasting voltages[17], and estimating remaining useful life[18,19]. While these models are impressive in their own right, they still treat batteries as black boxes, which is not particularly useful for control. Given perfect information, a problem formulation as in other works[20] could act as an abstraction around a proper control paradigm, estimating the remaining useful life as a function of charge profile, for instance. However, the information is never perfect, and even with tools as powerful as those available to data scientists, extrapolating too far from the collected data can result in poor accuracy.

In terms of applications for control, the most likely subject for data science are expensive computational models, like the single particle model (SPM) and pseudo two-dimensional (P2D) model. These models give key insight into the internal states of the battery, which

have been demonstrated to be extremely useful in control, in particular for monitoring anodic overpotentials during charging in order to extend useful cycle life. However, there are often situations where they may be too expensive or complex to calibrate or implement on commodity hardware. By creating surrogate models using data science techniques, model evaluation cost can be decreased by several orders of magnitude, while accuracy remains above 99%[21,22].

While data science methods are powerful, the results of a project can benefit immensely from creativity and domain expertise. By leveraging a thorough understanding of the systems to be improved or replaced, much more generalized results can be achieved with less data. The recent push at the University of Washington to incorporate data science tools into the Chemical Engineering toolbox has the potential to increase the number of domain experts utilizing these tools, accelerating the pace of research. While large companies which approach infinite compute capability can master tasks without the need for human knowledge[23], those with much more limited computing resources will find that asking the right questions and applying the proper tools is a much more efficient process than teaching neural networks everything from scratch.

1.3 Thesis Roadmap

When examining the pain points in the current electrochemical model application work flow, there are many avenues that may be taken. Operating under the assumption that data science techniques can provide comparable performance with drastically reduced execution time, a surrogate model for the physics-based model is an obvious choice. However, other problem formulations exist which can take advantage of the ability of machine learning to map arbitrary inputs to outputs, with the performance of these techniques being a function of data size, sensitivity, and complexity of the problem. Some of the ways in which this modeled data can be leveraged to answer specific problems is discussed in Chapter 2.

In addition to replacing the physics-based model with something equally accurate but faster to compute, another system which can be augmented by data science is the optimizer

itself, which tends to struggle with nonlinear models such as the pseudo two-dimensional model. The pursuit of a viable data-science-based, problem-specific optimizer is analyzed in detail in Chapter 3, where neural networks are tasked with mastering the relationship between deviations in physics-based model output space and physics-based model input space. For instance, if positive electrode thickness is reduced, how does that affect the shape of the discharge curve? How do these results compare to modifying the negative electrode thickness? By leveraging the informational content in the time series, rather than abstracting to a value like root mean squared error, the neural network is able to significantly improve the fitting capabilities of traditional black box optimization algorithms.

While Chapter 2 highlights some of the difficulties in trying to use data science techniques to create surrogate models which offer one-off predictions of time series, such as large-scale data requirements and physical consistency considerations, Chapter 4 examines alternatives to this formulation which can alleviate some of these difficulties. In particular, custom loss functions[24] have been shown to improve the physical consistency of neural networks, but these loss functions are not always applicable. For instance, during a drive cycle, voltage is not monotonically decreasing, and so the custom loss functions which work in a discharge context will no longer be valid. By changing the problem formulation, and outsourcing the responsibility of physical consistency to mapping functions, teams of hybrid-ordinary differential equation and neural network systems can be leveraged to reproduce the result of the single particle model with high accuracy and decreased solve time.

Data science is an extremely powerful tool, with many of the most powerful algorithms able to leverage massive amounts of data. When it comes to data creation, especially for the creation of simulated data, it is paramount to ensure organization and coherence throughout the effort. Towards this end, a python package has been created that offers high-performance numerical simulations of batteries wrapped in an intuitive, Sci-Kit Learn[25,26]-style interface with many useful functions and methods. The work in Chapter 3 required 21 million simulations, which would have been impossible without this package. The details of the package are outlined in Chapter 5.

While many small projects have been outlined, there is always more work to be done and new frontiers to push. Chapter 6 will outline likely avenues of further improvement, focusing on the incorporation of domain knowledge and creative application of data science in order to improve the applicability of sophisticated analysis techniques.

Chapter 2

DATA SCIENCE APPROACHES FOR ELECTROCHEMICAL ENGINEERS - AN INTRODUCTION THROUGH SURROGATE MODEL DEVELOPMENT FOR LITHIUM-ION BATTERIES

Note: this chapter was published as an article

- Dawson-Elli, N.; Lee, S. B.; Pathak, M.; Mitra, K.; Subramanian, V. R. Data Science Approaches for Electrochemical Engineers: An Introduction through Surrogate Model Development for Lithium-Ion Batteries. J. Electrochem. Soc. 2018, 165 (2), A1A15. https://doi.org/10.1149/2.1391714jes.

Abstract

Data science, hailed as the fourth paradigm of science, is a rapidly growing field which has served to revolutionize the fields of bio-informatics and climate science and can provide significant speed improvements in the discovery of new materials, mechanisms, and simulations. Data science techniques are often used to analyze and predict experimental data, but they can also be used with simulated data to create surrogate models. Chief among the data science techniques in this application is machine learning (ML), which is an effective means for creating a predictive relationship between input and output vector pairs. Physics-based battery models, like the comprehensive pseudo-two-dimensional (P2D) model, offer increased physical insight, increased predictability, and an opportunity for optimization of battery performance which is not possible with equivalent circuit (EC) models. In this work, ML-based surrogate models are created and analyzed for accuracy and execution time. Decision trees (DTs), random forests (RFs), and gradient boosted machines (GBMs) are shown to offer trade-offs between training time, execution time, and accuracy. Their ability to predict the dynamic behavior of the physics-based model are examined and the corresponding execution

times are extremely encouraging for use in time-critical applications while still maintaining very high (99%) accuracy.

2.1 Introduction

Data science, also known as data-intensive scientific discovery, is hailed as the fourth paradigm of science[11] super. A field focused on extracting knowledge or understanding from data, it includes the subdomains of machine learning, classification, data mining, databases, and data visualization. In the age of internet-scale data, these techniques are not only powerful, but also necessary to extract the signal from the noise and to have the throughput to do so in a reasonable amount of time. It has revolutionized the fields of bio-informatics, climate science, word recognition, advertising, medicine, and is finding more applications daily. In Googles Translate application, substantial improvements over previous methods were achieved using artificial neural network (ANN) structures, making 60% fewer errors than the previous state-of-the-art algorithm[27]. In climate science, where models are sophisticated and numerous, data science techniques are used to determine which of 20 models will give the best prediction on future and historical data, the accuracy of which surpasses the accuracy of the average of all models, the current benchmark[28]. As chemical engineers are increasingly tasked with the analysis of more complex data sets, these same data science tools which have revolutionized other fields become more relevant[21].

When data sets grow, they must be managed intentionally in order to be useful. Data management, a subfield of data science, fills this role and gives the tools to be able to correct for missing data points, ensure consistency of the data, and transform the content of the data such that it is suitable for use in other aspects of data science. For missing data, several techniques exist, such as deletion of the sample, deletion of the feature column, assignment of the average value of the feature column, and creation of a predictive model to fill the gaps. Of these, the creation of a predictive model is of significant interest for electrochemical modelers, as it can result in a better performing model through the preservation of more information[29]. In addition to cleansing the data set of missing values, this step is often where exploration

and correlation is done, and is an opportunity for initial feature engineering to take place. Feature engineering, discussed in more detail in Section 4, is the practice of combining input features that highlight patterns in the data with more efficacy than the individual features themselves. Feature engineering is a very manual activity and, along with data preprocessing, it will generally constitute 80% of the time spent on a data science project[30].

In addition to analysis and organization of data, data science techniques are also important in the communication of information. As the scale of data increases, the way information is communicated becomes more important. Data visualization is the field involving the research and implementation of the most efficient ways to communicate large scale, complex data. Historically, the ability of researchers to present findings in engaging and consumable ways is outpaced by the rate of acquisition of new findings, especially in large data-mined spaces[31]. As this field improves, the barrier to entry lowers, and information can be more effectively communicated.

Machine learning is an exciting subfield of data science which allows computers to learn patterns in data without being explicitly programmed[32]. There exist a multitude of algorithm types, all of which work on fundamentally different paradigms and which excel in different problem types. ML algorithms have been used to build functional relationships between input and output variables for engineering processes in the past[21,33,34].

Although there are countless different algorithms associated with machine learning, this paper is primarily concerned with decision trees (DTs), random forests (RFs), and gradient-boosted machines (GBMs). Decision trees create sets of rules which can be applied to new data of a similar format. RFs are ensembles of DTs which are randomized such that each has the possibility of yielding a different response for a given input. The outputs of the DTs are combined using a weighted sum, resulting in RFs outperforming DTs in a majority of cases, at the cost of larger size on disk and longer execution time. GBMs are ensembles of many small DTs where the maximum depth is heavily constrained such that the model generalizes more aggressively. Their implementation is based upon the theory that a combination of a large number of weak predictors enables the creation of a single, strong predictor. The

boosted term refers to the practice of focusing on samples from the training set which are poorly predicted by the previous model structure, and this is an iterative practice that leads to significantly longer training times than RFs and DTs.

Tree-based models were selected for this work due to their favorability in large parameter spaces and relatively low CPU time[35,36], their comparable results to ANNs[36,37], and their ability to act as a sensitivity analysis using feature importance, which improves the ability to construct new features in a process called feature engineering. In this context, a feature is any single column of input variables a list of values which are of constant type and location in the input vector. In general, RFs and DTs are extremely practical for prototyping due to their flexibility, rapid training times, and rapid execution times. To the best of our knowledge, studies reporting these techniques used for building functional relationships among inputs and outputs for battery models are very rare, though some studies using several versions of static and dynamic ANNs have surfaced in the recent past[38,39].

The performance of lithium-ion batteries is heavily dependent upon the operating conditions present during their use in addition to the states of several internal variables. This sensitivity has driven the development of a wide range of models to simulate battery behaviors, from computationally expensive molecular dynamics simulations down to simple empirical models, which trade predictive fidelity for decreased computation time[3,4]. Between these extremes lie a variety of continuum-scale models, which include the single particle model (SPM)[1,5] and pseudo two-dimensional (P2D) model[4,6,40], the latter of which represents a good tradeoff between physical fidelity, predictive validity across chemistries, and execution time[7].

Empirical models are generally simple functions which have been fit to experimental data and are used to predict future battery performance, and often perform very poorly outside of their narrow window of accuracy. Polynomial, trigonometric, logarithmic, and exponential fits to experimental data are examples of empirical models which may offer some form of local accuracy. Equivalent circuit (EC) models are a class of empirical models which combine a series of linear circuit elements in order to approximate the behavior of a battery. Typically

used in state of charge (SOC) estimation, ECs are the one of the most widely implemented battery models due to their simplicity and speed of calculation[41]. SOC, defined as the fraction of total capacity remaining in the battery and represented as a percentage ranging from 0% to 100%, cannot be directly measured from a battery and must be inferred or modeled.

$$SOC(t) = \frac{Q(t)}{Q_n} \tag{2.1}$$

When higher fidelity or higher charge rates are needed, the P2D model is typically used. Based on the principles of electrochemistry, transport phenomena, and thermodynamics, the P2D model is represented by coupled nonlinear partial differential equations (PDEs) which vary with respect to electrode thickness x, particle radius r, and time t. The predictive capabilities of the model are improved by the inclusion of internal variables, including electrolyte concentration, electrolyte potential, solid-state potential, and solid-state concentration within the porous electrodes, as well as the concentration and potential of the electrolyte within the separator. This higher fidelity model typically solves on the order of minutes. In an effort to make these models easier to implement, faster to solve, and to allow for greater flexibility of application, surrogate models will be created using ML algorithms.

In this work, RFs, DTs, and GBM based surrogate models are created and their abilities to predict the dynamic behavior of the physicsbased model are examined. The most comprehensive P2D model has been utilized to create the data set for this study and the results are analyzed for accuracy and execution time. Trade-offs among training time, execution time, and accuracy for different ML algorithms are reported. Although surrogate models based on the P2D model are demonstrated in this paper, the concept is applicable for detailed 2D and 3D models for batteries, including multiscale thermal models.

The rest of the paper is organized in the following fashion the second section discusses the formulation of P2D model, the associated parameters and values of those parameters, and how the complete parameter set can be reduced to a set of most important parameters. The third section presents the basics of the machine learning algorithms used in this work,

including an overview of the techniques and details about the structures they create. Results are discussed in the fourth section, followed by a perspective of where machine learning algorithms can fit into the context of numerical modeling and surrogate modeling in the fifth section, and a summary of the findings can be found in the conclusions in the sixth section.

2.2 Choice of Model

The model used to generate this data set is a Newman-type P2D porous electrode model, as described in other works[4,6,40]. For convenience, a summary of equations and parameters will be recounted in Table 2.1, Table 2.2, and Table 2.3 here.

The P2D model is favored in literature[4,42] due to its balance between accuracy and relatively low execution time, typically 80 seconds when solved using finite difference techniques with 50,20,50 node points in cathode, separator, and anode, respectively. The model describes the intercalation and deintercalation of lithium particles from spherical anode and cathode particles in combination with ionic and electronic conductivities of the anode and cathode materials and electrolyte. The model includes effects from porosity of the anode, cathode, and separator, and also includes effects related to the diffusivity of ions through the electrolyte solution. Rates of reaction are modeled using Butler-Volmer relationships and the potentials are dictated by open circuit potentials, which are electrochemical properties of the materials that constitute the anode and cathode.

The parameter count for this model is 46, including the 20 values used for a piecewise-continuous linear fit for the open circuit potential, Up. In an effort to reduce the parameter space, the shape of this curve was approximated prior to data generation by iteratively varying the values and selecting the one with the lowest absolute error. All of the values in the linear piecewise function were then scaled from 0.95 to 1.05 to allow for variance in the data set. This reduced the dimensionality to 27, while still retaining the flexibility associated with variance in the open circuit potential. Practically, the dimensionality of the model can be further reduced using sensitivities from the ML models, which will be discussed further in the fourth section.

2.3 Surrogate Models from P2D Models

In this section, the processes of building surrogate models using ML techniques and the intricacies associated with them are described in detail. A flowchart representing the methodology is shown in Figure . When attempting optimization and real-time control using a surrogate for a computationally expensive model, the first step is to create a data set for training, testing, and validating the surrogate model. Here, the size of the data set is 24,000 individual trials with variance across 27 parameters as dictated by a linearly-scaled Sobol sampling arrangement, implemented using the Julia package Sobol.jl.[43,44] The ranges for each parameter are shown in Table 2.3 and were chosen based on a combination of the sensitivity of the model to these parameters and value ranges taken from reasonable percentage deviations from values from literature for nickel manganese cobalt oxide (NMC) type batteries,16 and others were estimated using conventional optimization techniques. The total time of data generation was 533 CPU-hours spread across 12 threads in an i7-6800k running at 3.8 GHz.

Practical considerations when working with RFs include limiting the number of features used, limiting the size of the data set, and limiting the number of DTs in the forest. Training a RF is extremely parallelizable, but CPU time and RAM requirements scale linearly with the size of the forest and the number of output values, and can quickly out scale typical hardware capabilities, occasionally requiring over 30 Gigabytes of RAM for a single model.

Three types of models are created, which will be henceforth referred to as constant time forward, constant time inverse, and recurrent. The forward surrogate model takes in a vector of parameters and outputs a voltage discharge curve, acting as a faster version of the physics-based model. The inverse model takes a voltage discharge curve as an input and estimates a set of parameters that were used to create that discharge curve, allowing for O(1) parameter estimation. The recurrent model takes a vector of parameters and voltages from previous times as an input and estimates the next voltage, or number of voltages. The extremely flexible nature of problem formulation allows for a variety of applications from the same data set.

The three types of models used are based heavily on the classification and regression trees (CART) algorithm, which greedily creates binary splits in order to grow trees in a top-down fashion, meaning that it begins at the inputs and ends with the outputs.[45] This process continues until the termination condition is reached, typically a maximum depth or perfect accuracy. The accuracy and training time of the models are functions of the size of the training set, the parameter space sampled in the data set, and several hyper-parameters of the models.

2.3.1 Surrogate Model Formulation

The process begins with a loss metric for tree f , typically the mean squared error (MSE) between the predictions of an existing tree structure and the data, which is evaluated at each terminal node T, where Lj represents the total loss at node j.[45,46] In this context, each interior node in the tree represents a set of rules used to split the data, which is optimized to increase the predictability of the tree. For example, for some portion of the data, if variance in feature 25 can separate the discharge curves with high purity, a threshold will be chosen above which the data are sorted right and below which the data are sorted left. The rules are generally not as simple as a single feature value, and typically default to either considering every feature or a random number up to log (nfeatures), depending upon the implementation. The total number of trees, or weak learners, is represented by n, and the potential attributes which can be used to split are represented by Ij. The loss is effectively the sum of the error accumulated across each tree, n, at each terminal node, T, after considering the splits associated with each feature from the data set, whose indexes are represented by Ij.

$$L(f) = \sum_{i=1}^{n} \sum_{j=1}^{T} \sum_{i\epsilon 1} L(y_i, w_j) \equiv \sum_{j=1}^{T} L_j \tag{2.2}$$

A proposed split at node k, such that k = / = j, is done by maximizing gain, or the difference in loss before and after the split. In this way, new splits are created without redundancy, where the gain is defined as the difference between the previous loss associated

Table 2.1: Transformed Governing Equations

Governing Equation	Boundary Conditions
Positive Electrode	
$\varepsilon_p \frac{\partial c_p}{\partial t} = \frac{1}{l_p}\frac{\partial}{\partial X}[\frac{D_{eff,p}}{l_p}\frac{\partial c_p}{\partial X}] + a_p(1-t_+)j_p$	$\frac{\partial c_p}{\partial X} \mid_{X=0} = 0$
	$\frac{-D_{eff,p}}{l_p}\frac{\partial c_p}{\partial X} \mid_{X=1} = \frac{-D_{eff,s}}{l_s}\frac{\partial c_s}{\partial X} \mid_{X=0}$
$\frac{-\sigma_{eff,p}}{l_p}(\frac{\partial \Phi_{1,p}}{\partial X}) - \frac{\kappa_{eff,p}}{l_p}(\frac{\partial \Phi_{2,p}}{\partial X}) + \frac{2\kappa_{eff,p}RT}{F}\frac{(1-t_+)}{l_p}(\frac{\partial lnc_p}{\partial X}) = I$	$\frac{\partial \Phi_{2,p}}{\partial X} \mid_{X=0} = 0$
	$\frac{-\kappa_{eff,p}}{l_p}(\frac{\partial \Phi_{2,p}}{\partial X}) \mid_{X=1} = \frac{-\kappa_{eff,s}}{l_s}(\frac{\partial \Phi_{2,s}}{\partial X}) \mid_{X=0}$
$\frac{1}{l_p}\frac{\partial}{\partial X}[\frac{\sigma_{eff,p}}{l_p}\frac{\partial}{\partial X}\Phi_{1,p}] = a_p F j_p$	$(\frac{1}{l_p}\frac{\partial \Phi_{1,p}}{\partial X}) \mid_{X=0} = -\frac{I}{\sigma_{eff,p}}$
	$\frac{\partial \Phi_{1,p}}{\partial X} \mid_{X=1} = 0$
$\frac{\partial c_p^s}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}[r^2 D_p^s \frac{\partial c_p^s}{\partial r}]$	$\frac{\partial c_p^s}{\partial r} \mid_{r=0} = 0$
	$-D_p^s \frac{\partial c_p^s}{\partial r} \mid_{r=R_s} = j_p$
$\rho_p C_{p,p}\frac{dT_p}{dt} = \frac{1}{l_p}\frac{\partial}{\partial X}[\frac{\lambda_p}{l_p}\frac{\partial T_p}{\partial X}] + Q_{rxn,p} + Q_{rev,p} + Q_{ohm,p}$	$-\kappa_{eff,p}\frac{\partial T_p}{\partial X} \mid_{X=0} = h_{env}(T_p \mid_{X=0} - T_{air}) - \frac{\lambda_p}{l_p}\frac{\partial T_p}{\partial X} \mid_{X=1} = -\frac{\lambda_s}{l_s}\frac{\partial T_s}{\partial X} \mid_{X=0}$
Separator	
$\varepsilon_s \frac{\partial c_s}{\partial t} = \frac{1}{l_s}\frac{\partial}{\partial X}[\frac{D_{eff,s}}{l_s}\frac{\partial c_s}{\partial X}]$	$c_p \mid_{X=1} = c_s \mid_{X=0}$
	$c_s \mid_{X=1} = c_n \mid_{X=0}$
$\frac{\kappa_{eff,s}}{l_s}(\frac{\partial \Phi_{2,s}}{\partial X}) + \frac{2\kappa_{eff,s}RT}{F}\frac{(1-t_+)}{l_s}(\frac{\partial lnc_s}{\partial X}) = I$	$\Phi_{2,p} \mid_{X=1} = \Phi_{2,s} \mid_{X=0}$
	$\Phi_{2,s} \mid_{X=1} = \Phi_{2,n} \mid_{X=0}$
$\rho_s C_{p,s}\frac{dT_s}{dt} = \frac{1}{l_s}\frac{\partial}{\partial X}[\frac{\lambda_s}{l_s}\frac{\partial T_s}{\partial X}] + Q_{ohm,s}$	$T_p \mid_{X=1} = T_s \mid X = 0$
	$T_s \mid_{X=1} = T_n \mid X = 0$
Negative Electrode	
$\varepsilon_n \frac{\partial c_n}{\partial t} = \frac{1}{l_n}\frac{\partial}{\partial X}[\frac{D_{eff,n}}{l_n}\frac{\partial c_n}{\partial X}] + a_n(1-t_+)j_n$	$\frac{\partial c_n}{\partial X} \mid_{X=1} = 0$
	$\frac{-D_{eff,s}}{l_s}\frac{\partial c_s}{\partial X} \mid_{X=1} = \frac{-D_{eff,n}}{l_n}\frac{\partial c_n}{\partial X} \mid_{X=0}$
$\frac{-\sigma_{eff,n}}{l_n}(\frac{\partial \Phi_{1,n}}{\partial X}) - \frac{\kappa_{eff,n}}{l_n}(\frac{\partial \Phi_{2,n}}{\partial X}) + \frac{2\kappa_{eff,n}RT}{F}\frac{(1-t_+)}{l_n}(\frac{\partial lnc_n}{\partial X}) = I$	$\frac{\partial \Phi_{2,n}}{\partial X} \mid_{X=0} = 0$
	$\frac{-\kappa_{eff,s}}{l_s}(\frac{\partial \Phi_{2,s}}{\partial X}) \mid_{X=1} = \frac{-\kappa_{eff,n}}{l_n}(\frac{\partial \Phi_{2,n}}{\partial X}) \mid_{X=0}$
$\frac{1}{l_n}\frac{\partial}{\partial X}[\frac{\sigma_{eff,n}}{l_n}\frac{\partial}{\partial X}\Phi_{1,n}] = a_n F j_n$	$\frac{\partial \Phi_{1,n}}{\partial X} \mid_{X=0} = 0$
	$(\frac{1}{l_n}\frac{\partial \Phi_{1,n}}{\partial X}) \mid_{X=1} = -\frac{I}{\sigma_{eff,n}}$
$\frac{\partial c_n^s}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}[r^2 D_n^s \frac{\partial c_n^s}{\partial r}]$	$\frac{\partial c_n^s}{\partial r} \mid_{r=0} = 0$
	$-D_n^s \frac{\partial c_n^s}{\partial r} \mid_{r=R_s} = j_n$
$\rho_n C_{p,n}\frac{dT_n}{dt} = \frac{1}{l_n}\frac{\partial}{\partial X}[\frac{\lambda_n}{l_n}\frac{\partial T_n}{\partial X}] + Q_{rxn,n} + Q_{rev,n} + Q_{ohm,n}$	$-\frac{\lambda_s}{l_s}\frac{\partial T_s}{\partial X} \mid_{X=1} = -\frac{\lambda_n}{l_n}\frac{\partial T_n}{\partial X} \mid_{X=0} -\kappa_{eff,n}\frac{\partial T_n}{\partial X} \mid_{X=1} = h_{env}(T_{air} - T_n \mid_{X=0})$

Table 2.2: Additional Equations

$Q_{rxn,i} = Fa_i j_i(\Phi_{1,i} - \Phi_{2,i} - U_i), i = p, n$

$Q_{rev,i} = Fa_i j_i T_i \frac{\partial U_i}{\partial T}, i = p, n$

$Q_{ohm,i} = \sigma_{eff,i}(\frac{1}{l_i}\frac{\partial \Phi_{1,i}}{\partial X})^2 + \kappa_{eff,i}(\frac{1}{l_i}\frac{\partial \Phi_{2,i}}{\partial X})^2 + \frac{2\kappa_{eff,i}RT_i}{F}(1 - t_+^0)\frac{1}{l_i^2}\frac{1}{c_i}\frac{\partial c_i}{\partial X}\frac{\partial \Phi_{2,i}}{\partial X}, i = p, n$

$Q_{ohm,s} = \kappa_{eff,s}(\frac{1}{l_i}\frac{\partial \Phi_{2,s}}{\partial X})^2 + \frac{2\kappa_{eff,s}RT_s]}{F}(1 - t_+^0)\frac{1}{l_s^2}\frac{1}{c_s}\frac{\partial c_s}{\partial X}\frac{\partial \Phi_{2,s}}{\partial X}$

$U_i(T_i, \theta_i) = U_{ref,i}(T_{ref}, \theta_{ref}) + (T_i - T_{ref})[\frac{dU_i}{dT}] \mid_{T_{ref}}, i = p, n$

$D_{eff,i}^s = D_i^s \, exp(-\frac{E_a^{D_i^s}}{R}[\frac{1}{T_i} - \frac{1}{T_{ref}}]), i = p, n$

$k_{eff,i} = k_i \, exp(-\frac{E_a^{k_i}}{R}[\frac{1}{T_i} - \frac{1}{T_{ref}}]), i = p, n$

$U_n = 0.7222 + 0.1387\theta_n + 0.029\theta_n^{0.5} - \frac{0.0172}{\theta_n} + \frac{0.0019}{\theta_n^{1.5}} + 0.2808exp(0.90 - 15\theta_n) - 0.7984exp(0.4465\theta_n - 0.4108)$

$\kappa_{eff,i} = \epsilon_i^{brugg}\left(4.1253 \times 10^{-2} + 5.007 \times 10^{-4}c - 4.7212 \times 10^{-7}c^2 + 1.5094 \times 10^{-10}c^3 - 1.6018 \times 10^{-14}c^4\right), i = p, s, n$

$D_{eff,i} = D\epsilon_i^{brugg}, i = p, s, n$

$\sigma_{eff,i} = \sigma_i(1 - \epsilon_i - \epsilon_{f,i}), i = p, s, n$

$a_i = \frac{3}{R_i}(1 - \epsilon_i - \epsilon_{f,i}), i = p, n; \theta_p = \frac{c^s|_{r=Rp}}{c_{max,p}^s}; \theta_n = \frac{c^s|_{r=Rn}}{c_{max,n}^s}$

with node k, Lk, and the losses associated with the new terminal nodes which are the left and right splits, LkL and LkR, respectively. This process is demonstrated in Figure 2.2.

$$Gain = L(f_{before}) - L(f_{after}) = L_k - (L_{kL} + L_{kR}) \tag{2.3}$$

This process establishes the structure of the DT or RF but does not instantiate any of the nodes with weights. A separate process optimizes the weights at each node, wj, such that the total loss is minimized.

$$w_j = argmin_w \sum_{i \epsilon I_j} L(y_i, w) \tag{2.4}$$

This is a typical stopping criterion for a DT, which contains only one structure. This can be further extended and used as the termination criterion for a RF by repeatedly solving for w j for additional structures until nestimators is reached. The structures for these models are very large and will go on to perfectly memorize the training data set unless an artificial limit is imposed. For GBMs, this process is iteratively repeated using a boosting algorithm, which greedily fits additional small (e.g. max depth < 5) structures until the nestimators

Table 2.3: Parameter Value Ranges

Symbol	Parameter	Positive Electrode	Separator	Negative Electrode	Units
σ_i	Solid phase conductivity	7.00 - 14.3		70.0 - 143	S/m
$\epsilon_{f,i}$	Filler fraction	1.74e-2 - 3.56e-2		2.29e-2 - 4.67e-2	
ϵ	Porosity	0.380 - 0.410	0.310 - 0.640	0.260 - 0.520	
Brugg	Bruggeman Coefficient	1.05 - 2.14	1.05 - 2.14	1.05 - 2.14	
D	Electrolyte Diffusivity	2.40e-11 - 1.50e-7	2.40e-11 - 1.50e-7	2.40e-11 - 1.50e-7	m^2/s
D_i^s	Solid Phase Diffusivity	9.62e-16 - 7.37e-12		1.38e-15 - 7.2e-12	m^2/s
k_i	Reaction Rate Constant	1.36e-11 - 1.34e-9		1.03e-11 - 1.03e-9	$mol/(sm^2)/(mol/m^3)^{1+\alpha a,i}$
$c_{i,max}^s$	Maximum Solid Phase Concentration	4.40e4 - 4.77e4		2.90e4 - 3.22e4	mol/m^3
$c_{i,0}^s$	Initial Solid Phase Concentration	1.56e4 - 2.19e4	2.56e4 - 3.21e4	mol/m^3	
c_0	Initial Electrolyte Concentration		1.14e3 - 1.26e3		$mol/m3$
$R_{p,i}$	Particle Radius	3.98e-6 - 1.59e-5		4.99e-6 - 2.00e-5	m
l_i	Region Thickness	4.11e-5 - 4.37e-5	1.28e-5 - 2.00e-5	m	
t_+	Transference Number	0.314 - 0.387			
F	Faraday's Constant		96487		C/mol
R	Gas Constant		8.314		$J/(molK)$
T_{ref}	Temperature		298.15		K
ρ	Density	2500	1100	2500	kg/m^3
C_p	Specific Heat	700	700	700	$J/(kgK)$
Γ	Thermal Conductivity	2.1	0.16	1.7	J/(m K)
$E_a^{D_i^s}$	Activation Energy for Temperature Dependent Solid Phase Diffusion	5000		5000	J/mol
$E_a^{k_i}$	Activation Energy for Temperature Dependent Reaction Constant	5000		5000	J/mol

limit is achieved. This iterative nature leads to considerably longer training times than either DTs or RFs. However, the ensemble of multiple weak and varied predictors allows for better generalizability than either DTs or RFs.[45] Formally, GBMs are a solution to the below problem statement, where θ_m is the weights for the structure ϕ_m is a list of which minimizes the loss function.

$$\{\theta_m, \phi_m\} = argmin_{\{\theta_m, \phi_m\}} \sum_{i=1}^{n} L(y_i, f^{m-1}(x_i) + \theta_m \phi_m(x_i)) \tag{2.5}$$

The boosting process requires a gradient calculation of the loss in function space $g_m(x)$, which is calculated by fitting a regression tree model using MSE as its loss function.

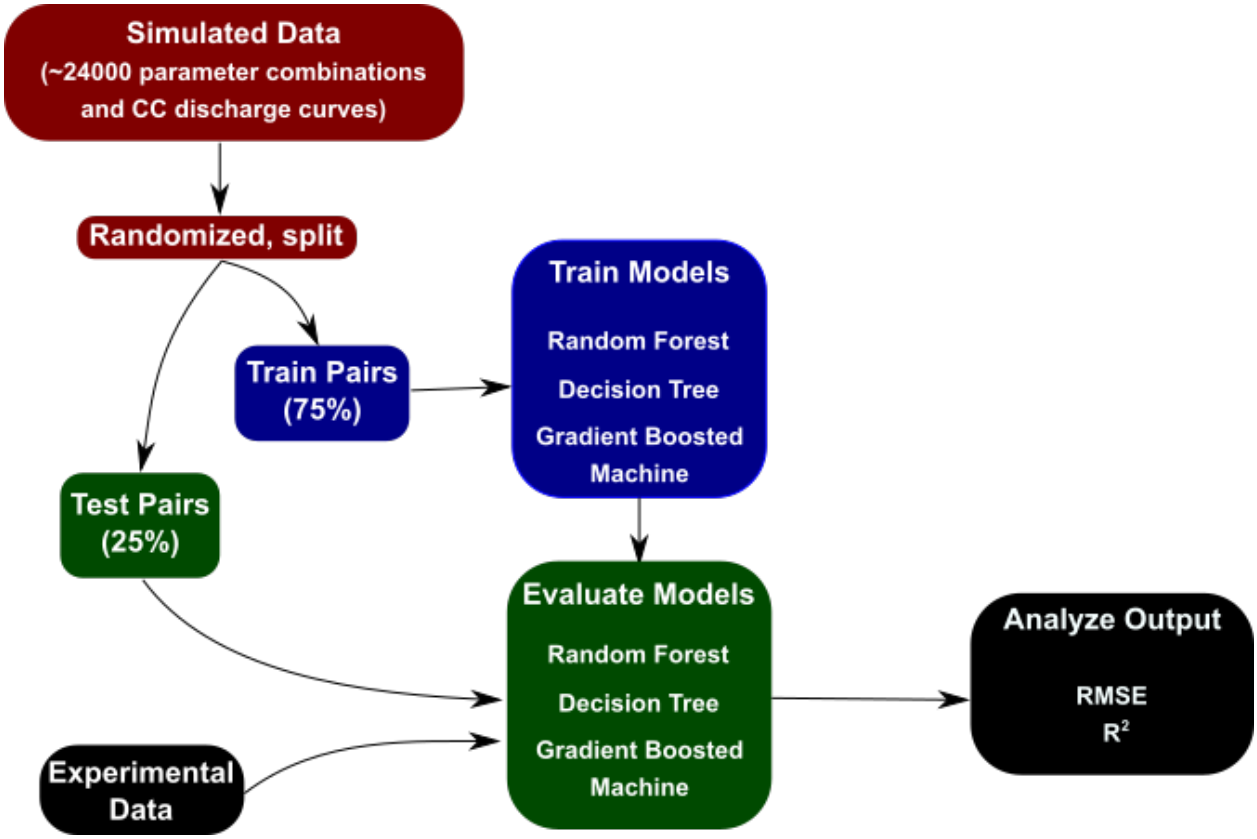


Figure 2.1: A process flowchart for the creation of surrogate models from simulated data.

$$-g_m(x) = -\left[\frac{\partial L(y, f(x))}{\partial f(x)}\right])_{f(x)=f^{m-1}(x)))} \tag{2.6}$$

$$\phi_m = argmin_\phi \sum_{i=1}^{n} [(-g_m(x_i)) - \phi(x_i)]^2 \tag{2.7}$$

In addition to the direction of the step, a step length ρ_m must also be determined. This is typically done using a line search algorithm which inherits a learning rate from the initial GBM call.

$$\rho_m = argmin_\rho \sum_{i=1}^{n} L(y_i, f^{m-1}(x_i) + \rho\phi(x_i)) \tag{2.8}$$

$$f_m(x) = \eta \rho_m \phi_m(x) \tag{2.9}$$

Finally, these concepts are implemented in the python package Sci-Kit Learn[46]

2.3.2 Building the Models - Constant Time Models

Constructing a forward model is a multi-input, multi-output problem, which disqualified GBMs from this section, as their current implementation is limited to a single output. While an ensemble of GBMs on the basis of single output model could have been created, the training time was prohibitive. When implementing a machine learning model, it is important to format the data such that the containers are of constant size. In this case, the discharge voltage was sampled at 100 points using linear interpolation, with the time value at the end of the vector equal to 1800 seconds such that samples which overshoot the final experimental time of 1600 seconds can still be fully captured. In the case of the forward models, the inputs are the physical parameters, as shown in the second section, and the output is a time-scaled voltage discharge curve.

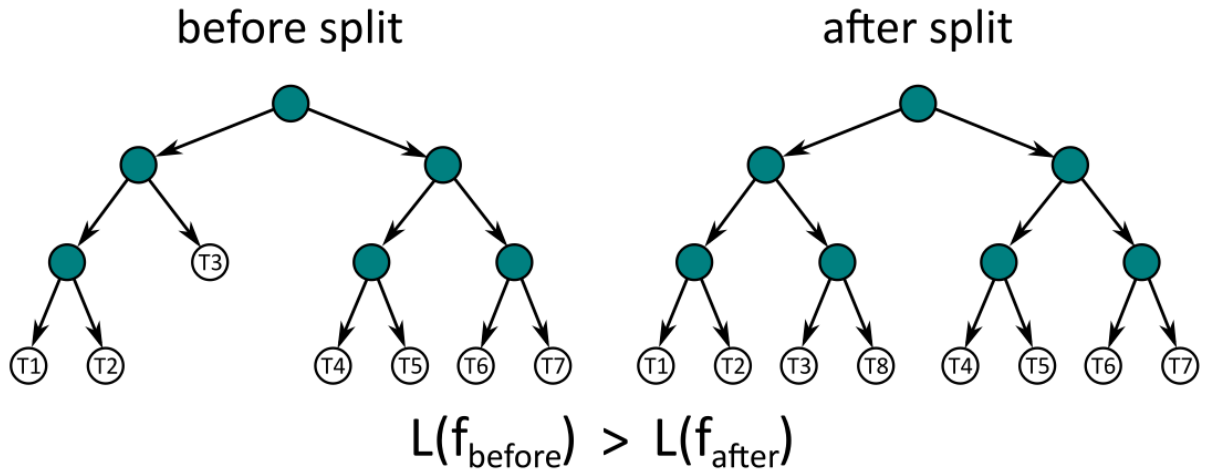


Figure 2.2: Visual representation of addition of splits during tree structure building. A split is created at T3 and reduces the total loss L, so it is kept.

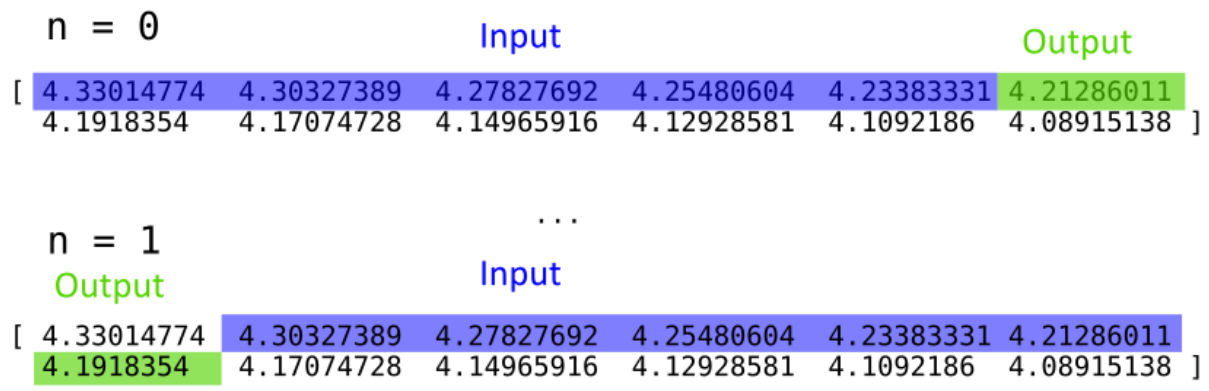


Figure 2.3: Representation of windowed clustering for creation of recurrent data set. For each time step, n voltages are taken as the input and the next voltage is taken as the output. In this case, n = 5.

Construction of the inverse models was similar to that of the forward models, however, the inputs and outputs were switched. In this way, it was possible to use the inverse models for O(1) parameter estimation for the surrogate or physics-based models. Due to the reduced parameter space, it was also feasible to create per-parameter GBMs.

2.3.3 Building the Models - Recurrent Models

The extreme flexibility of machine learning models allows for more creative uses of a typical data set than matching or swapping input and output. In order to create a recurrent model, the training and test sets need to be manipulated in such a way that time or previous voltages are used as inputs to predict some next range of voltages. This is done by first disassembling the vectors of voltages and then reassembling the data such that the input values contain a list of parameters and α voltages, and the output value is the next voltage. In Figure 2.3 below, and in the later models, α is equal to 5. This can be modified to predict SOC by simply generating a value for the SOC at each voltage point and substituting the SOC

value for the predicted voltage value. While GBMs are restricted to single value prediction, DTs and RFs can easily estimate anything related to the data set. In the equation below, new X and Y arrays are constructed using the old arrays which use a set of parameters and previous voltages to predict the next voltage.

$$X_{new}[i] = [X_{old}[i, n : n + \alpha], Y[i]], Y_{new}[i] = X_{old}[i, n + \alpha + 1] \tag{2.10}$$

2.4 Results and Discussion

2.4.1 Effect of Sample Size

When splitting the data into test and training sets, it is advantageous to randomize the trials to minimize the effects of sampling bias. In this data set, the individual trials were randomized in batches of 100, such that smaller subsamples of the data cannot contain runs from later batches. For example, when selecting only the first 1000 trials, the training and test sets are randomized, but do not contain samples from simulations 100024000, as they might be in a single batch randomization. In this way, it is possible to sequentially give more information to an RF or DT and examine the effects of a larger training set, as demonstrated in Figure 2.4.

2.4.2 Constant Time Models

The more traditional form of surrogate models are effectively constant time models, in that they take in a set of parameters and output the entire discharge curve at once, rather than solving for the next time step iteratively, as is the case with the recurrent models. The recurrent models can be used to iteratively rebuild the constant time model result for a given set of parameters, but doing so is a slower implementation of the same constant time model, as the function must be called for each discretization of time rather than once. The constant time surrogate models perfectly learn the training data set, and any new set of parameters, like a sample from the test set, yields a curve from the training set whose parameters are

similar in value. Depending upon the coarseness of the parameter sampling, this may or may not yield a good result.

DTs are known as constant output estimators, meaning that they allow for continuous inputs, but they do not interpolate between the data points, the values are connected in a way similar to a step function. In other words, although the input space is continuous, the output space can only have the same values as the discrete values in the training set. As such, an optimized result of a surrogate model using a DT will yield, at best, the closest result from the training set. While this is a good result, the act of using a DT contributes no tangible benefits over calling a look-up table of the training data, other than a significant speed increase. The closest result from the lookup table is used as a benchmark in other problem formulations, so these results are not shown here.

Since the constant time models exclusively use the parameters to estimate the output curves, their ability to predict the output voltages is inherently dependent upon their ability to identify meaningful parameters and correlate them to the changes they cause. The physicsbased model is more sensitive to some parameters than others, and this sensitivity carries over to the surrogate models. RFs are able to report this relative sensitivity in a metric called feature importance, the definition of which can vary by package. In this instance, it is defined as the total number of times that feature is used in a split divided by the total number of splits. A feature is a single input from the input vector, meaning that each parameter in these constant time models is a feature. In addition to the given features, it is also possible to create artificial features that are combinations of your inputs in a practice known as feature engineering.

Feature engineering is one of the most overlooked and most important facets of machine learning[30]. It entails findings relationships between features which can better represent the trends in the data than the individual features themselves. Other works[47] have already demonstrated that the types of engineered features that can be learned by various models can differ, and helps give insight into which types of features to look for. If RFs can learn the relationship between an input x and an output x^2, for example, there is no need to explicitly

Table 2.4: Engineered features, groups with importance > 0.0064 bolded

Number	Relationship	Positive Electrode	Separator	Negative Electrode	Units
1,2,3	$\frac{\epsilon_i^{Brugg} D}{l_i^2 \epsilon_i}$	X	X	X	1/s
4,**5**	$a_i(1 - 0.1 t_+ k_i)$	X	X	**X**	$m^2/m^3(mol/(sm^2)/(mol/m^3)^{1+\alpha a,i})$
6,7	$a_i(1 - 0.1 t_+ k_i)\sqrt{c_0}$	X		X	$m^2/m^3(mol/(sm^2)/(mol/m^3)^{1+\alpha a,i})(mol/m^3)^{0.5}$
8,9	$a_i(1 - 0.1 t_+ k_i)\sqrt{c_0}\sqrt{c_{i,0}^s}$	X		X	$m^2/m^3(mol/(sm^2)/(mol/m^3)^{1+\alpha a,i})(mol/m^3)$
10,11,12	$\frac{\epsilon_i^{Brugg} D}{l_i}$	X	X	X	m/s
13,14	$\frac{\sigma_i}{l_i}$	X		X	s/m^2
15,16,17	$\frac{\epsilon_i^{Brugg}}{l_i}$	**X**		X	1/m
18,19,20	$\frac{\epsilon_i^{Brugg}(1-0.1t_+)}{l_i}$	**X**		X	1/m
21,22	$\frac{\sigma_i}{l_i^2}$	X		X	s/m^3
23,24	$\frac{D_{eff,i}^2}{R_{p,i}^2}$	X		X	1/s
25,26	$\frac{k_i}{\sqrt{c_0}}$	X		X	$mol/(s\ m^2)/(mol/m^3)^{1+\alpha a,i}/(mol/m^3)^{0.5}$
27,29	$\frac{R_{p,i}^2}{D_{eff,i}^2}$	**X**		X	s
28,30	$\frac{R_{p,i}^2}{\epsilon_i^{Brugg} D}$	**X**		X	s
31,**32**	$\frac{R_{p,i}^2}{2k_i c_0^{0.5}}$	X		**X**	$m/(mol/m^3 mol/(sm^2)/(mol/m^3)^{1+\alpha a,i}$
33,**34**	$\epsilon_i c_{i,0}^s c_0$	X		**X**	$(mol/m^3)^2$
35,36	$\frac{\epsilon_i^{Brugg} l_i}{\epsilon_s^{Brugg} l_s}$	X		X	m/m
37	$\frac{\epsilon_p}{\epsilon_n}$				1
38	$\frac{l_p}{l_n}$				m/m
39,40	$\frac{1}{R_{p,i}}$	X		X	1/m

create the feature x^2 in the input set, as no new information will be added. This can help reduce the dimensionality of the feature engineering problem.

Since this data is simulated, the equations themselves can be a good source of inspiration for engineered features. Upon looking in the equations, some of the below relationships became apparent.

Physically building these features requires creating a new array containing the old features and having room for the new ones, and then adding in the values which represent the combinations of features to complete the array. This process can be vectorized trivially for performance improvements, as shown in Equation 11. Figure 2.5 demonstrates the feature importance of the entire spectrum of features, which includes 26 of the original 27 parameters, where the single parameter representing the scale of the OCP has been expanded to the actual

values, bringing the parameter total up to 46. Additional parameters are those represented in Table 2.4, in order of appearance. Only some of the features have a strong feature importance, demonstrating that these features have added patterns that accurately represent trends in the data. The algorithm score as a function of importance threshold is shown in Figure 2.6.

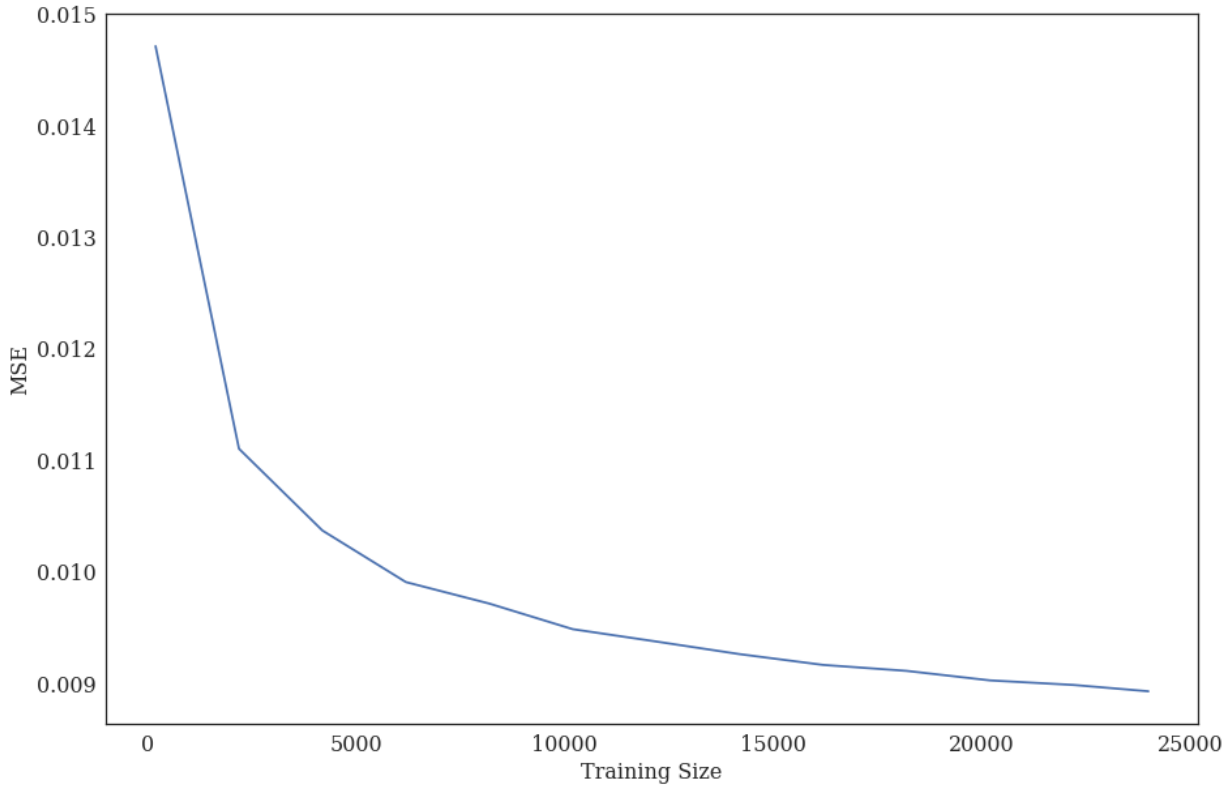


Figure 2.4: Effect of training size on test set error for a RF of size n=100.

The best possible result would come from exhaustively creating and trying every permutation of feature combinations, but that is an intractable amount of features and is generally not computationally feasible. Even with only 27 parameters, and for a single relationship, like division, this represents 227 parameter combinations. When higher-order interactions are considered, manual feature engineering becomes the only option. To demonstrate this, both processes have been done 40 manual features have been created using relationships found in the data and various dimensionless groups, and 250 features have been generated

by randomly selecting 2 features and randomly multiplying or dividing them. This has been done in 3 batches of sizes 150, 50, and 50, in order to allow for the later trials to create more complex interactions by potentially sampling from parameter groups.

$$F_{new}[:] = Y_{old}[:, 5] * Y_{old}[:, 6] / Y_{old}[:, 7] \tag{2.11}$$

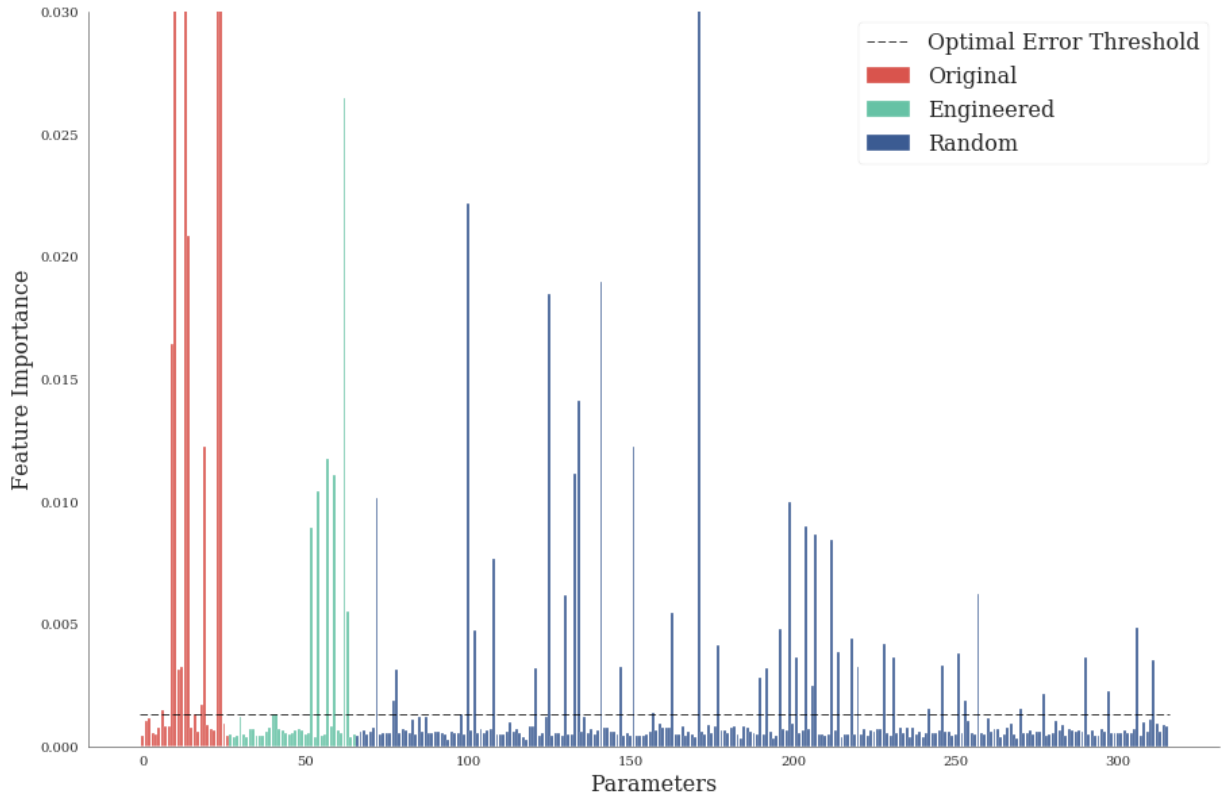


Figure 2.5: Per-parameter feature importance. Parameters 0-27 represent the original input parameters, 28-66 represent the engineered parameters, and 67-316 represent the randomly created parameters. Clipped at 0.030 for visibility.

Once engineered features have been added, it is important to remove any features which are not contributing to the accurate prediction of new data in a process called feature pruning.[48] These features, which have no true predictive value, can only make contributions to overfitting, and cannot be relied upon to accurately predict the test data.

By removing features whose importance is below a certain threshold, the accuracy of the validation set is improved. At the two pruning extremes, the model would use either all the features or one of the features, and response of the score to the pruning threshold should be a U-shaped curve, with a maximum at an intermediate value, as implied by the bias-variance tradeoff.[49] With increasing feature count, the variance is reduced, but if some features contain little predictive power, their effects may only be seen in the training set and not in the test set, leading to increased bias, also called over-training.

As demonstrated in Figure 2.6, there is an optimal range of threshold values where features which contribute exclusively to overfitting are discarded, but features which add valuable information are retained, increasing the accuracy on both seen and unseen data. The lines in Figure 2.5 and Figure 2.6 are drawn at the same importance threshold, which is determined by selecting the value that corresponds to the highest R^2 score, where the R^2 value represents the quality of the fit of the line $y = x$ on a plot of predicted values vs true values. In this instance, a value of 1.0 is perfect and a value of 0.0 is uncorrelated. Of the 40 hand-created features, 5 represented high scoring features, compared to 13 of the 250 randomly generated features and 7 of the initial 27 features. It is worth nothing that the feature importance is relative to the other features, and does not define an absolute relationship between any particular input with the outputs.

Initially, it seemed that feature importance was an extremely reliable method of feature pruning, and while it should certainly be considered, it is not enough for a new feature to simply have a high sensitivity it must also describe a relationship that is not currently described in the existing features. For instance, the most important feature is *socp*, or initial state of charge in the positive electrode, with a feature importance of 0.206. This feature is so important that its presence in another random group, like $\frac{1}{socp}$, can have a high selectivity without necessarily generating new useful information. In this instance, the addition of the randomized features actually decreases the training accuracy as these artificially high-scoring parameter combinations can edge out features which may be less selective but which may add meaningful relationships. By pruning the original features, it is possible to achieve a

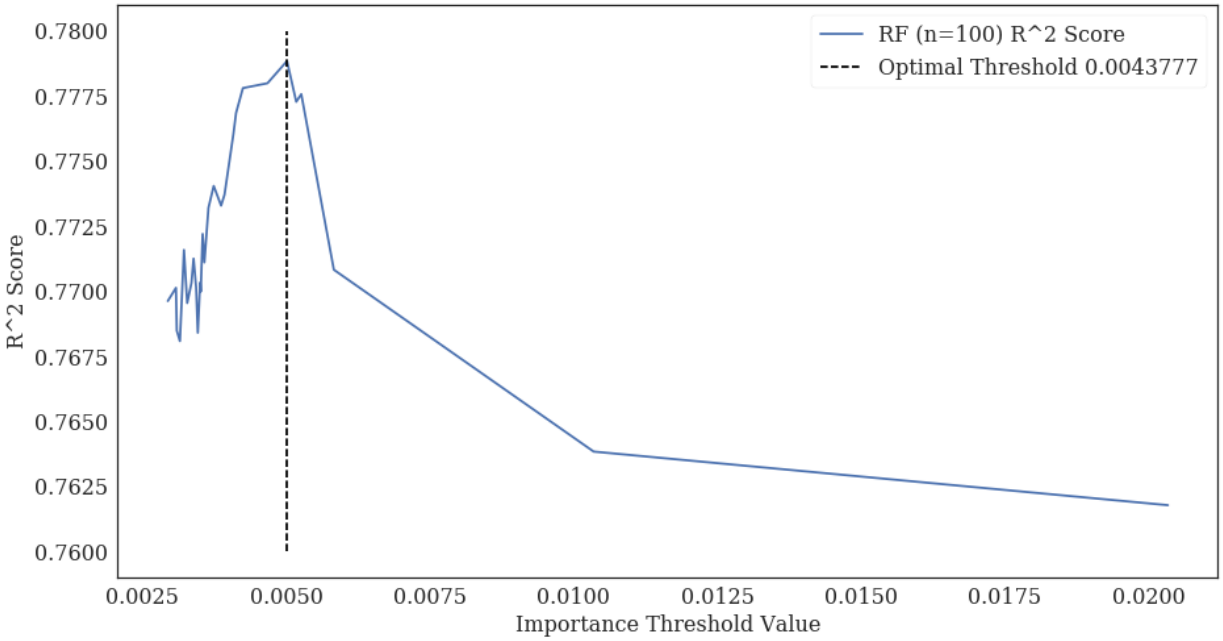


Figure 2.6: Random forest score with varying feature importance threshold, obtained from model trained on original and engineered features.

test set RMSE of 0.0953 volts. Adding in hand-engineered features can reduce this to a test set RMSE of 0.0832 volts, while the addition of randomly created features, even with the hand-engineered features present, raises this again to an test set RMSE 0.0861 volts. When only the original and randomized features are used, the test set RMSE peaks at 0.0878 volts, as shown in Figure 2.7. Despite this, it can be seen in Figure 2.5 that many of the randomly created features have a very high feature importance when compared to the hand-created features. As such, feature engineering is a useful tool, but having features with higher importance does not guarantee a better result on the test data set.

In addition to engineering new features, feature scaling is also an important attribute. Due to the loss function lacking a normalization term, the data must be scaled such that error can be adequately represented in the loss function, even for small numbers. As a concrete example from this data set, the parameter kn, representing the rate of reaction in the negative electrode, has a value of the order 10^{-14}. Due to its very small value, even

extremely large changes in the relative value of the prediction will be counted as accurate guesses by the loss function.

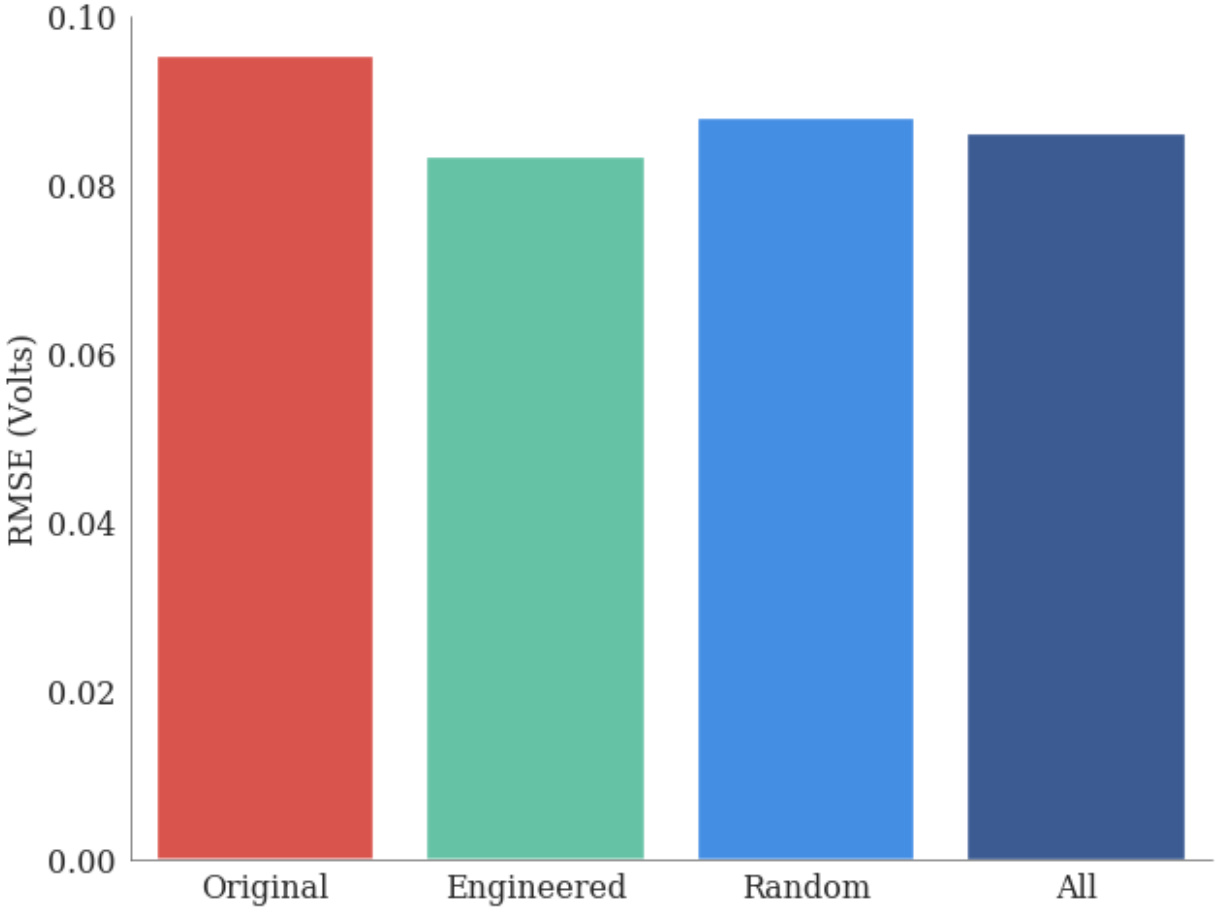


Figure 2.7: Test data set RMSE of identical models with original, original and engineered, original and random, or all features.

In addition to using the forward model to create a discharge curve from a set of model parameters, it is also possible to train a model in reverse, such that for a given discharge curve, the model outputs the parameters used to create that curve, which enables an O(1) function for parameter estimation. Since this model is also trained using a DT, it is subject to the same limitations as the forwards models above, namely that the best result achievable

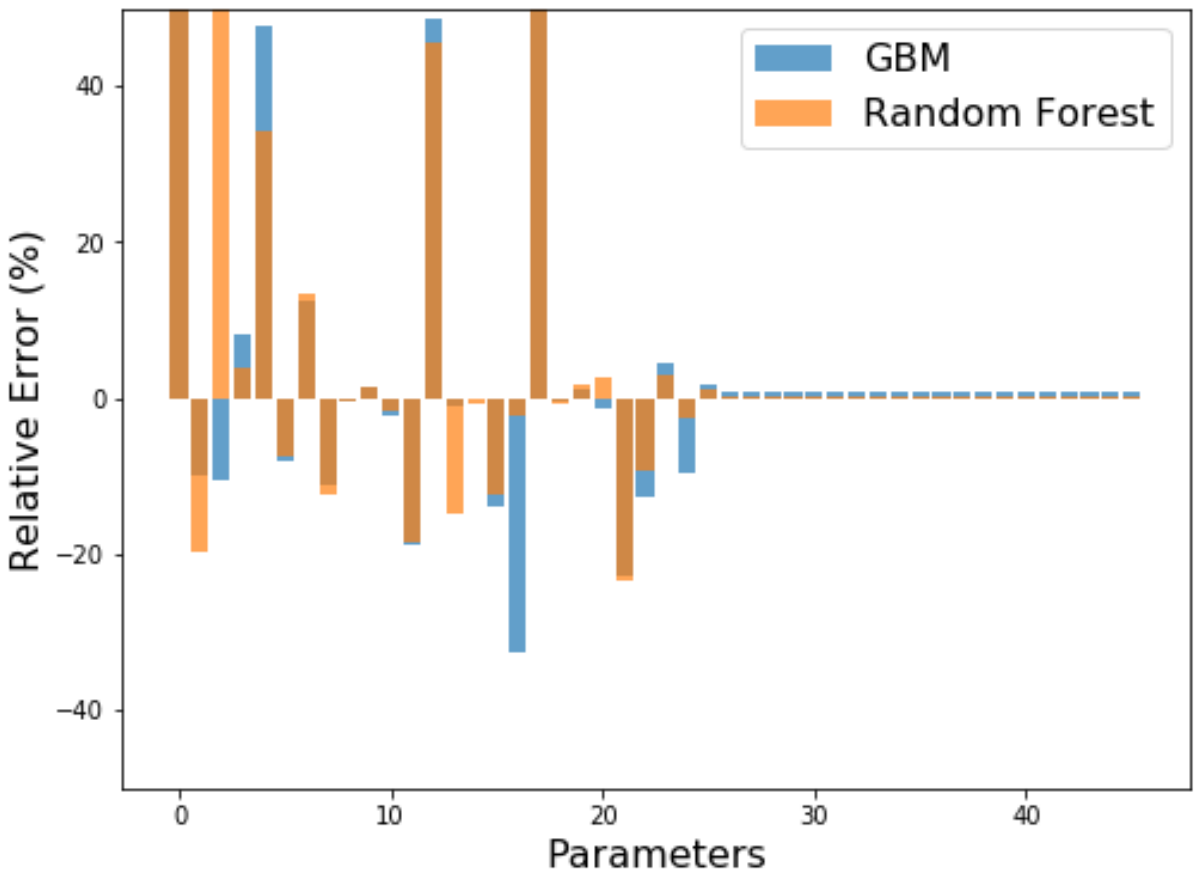


Figure 2.8: Relative error with respect to every parameter for inverse model predictions, cropped at ± 50% error for visibility.

is the closest fit from the training data set.

Once this inverse model is trained, it is simple to apply to experimental data, which requires scaling the X-axis to the same scaled time as the training data and ensuring that the length of the vectors is the same. Once this is done, it can be directly predicted using the algorithms. By taking the closest result from the training set as the correct values, it is possible to score the algorithms on their accuracy, the results of which are shown in Figure 2.8 and whose scores and execution times are documented in Table 2.5. The resulting discharge curves are shown in Figure 2.9. The voltage remaining at 2.5 V after discharge

Table 2.5: Root mean square error (RMSE) and execution times of inverse-model estimated discharge curves

Model	Error (V)	Execution Time (s)
Training Set Fit	0.02049	4.98
Gradient boosted Machine	0.1850	5e-6
Random Forest	0.1074	5e-4

Table 2.6: Root mean square error (RMSE) and execution times of inverse-model estimated discharge curves

Method	n+1	n+2	n+3	n+4	n+5
DT	0.0290	0.0333	0.0720	0.1068	0.1414
RF (nt = 100)	0.0314	0.0326	0.0609	0.0955	0.1345
GBM (nt = 500)	0.0319	0.0711	0.1058	0.1364	0.1640

is not physically meaningful, but padding the relevant discharge information with values is necessary to keep a constant vector length for the model, so the minimum voltage from the discharge curve was used.

While this method is easy to implement, the results are not phenomenal. It is possible that with more data, or with a machine learning technique that is capable of true interpolation, that this technique could be more successful. However, it is apparent that even the closest fit from the training set is not a great fit to the experimental data. This implies that in order to have the potential for a better fit using this technique, it would be necessary to either generate more data or to reduce the sampled parameter space, which is difficult to do without knowing the values for the optimal fit. However, the speed increase is significant, even compared to the lookup table.

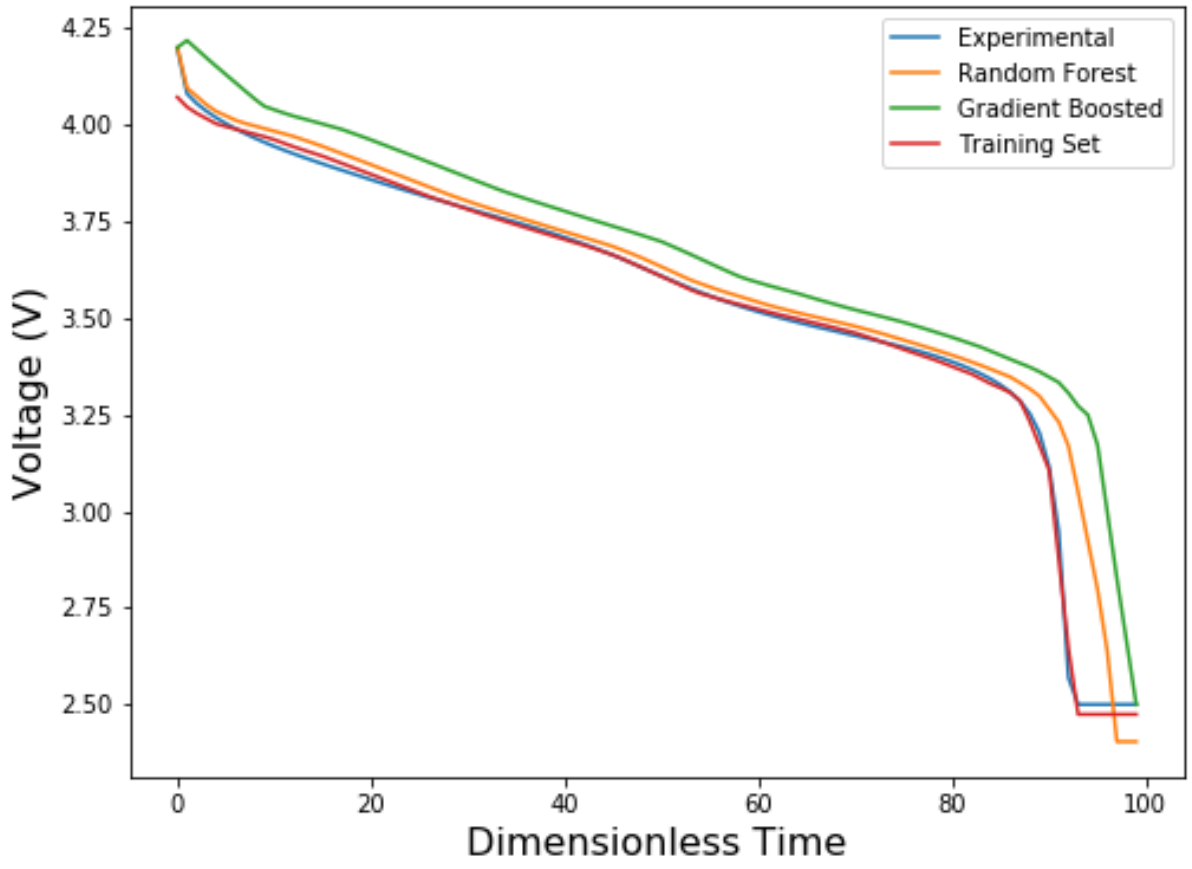


Figure 2.9: Simulated discharge curves associated with the estimated parameters.

2.4.3 Recurrent Voltage Prediction

One of the more interesting applications of the data set is the ability to create models for closed-loop, real-time control which are not possible using the set of equations in the physics-based model as-is. For example, it becomes possible to create models where the previous voltages are used to predict the next voltage, or the next series of voltages, and are updated at regular intervals. This allows for a fit which outperforms the training set in the upper portion of the discharge curve, but which overshoots the final voltage, as shown in Figure 2.10. The GBM RMSE is 0.0267 volts, and the RMSE of the closest training data

is 0.0158.

By changing the number of voltage points predicted, it is possible to create a moving window of prediction where the errors can be used to quantify the confidence of each prediction. Table V2.6 below represents the RMSE of the points as a function of distance from the voltage points used to predict. Interestingly, the GBM ensemble outperforms both the DT and RF at predicting the next voltage point of the experimental data, but does significantly worse predicting further voltages. This is demonstrated in Figure 2.11, where the downward choppiness of the prediction belies the tendency of the model to estimate an early termination of the discharge curve.

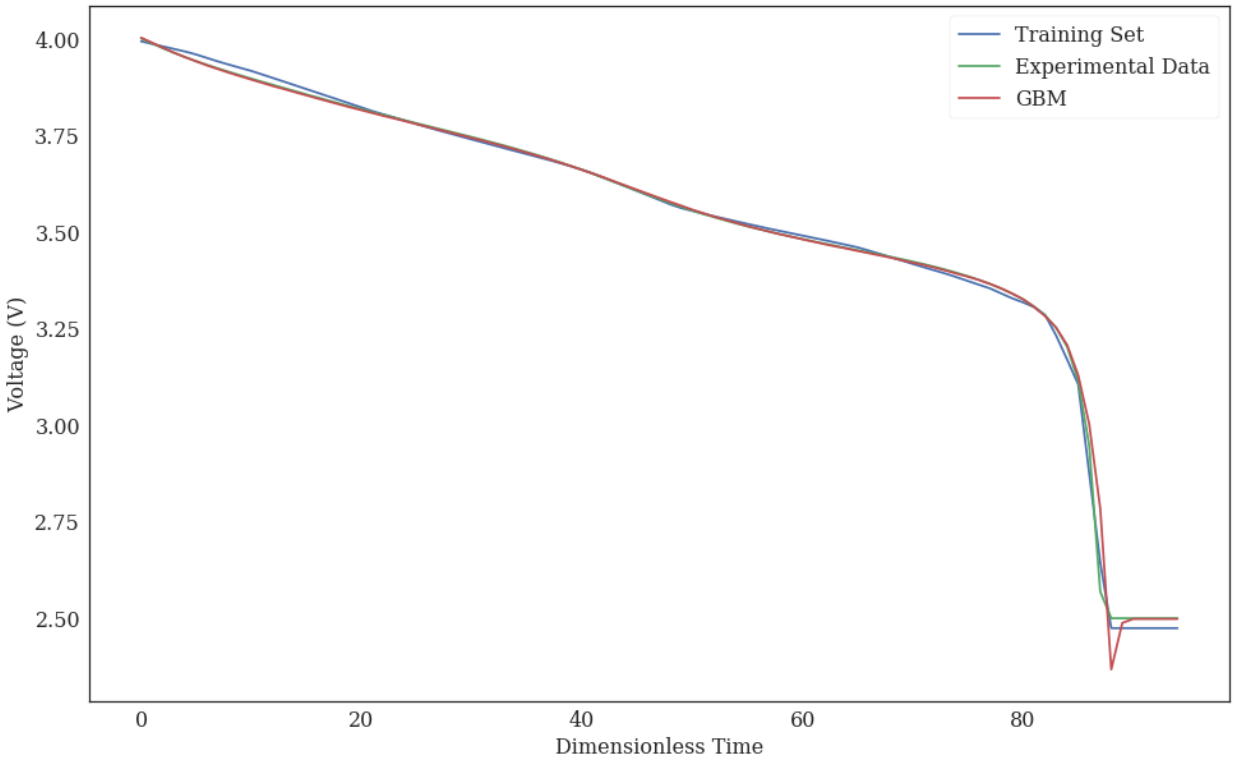


Figure 2.10: Gradient-boosted machine voltage estimation performance using 5 previous voltages to predict the next voltage, and the closest discharge curve from the training set.

Due to the requirements of DTs, RFs, and GBMs for consistent dimensions of inputs,

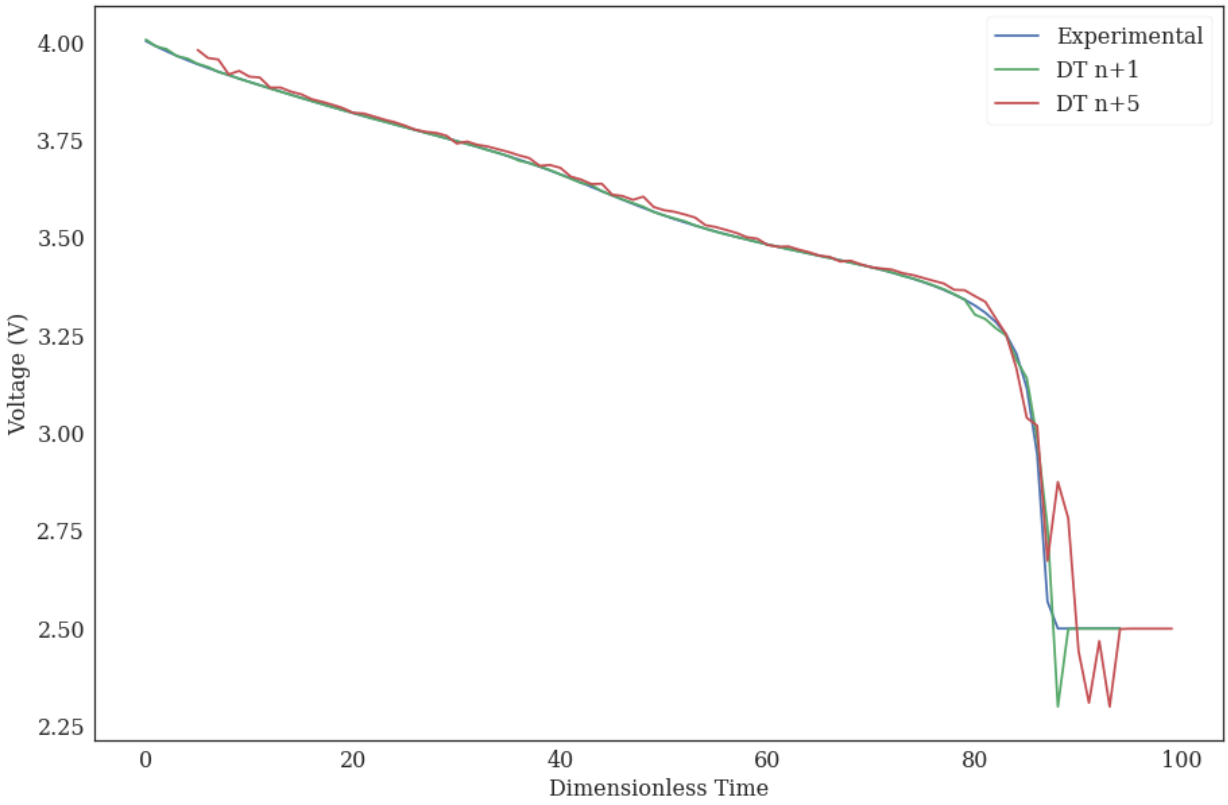


Figure 2.11: DT predicted voltage vs dimensionless time for n+1 and n+5. Using 5 input voltages, either the voltage at n+1 (green line) or the voltage at n+5 (red line) are predicted. This demonstrates the change in accuracy as a function of distance in time from n.

all information other than the direct model inputs are discarded. As such, the model has no record of previous discharge history beyond the previous 5 voltage values. There are two approaches to quantifying error using these methods: the error of the predicted points can be calculated sequentially using the experimental data as inputs, or the error can be calculated by iteratively calling the model using previously predicted values until the voltage has hit a certain value, and examining how that accuracy changes as a function of time. In other words, it is possible to sequentially predict only the next voltage and rebuild a single curve from this series using exclusively experimental data as input, as in Figure 2.11, and it is also possible to extend each individual prediction past a single point to an entire discharge curve,

and to examine this series of curves, as done in the next section.

2.4.4 Recurrent SOC Prediction

So far, these methods have been applied exclusively to voltage prediction, but they can easily be adapted to estimation of state of charge. In the data set, rather than having voltage as a function of time, it is possible to simply coulomb count and give each time point a SOC measure. There are many different ways to structure the predictive model. Based on the control scheme, it may be desirable to have a SOC estimate for the current time, for a future time, or for both. The flexibility of machine learning allows for any formulation using any input, all generated from the same data set. In this analysis, the current SOC was predicted using [n-4...n] voltages.

Using the previously mentioned voltage discharge curve data set, another output array was created that represents the simulated battery SOC vs time. The original data were created using a constant 2C discharge, so the SOC is linearly decreasing at all points between the maximum voltage and the minimum voltage, as shown in Figure 2.12. The recurrent methods were created such that the present voltage and previous 4 voltage points can be used to predict the present SOC. The variance of the data set is an advantage when using previous voltages to predict the next voltage, as it allows for the model to adapt to patterns that it has seen even if the entirety of the new curve is not in the training set. However, this high variance causes SOC estimation to be difficult due to the sensitivity of the calculation to the ending time of the simulated data.

To create the SOC data set, the simulated data that did not get below the typical SOC cutoff of 2.5 volts was omitted, which halved the size of the data set. The lowest voltage was taken as 0the highest voltage was taken as 100% SOC. Since the current was constant for these simulations, the SOC decreased linearly with time. This decreased the uniqueness of the data set, simply requiring that two simulated batteries hit their lowest voltage at the same time to be considered identical.

Figure 2.13a gives a bit of insight into what is happening when the DT fits a given curve

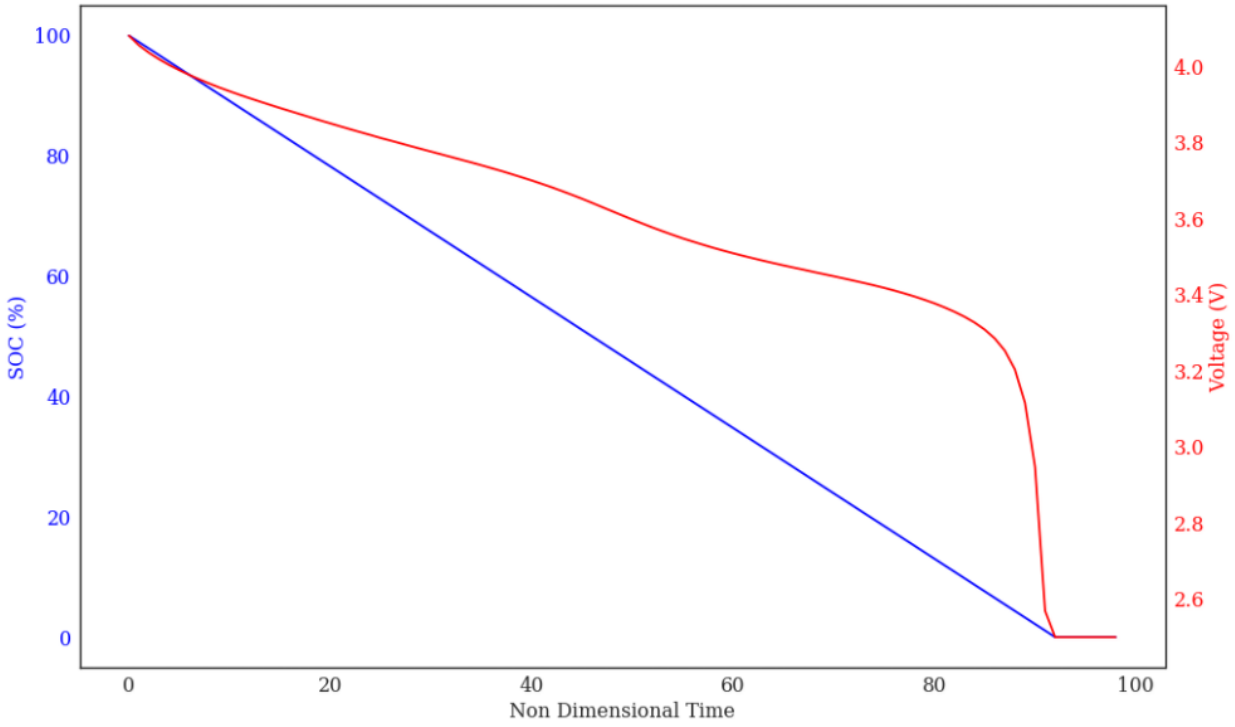


Figure 2.12: SOC and voltage relationship at constant current draw.

via a series of voltage predictions, the output of which is shown in Figure 2.13b. Figure 2.13a represents the combinations of voltage forecasts that the DT makes as it iteratively fits sections of voltage data. At the beginning of the discharge curve, there are fewer predictions, and they are all very similar. By the 5th estimate, the deviations of the data from the training set can be seen, and several curves develop. While the extended voltage predictions are fairly tightly grouped during the bulk of the discharge, we can see there is a large amount of variance toward the end, in particular when the curve achieves minimum voltage. The relatively large parameter variance covers a number of batteries, and the target curve is similar to only a handful of them. While the large error in the extended voltage predictions does not hurt the short-range voltage predictions, it is very detrimental to the SOC predictions, which are heavily dependent upon the ending time of the curve. Figure 2.13b represents the accompanying prediction of subsequent voltages, sampled only at the next point.

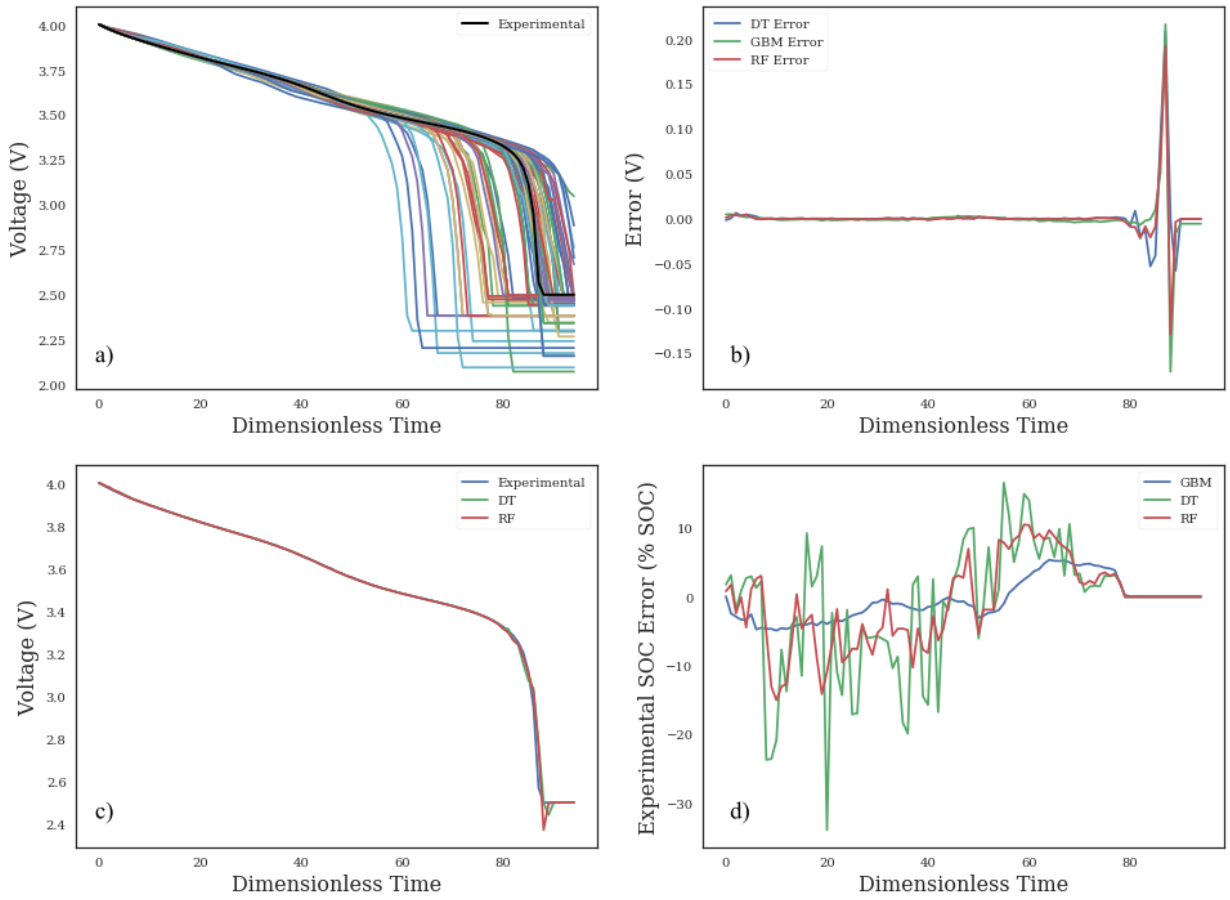


Figure 2.13: a) Extended sequential voltage predictions for the target data, demonstrating the long-term variance of the predictions which are accurate in the short-term. b) Error between the experimental and predicted curves for recurrent networks. c) Discharge graphs compared in b. d) Recurrent SOC error for experimental data.

The errors in the predictions, in absolute volts, are shown in Figure 2.13c. It can be seen that the accuracy is well under 0.01 volts for the majority of the discharge, and the major source of error occurs when the experimental data stops at 2.5 volts, while the simulated data often continues. This error can be reduced by restricting the data set discharge voltages to terminate at exactly 2.5 volts. In Figure 2.13d, the corresponding SOC estimate errors can be seen for each of the predictive methods. The GBM performs the best, and also has the smoothest output, likely due to its ability to generalize, enabled by the creation of an

ensemble of weak learners rather than completely learning each data sample independently, as with RFs and DTs.

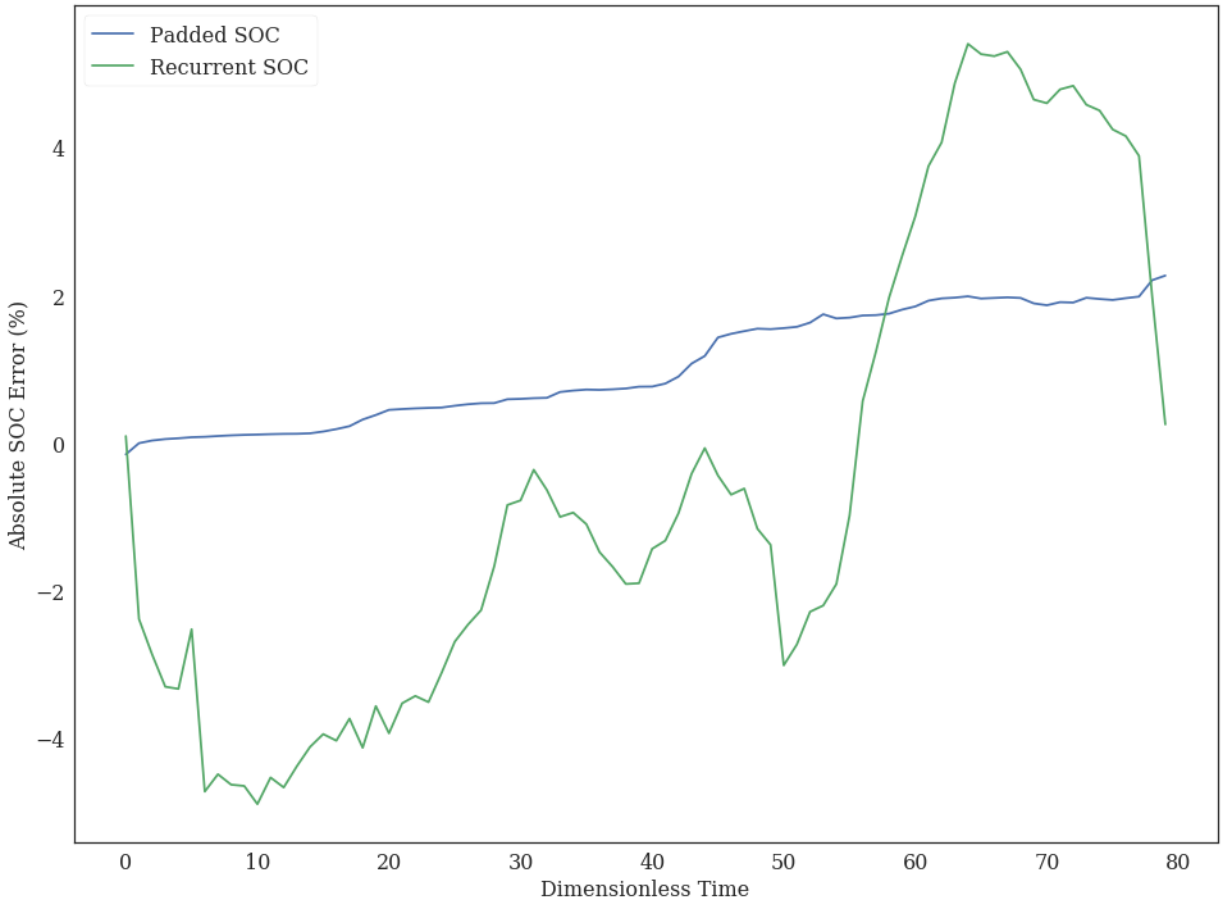


Figure 2.14: Effects of problem formulation on SOC estimation. The recurrent method uses only voltages [n-4...n] and the padded method uses [n=0...n], which results in significantly lower error for the experimental data set.

In order to combat the errors associated with the recurrent SOC prediction, it is possible to restructure the data once again in order to give only the information present at certain points of the discharge, like the recurrent method, but to include the voltage history of the battery in order to improve the SOC prediction accuracy, like the constant time model. To do this, the constant time data is modified such that any points past the current point

in time are padded with zeros. For instance, the first data point, when SOC is 100%, will contain just the first voltage from the discharge cycle followed by 99 zeros. The fifth data point will contain the first five voltages followed by 95 zeros. In this way, much more relevant information is available to the algorithm in order to predict SOC, and the results are significantly better, with a greatly reduced MSE of 1.27%, down from the previous best of 3.25%. The SOC prediction error is shown in Figure 2.14.

2.5 Perspective

One of the main advantages of using machine learning in a simulation application is for the creation of highly accurate, extremely fast and robust surrogate models. These models are approximations for the original model, in this case the P2D lithium-ion battery model, which is itself an approximation of reality.

There have been other instances in which researchers resort to approximations for algorithms, and must verify that the result remains meaningful. A good example from the battery community is the work by Doyle[40] using BAND(J) routines, which were restricted to a single dimension in x. In order to solve for the diffusion in the radial direction of the particle, rather than adding node points in the radial direction, the authors used Duhamels superposition theorem, which is typically used to give a solution for the model when the constant current boundary position is replaced with a known profile which varies dynamically in time. In the P2D model, this time-variant profile for porewall flux is not known a priori, and the authors were able to solve the model by approximating the unknown time integral. Another example in the battery modeling literature is the paper by Paxton and Newman.[50] Since a robust code that can solve when D is a constant was already present, they provided an approach for materials where D changes with θ, the state of charge. For example, by focusing just on the single particle, the authors compared a D(θ) model to a D(iapp) model and concluded that the D(θ) can be replaced by the D(iapp) model under a range of operating conditions, although the variation in the magnitude and direction of D(θ) will determine the accuracy. This means that the Duhamels superposition based model can

be used for D(θ) cases.

Similarly, a parabolic profile[51] approximation for single particle diffusion has been derived and compared to higher[52,53,54,55] order approximations for varying applied current. Both the parabolic profile and higher order approximations have successfully been applied to the P2D model and have been used for simulation, control, design, and estimation purposes.[51,56] The effective approximate model for solid phase diffusion is different for constant diffusivity, where the Galerkin approach[54] is valid, and for varying diffusivity, where the only valid approach is orthogonal collocation.[2,57] It is often possible to switch between different approximate models depending upon operating conditions using Faradays law for low rates, a parabolic profile for low rates, a higher order model for moderate rates, and boundary layer type finite difference or finite element models[7,58,59] for very high rates. As evidenced by many of these effective approximations, there are always pros and cons for such approximations. Sometimes, approximation is only the way to simulate[60] and, sometimes, approximation is necessitated by the availability of software and codes. Nevertheless, the approximations had always been validated by comparing with the full-order model and/or experimental data. Similarly, surrogate models from ML, if carefully developed, are expected to move the field forward significantly.

Future work includes the creation of surrogate models which are trained only at certain scales, but which can be extended to become part of a more complex model, much like the parabolic profile or similar approximations for a particular scale. An important consideration during the creation of these models will be the ensuring the conservation of flux, mass, and charge. For example, when solid phase diffusion is approximated with any method, the average concentration inside the particle is directly proportional to the applied current or porewall flux.[50,61] Enforcing this constraint on surrogate models will result in much faster convergence.

The success or failure of a machine learning project depends heavily upon the problem formulation and the desired outputs. For example, this project began with a specific goal to create a surrogate model which allows for the prediction of a voltage vs time discharge

curve for a given set of model input parameters. After varying degrees of success with this approach, a new goal was set estimate the SOC as well as the voltage, but using the same data set. While the approach adopted for this task was reasonable, given the constant discharge rate assumption, it would have failed for a varying discharge rate. However, the P2D model is capable of calculating the SOC by reporting the fraction of lithium present in the anode and cathode, which are a much better means of calculating SOC than using voltage. Since those data were not stored when creating the data set, they were not used for this paper, but a new surrogate model could easily be created from a data set containing these lithium concentrations, electrolyte concentrations, or other internal states, which may have much more success in predicting SOC than the current surrogate model and may prove more useful, especially in control applications.

2.6 Conclusion

Machine learning is a promising new field of research that gives the ability to create surrogate models for faster execution of higher fidelity models, as well as giving the ability to restructure the input-output structures of the model without compromising accuracy. In this work, decision trees, random forests, and gradient boosted machines have been evaluated as candidates for the creation of surrogate models for the porous electrode pseudo two-dimensional physics-based lithiumion battery model during a 2C constant-current discharge event. In this scenario, the surrogate models perform exceptionally well at nextvoltage prediction, but due to the structure of the data set, have no ability to function at currents that are not 2C and chemistries that are dissimilar to that of the simulated data set. In this version of the P2D model, the open circuit potential for the positive electrode is fit using a piecewise continuous linear curve. If a data set were created that varied this curve significantly, it may be possible to express different chemistries, or to have multiple chemistries contained in the same data set. To incorporate them, a new model could be trained using the same techniques demonstrated here.

It can be seen that while the voltage prediction using recurrent methods is exceptional, the

state of charge (SOC) estimation is fairly poor due to the large variance in the terminal times of the simulated discharge curves. The same variability that allows for high-accuracy voltage prediction causes higher error in SOC estimates. By restructuring the data set, it is possible to improve the resulting estimate for SOC, which is highly dependent upon the discharge history of the battery. The ability to solve high fidelity models with low computational cost has many applications, including real-time control in BMS applications, fast optimization for battery design, and multiscale modeling. Future work will include the use of more complex ML algorithms, such as ANNs, which take longer to train, but are similarly fast to execute and offer superior interpolation performance in addition to much smaller model size on disk. Additionally, while decision trees can only select from training set data, and hence can only give physically meaningful results while the training data remains physically meaningful, this may not be the case for other models while interpolating or extrapolating, and may not be the case for random forests. In random forests, the outputs of several curves are directly averaged together, often resulting in a curve that is closer in error but qualitatively dissimilar the curve may be blocky, with sudden drops in voltage as the combined discharge curves are averaged. Future work will also examine using multiple models whose algorithmic layouts are based on physically relevant relationships, and work will be done to ensure that the outputs remain meaningful. Although surrogate models based on the P2D model are demonstrated in this paper, the concept is applicable for detailed 2D and 3D models for batteries, including multiscale thermal models.

A saved version of the recurrent model can be found at https://github.com/nealde/P2D_Surrogate, along with instructions for downloading and executing it locally.

Acknowledgements

The authors thank the Department of Energy (DOE) for providing partial financial support for this work, through the Advanced Research Projects Agency (ARPA-E) award number DE-AR0000275. The authors would also like to thank the Clean Energy Institute (CEI) at the University of Washington (UW) and the Data Intensive Research Enabling Clean Tech-

nologies (DIRECT), a UW CEI program funded by the National Science Foundation (NSF), in addition to the Washington Research Foundation (WRF) for their monetary support of this work. The battery modeling work has been supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Vehicle Technologies of the U. S. Department of Energy through the Advanced Battery Material Research (BMR) Program (Battery500 Consortium).

Chapter 3

ON THE CREATION OF A CHESS-AI-INSPIRED PROBLEM-SPECIFIC OPTIMIZER FOR THE PSEUDO TWO-DIMENSIONAL BATTERY MODEL USING NEURAL NETWORKS

Note: this chapter was published as an article

- Dawson-Elli, N.; Kolluri, S.; Mitra, K.; Subramanian, V. R. On the Creation of a Chess-AI-Inspired Problem-Specific Optimizer for the Pseudo Two-Dimensional Battery Model Using Neural Networks. J. Electrochem. Soc. 2019, 166 (6), A886A896. https://doi.org/10.1149/2.1261904jes.

Abstract

In this work, an artificial intelligence based optimization analysis is done using the porous electrode pseudo two-dimensional (P2D) lithium-ion battery model. Due to the nonlinearity and large parameter space of the physics-based model, parameter calibration is often an expensive and difficult task. Several classes of optimizers are tested under ideal conditions. Using artificial neural networks, a hybrid optimization scheme inspired by the neural network-based chess engine DeepChess is proposed that can significantly improve the converged optimization result, outperforming a genetic algorithm and polishing optimizer pair by 10-fold and outperforming a random initial guess by 30-fold. This initial guess creation technique demonstrates significant improvements on accurate identification of model parameters compared to conventional methods. Accurate parameter identification is of paramount importance when using sophisticated models in control applications.

3.1 Introduction

Lithium ion batteries are complex electrochemical devices whose performance is dictated by design, thermodynamic, kinetic, and transport properties. These relationships result in nonlinear and complicated behavior, which is strongly dependent upon the conditions during operation. Despite these complexities, lithium ion batteries are nearly ubiquitous, appearing in cell phones, laptops, electric cars, and grid-scale operations.

The battery research community is continually seeking to improve models which can predict various state of the battery. Due to this drive, a multitude of physics-based models which aim to describe the internal processes of lithium ion batteries can be found, ranging in their complexity and accuracy from computationally expensive molecular dynamics simulations down to linear equivalent circuit and empirical models. Continuum scale models such as the single particle model[1] and pseudo two-dimensional model (P2D)[4,2,6,40] exist between these extremes and trade off some physical fidelity for decreased execution time. These continuum models are generally partial differential equations, which must be discretized in space and time in order to be solved. In an earlier work, a model reformulation based on orthogonal collocation[4] was used to greatly decrease solve time while retaining physical validity, even at high currents.

Sophisticated physics-based battery models are capable of describing transient battery behavior during dynamic loads. Pathak et. al.[8] have shown that by optimizing charge profiles based on internal model states, real-world battery performance can be improved, doubling the effective cycle life under identical charge time constraints in high current applications. Other work[62] has shown similar results, lending more credence to the idea that modeled internal states used as control objectives can improve battery performance. However, in order for these models to be accurate, they must be calibrated to the individual battery that is being controlled. This estimation exercise is a challenging task, as the nonlinear and stiff nature of the model coupled with dynamic parameter sensitivity can wreak havoc on black-box optimizers. Going into different approaches for estimating parameters is beyond

the scope of this paper.

Data science, often hailed as the fourth paradigm of science[11], is a large field, which covers data storage, data analysis, and data visualization. Machine learning, a subfield of data science, is an extremely flexible and powerful tool for making predictions given data, with no explicit user-parameterized model connecting the two. Artificial neural networks, and the most popular form known as deep neural networks (DNNs), are extremely powerful learning machines which can approximate extremely complex functions. Previous work has looked at examining the replacement of physics-based models with decision trees, but this has proven to be moderately effective, at best[17,63,64,65]. In this work, neural networks are used not to replace the physics-based model, but to assist the optimizer. The flexibility in problem formulation of neural networks affords the ability to map any set of inputs to another set of outputs, with statistical relationships driving the resulting accuracy. In this instance, the aim is to use the current parameter values coupled with the value difference between two simulated discharge curves to refine the initial parameter guess and improve the converged optimizer result. The outputs of the neural network will be the correction factor which, when applied to the current parameter values, will approximate the necessary physics-based model inputs to represent the unknown curve.

One interesting aspect of data science is the idea that it is difficult to create a tool which is only valuable in one context. For example, convolutional neural networks were originally created for space-invariant analysis of 2D image patterns[66], and have been very successful in image classification and video recognition[67], recommender systems[68], medical image analysis[69], and natural language processing[70]. To this end, the comparative framework inspired by DeepChess[71] has found effective application in the preprocessing and refinement of initial guesses for optimization problems using physics-based battery models, where the end goal of the optimization is to correctly identify the model parameters that were used to generate the target curves. Specifically, the problem formulation is inspired by DeepChess the ideas of shuffling the data each training epoch and of creating a comparative framework were instrumental, and the inspiration is the namesake of this work. The problem of accurate

parameter identification and model calibration is paramount if these sophisticated models are to find applications in control and design. The process of sampling the model, creating the neural networks, and analyzing the results are outlined for a toy 2-dimensional problem using highly-sensitive electrode thicknesses and a 9-dimensional problem, where the parameters vary significantly in sensitivity, scale, and boundary range.

In this article, the sections are broken up as follows: the problem formulation and an overview of neural networks are discussed first, followed by the sensitivity analysis and a two-dimensional demonstration of the neural network formulation. Then, the problem is discussed in 9 dimensions, where the neural network recommendation is restricted to a single function call, meaning that the neural network performs a single refinement of an initial guess, and different optimization algorithms are explored. In the next section, the same 9 dimensions are used, but the neural network is randomly sampled up to 100 times, and the best guess from the resulting refinements is used in the optimization problem. This is shown to dramatically improve the converged result. The resulting neural network refinement system is only applicable to this model, with these varied parameters, over these bounds and at the sampled current rates, which effectively makes the neural network a problem-specific optimizer.

3.2 Ideas of Data Science and Problem Formulation

Traditionally, data science approaches to optimization or fitting problems would include creating a forwards or inverse model, as has already been explored using decision trees[17]. In the forwards problem formulation, a machine learning algorithm would map the target function inputs to the target function outputs, and then traditional optimization frameworks would use this faster, approximate model rather than the original expensive high fidelity model, with the optimization scheme remaining unchanged. However, this is not the most efficient way to use the data that has been generated, and comes with additional difficulties, including physical consistency[24]. The concerns with physical consistency stem from machine learning algorithms native inability to enforce relative output values, which can violate

output conditions like monotonicity or self-consistency.

An additional problem formulation style, known as inverse problem formulation, seeks to use machine learning in order to map the outputs of a function to its inputs, seeking to create an O(1) optimization algorithm tailored to a specific problem. While this ambitious formulation will always provide some guess as to model parameters, they are subject to sensitivity and sampling considerations and a large amount of data is needed for these inverse algorithms to be successful in nonlinear systems[17].

In this work, the flexibility of neural network problem formulation is leveraged in order to create a comparative framework. A comparative framework could be considered a way of artificially increasing the size of the data set, one of several tricks which are common in the machine learning space known as data augmentation[72]. Another popular technique, especially in 2-dimensional image classification, is subsampling of an image with random placement. This forces the neural network to generalize the prediction with respect to the location of learned features and, more importantly, allows the neural network to leverage more information from the same data set by creating more unique inputs to the model.

In order to make the model nonlinear, activations functions are applied element-wise during neural network calculations. In this work, an exponential linear unit (eLU)[73] was used, the input-output relationship of which is described in Figure 3.1. This activation function was chosen to keep the positive attributes of rectified linear units (ReLU)[74] while preventing ReLU die-off associated with zero gradients at negative inputs. Another consideration when selecting eLU as the activation function was to avoid the vanishing gradient problem[75]. During training, error is propagated as gradients from the last layers of the neural network back to the top layers. When using activation functions which can saturate, such as sigmoids or hyperbolic tangents, the gradients passed through these activation functions can be very small if the activation function is saturated. This effect is compounded as the depth of the neural network increases. By selecting an activation function which can only saturate in one direction, and by restricting the depth of the neural network to only three layers, vanishing gradients can be avoided. Additionally, each of the neural networks used in this work are of

identical size, consisting of hidden layers of 95, 55, and 45 nodes, and each neural network was trained for a maximum of 500 epochs. Mean squared error was used as the loss function for each neural network, and Adam[76] was the training optimizer of choice.

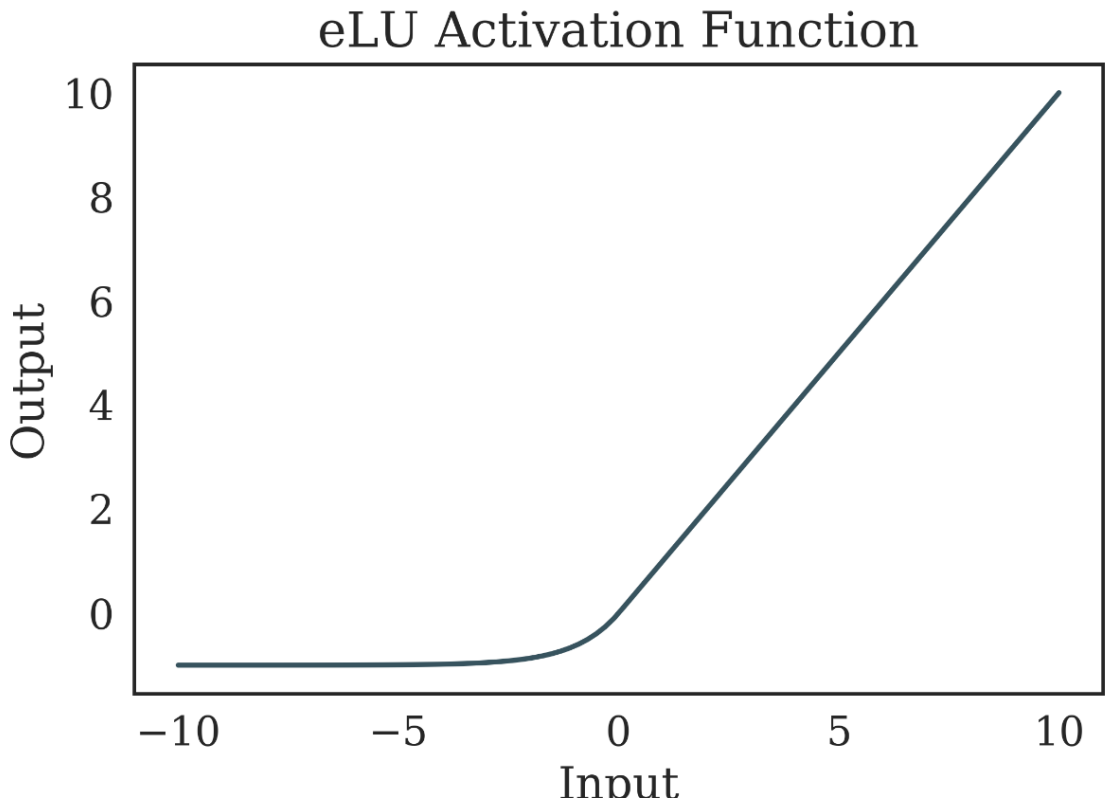


Figure 3.1: Exponential Linear Unit activation function input-output mapping. The activation function adds nonlinearity to the neural network which allows the outputs to capture more complex phenomena.

3.3 2D Application and Sensitivity Analysis

The version of the P2D model used in this work has 24 parameters which can be modified in order to fit experimental data; a set composed of transport and kinetic constants. In order to reduce this large dimensionality to something more reasonable, a simple one-at-a-time sensitivity analysis was performed. A range for each parameter was established based on

Table 3.1: Sensitivity and bounds of available P2D Model parameters. Bolded values are selected for use in the 9-dimensional analysis which makes up the bulk of this work.

Parameter	Description	Lower Bound	Upper Bound	Sensitivity	Units
D1	**Electrolyte diffusivity**	**7.50E-11**	**7.50E-09**	**0.0053579**	$\frac{m^2}{s}$
Dsn	**Solid-phase diffusivity (n)**	**3.90E-15**	**3.90E-13**	**0.0143956**	$\frac{m^2}{s}$
Dsp	**Solid-phase diffusivity (p)**	**1.00E-15**	**1.00E-13**	**0.001759**	$\frac{m^2}{s}$
Rpn	Particle radius (p)	2.00E-07	2.00E-05	0.0841821	m
Rpp	Particle radius (n)	2.00E-07	2.00E-05	0.1211896	m
brugp	Bruggeman coef (p)	3.6	4.4	0.0312093	
brugs	Bruggeman coef (s)	3.6	4.4	0.0702692	
brugn	Bruggeman coef (n)	3.6	4.4	0.0032529	
ctn	**Maximum solid phase concentration (n)**	**27499.5**	**33610.5**	**0.2041194**	$\frac{mol}{m^3}$
ctp	**Maximum solid phase concentration (p)**	**46398.6**	**56709.4**	**0.0373641**	$\frac{mol}{m^3}$
efn	Filler fraction (n)	0.02934	0.03586	0.0827452	
efp	Filler fraction (p)	0.0225	0.0275	0.012084	
en	**Porosity (n)**	**0.4365**	**0.5335**	**0.1816505**	
ep	**Porosity (p)**	**0.3465**	**0.4235**	**0.0379277**	
es	Porosity (s)	0.6516	0.7964	0.0010536	
iapp	Current density	13.5	16.5	0.9174174	$\frac{A}{m^2}$
kn	Reaction rate constant (n)	5.03E-12	5.03E-10	0.0028494	$\frac{\frac{mol}{sm^2}}{(\frac{mol}{m^3})^{1+\alpha_{a,i}}}$
kp	Reaction rate constant (p)	2.33E-12	2.33E-10	0.0032171	$\frac{\frac{mol}{sm^2}}{(\frac{mol}{m^3})^{1+\alpha_{a,i}}}$
lp	**Region thickness (p)**	**8.80E-06**	**0.00088**	**0.7764841**	m
ln	**Region thickness (n)**	**8.00E-06**	**0.0008**	**0.7386793**	m
ls	Region thickness (s)	2.50E-06	0.00025	0.0398143	m
σ_n	Solid-phase conductivity (n)	90	110	7.45E-06	$\frac{S}{m}$
σ_p	Solid-phase conductivity (p)	9	11	1.74E-05	$\frac{S}{m}$
t1	Transference number	0.3267	0.3993	8.88E-05	

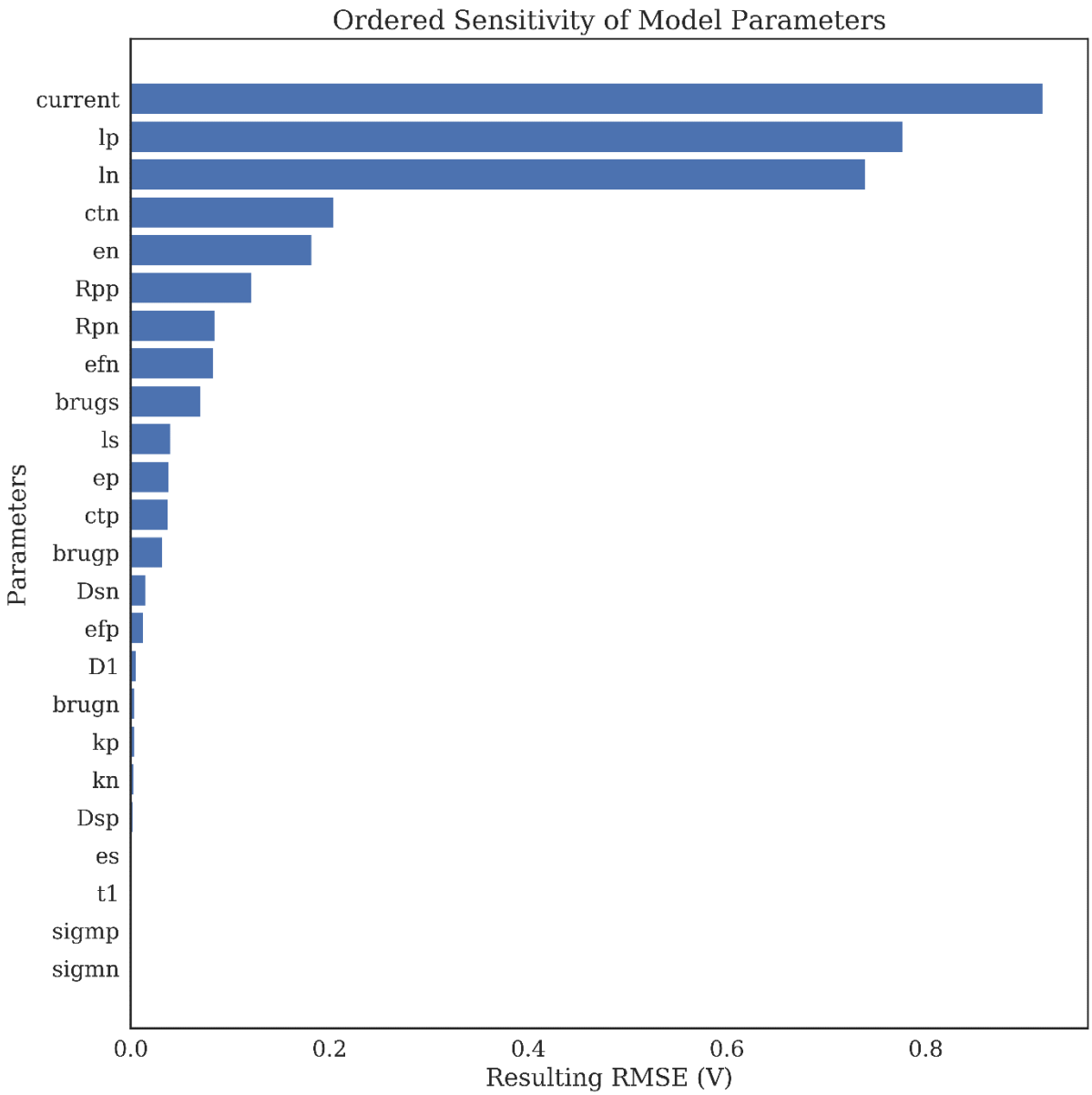


Figure 3.2: Ordered sensitivity of model parameters, calculated by measuring the root mean squared error change in output voltage associated with the values at the bounds of the parameters. It is important to note that this is a function of the current parameter values.

a combination of literature values[17] and model solution success, and an initial parameter set was chosen. For each of the parameters, the values were permuted from the initial

value to the upper and lower bounds. The resulting change in root mean squared error (RMSE) was used as the metric for sensitivity. The results are summarized along with the bounds in Table 3.1 and Figure 3.2. These sensitivities were used to inform the parameter down-sampling to 9 dimensions. The selected parameters are bolded in Table 3.1. The diffusivities were selected for their relatively low sensitivity and wide bounds, while the maximum concentrations in the positive and negative electrodes, porosity in positive and negative electrodes, and thicknesses of positive and negative electrodes were selected for a combination of high sensitivity and symmetry. In general, it would be advisable to select only the most sensitive parameters when fitting, regardless of symmetry. However, since these sensitivities are only locally accurate, symmetry was prioritized above absolute sensitivity for interpretability. The diffusivities were selected to examine how the DNN performs when guessing parameters with low sensitivity.

The selected parameters, with the exception of the diffusivities, were varied uniformly across their bounds in accordance with a quasi-random technique known as Sobol sampling[77]. The diffusivities had a very large parameter range, roughly two orders of magnitude, so log scaling was used before a uniform distribution was applied, which ensured that the sampling was not dominated by the upper values from the range. The main benefit of the Sobol sampling technique is that it more uniformly covers a high dimensional space than nave random sampling without the massive increase in sampling size required to perform a rigorous factorial sweep. The downside is that it is not as statistically rigorous as other techniques like fractional factorial or Saltelli sampling, which have the added advantages of allowing for sensitivity analyses[78]. However, the number of repeated levels for each parameter is much higher with these methods of sampling, which results in worse performance when used as the sampling technique for training a neural network.

Two of the most sensitive parameters, the thicknesses of the negative and positive electrodes, were selected to demonstrate the non-convexity of the optimization problem while performing parameter estimation for the P2D model. A relatively small value range was selected, and 50 discrete levels for each parameter were chosen, resulting in 2500 discharge

curves, which were generated using each of the values. Then, a random, near-central value was selected as the true value, corresponding to 0 root mean squared error (RMSE), and an error contour plot was created as shown in Figure 3.3. The blue dot represents the true values of parameters and an RMSE of 0. As these discharge curves are simulated, their final times can vary. In order to make the two curves comparable, the target curve is padded with values equal to the final voltage of 2.5 V, and any curve which terminates early is also padded with values equal to 2.5 V. Another option would be to simply interpolate at the target data points and throw away any data past either terminal time, but this would change the sensitivity of the parameters arbitrarily, as the end of the curves are thrown away each time. Several example optimization problems are demonstrated with their lines traced on the contour plot, showing the paths the algorithms have taken in their convergence. The demonstrated optimization method here is Nelder-Mead, a simplex-type method implemented in Scientific Python (Scipy), a scientific computing package in Python[79].

After using Nelder-Mead, a deep neural network was implemented which seeks to act as a problem-specific optimizer, comparing two model outputs and giving the difference in parameters used to create those two curves. The exact formulation of the neural network is discussed in the next section. As seen in Figure 3.4, when starting at some of the same points as the above optimizer, the neural network gets extremely close to the target value in a single function call, even with only 100 training examples. In this instance, the neural network has demonstrated added value by outperforming the look-up table, meaning that the estimates from the neural network are closer than the nearest training data.

When looking at RMSE as the only metric for difference between two curves, this seems to be an extremely impressive feat how can all of this information be extracted from a single value? In the case of optimization, the goal is simply to minimize a single error metric, an abstraction from the error between two sets of time series data, in this case. However, during this abstraction, a lot of information is destroyed in particular, exactly where the two curves differ is completely lost. A similar example can be found in Anscombes Quartet[80], which refers to four data sets which have identical descriptive statistics, and yet

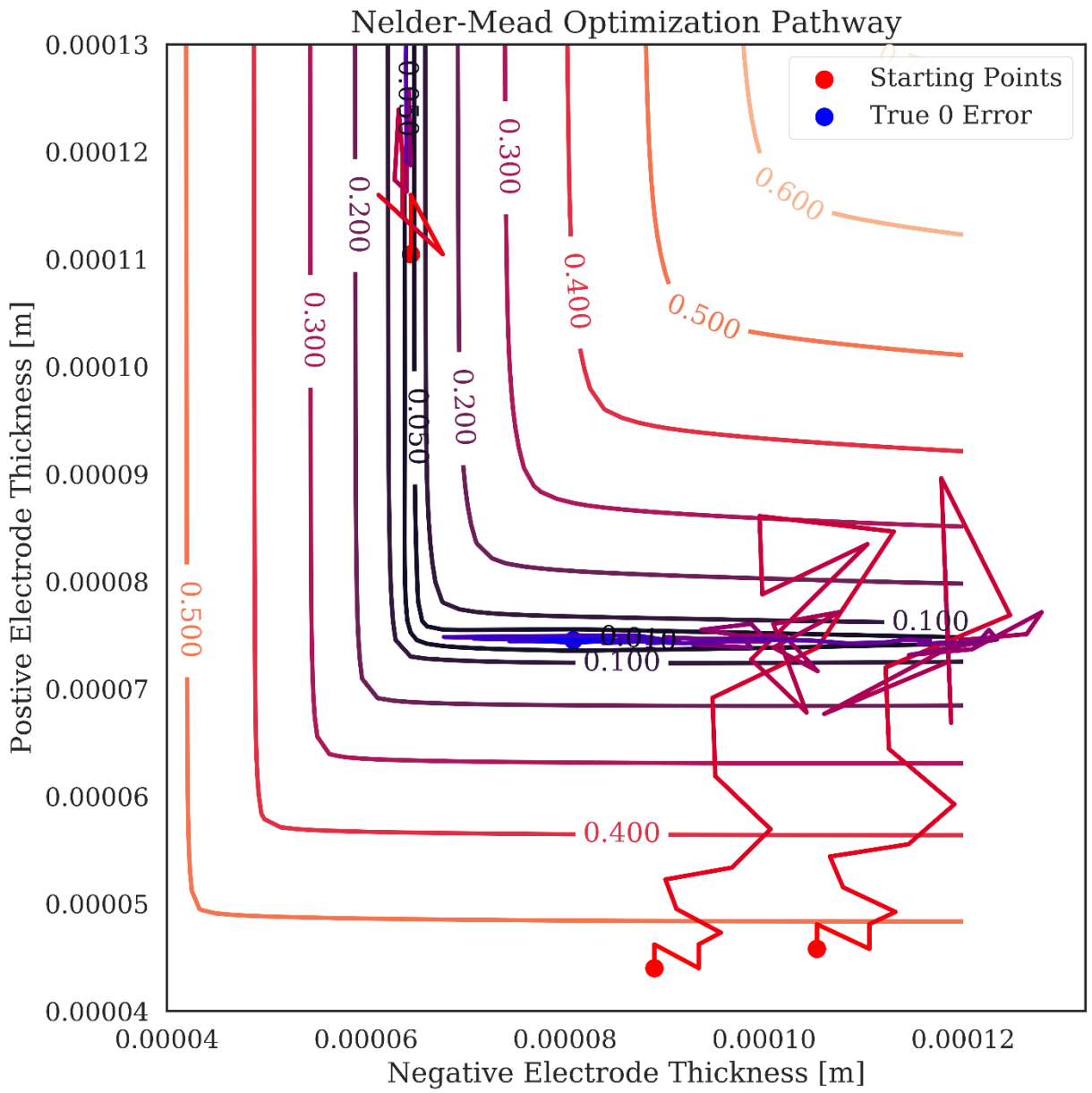


Figure 3.3: 2D error contour plot demonstrating the optimization pathway based on a set of initial guesses. The Nelder-Mead algorithm fails to achieve an accurate result from one of three starting points due to the nearest error trough, which drives the optimization algorithm away from the true 0 error point. It should be mentioned that Nelder-Mead cannot use constraints or bounds.

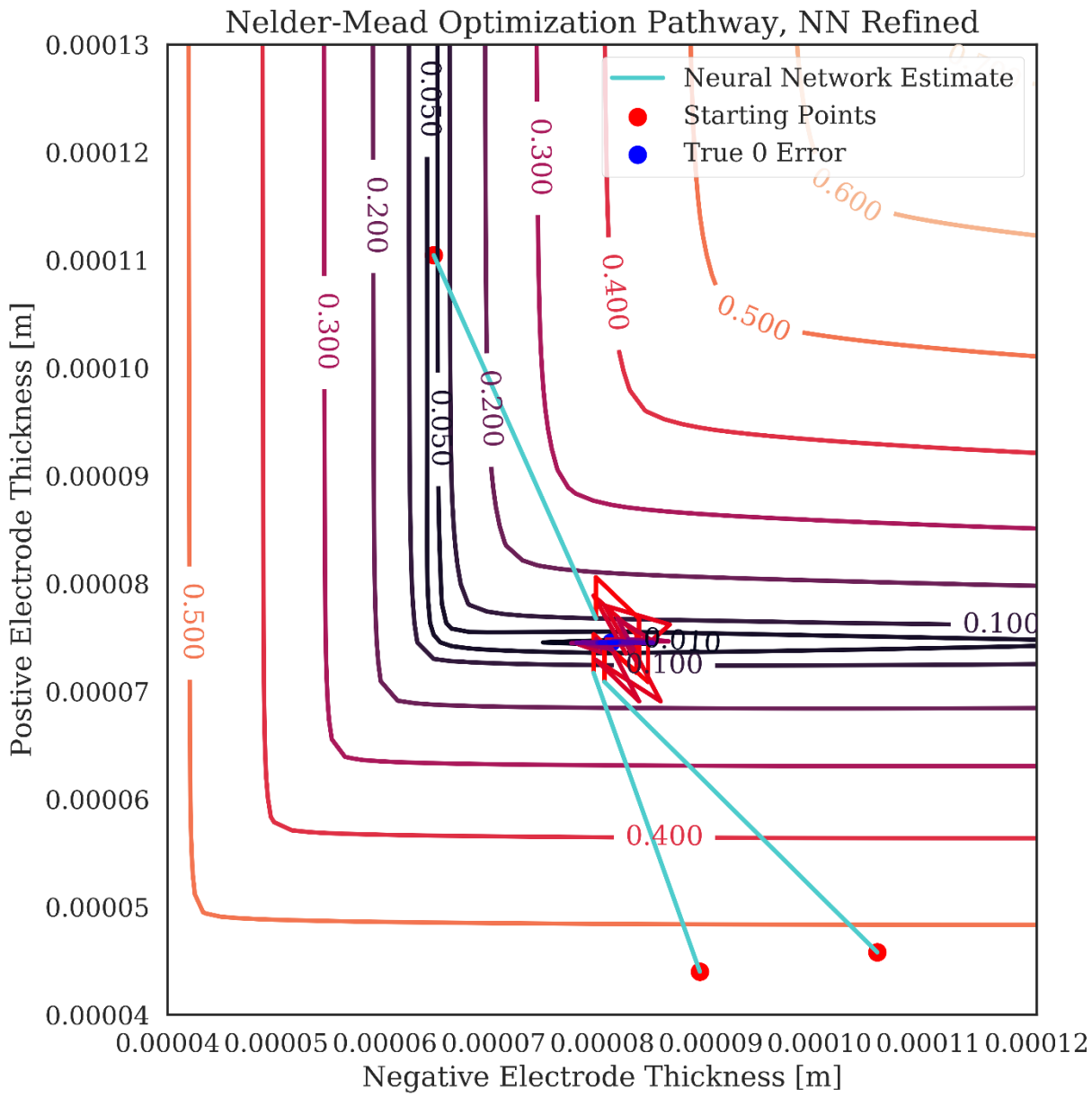


Figure 3.4: Using the trained neural network as a refinement of the initial guess, all starting points can be made to converge. The changes made by the neural network are highlighted in teal, and the red lines fade to blue as the optimization progresses. In this instance, all optimization algorithms successfully arrive at the true 0 error location.

are qualitatively very different from one another. In Figure 3.5, the differences in the time series are demonstrated, and it is clear that the curves look very different depending upon which electrode is limiting, the positive electrode or the negative electrode. This information is leveraged in the neural network, but is lost in the nave optimization. The next section looks to apply this same idea to a higher dimensional space and quantify the benefit that is present when using a neural network to improve a poor initial guess, as is the case with an initial model calibration. Figure 3.6 demonstrates how the time series data can differ with respect to positive and negative electrode thicknesses in the 2D problem.

3.4 Methodology

In this section, the process of using a DNN as a problem-specific optimizer for a 9-dimensional physics-based lithium ion battery model is outlined. The DNN used is typical, with 3 hidden layers comprising 95, 55, and 45 nodes each with 9 nodes on the output dimension, one for each of the model parameters to be estimated. Before being input to the model, each of the parameters was normalized over the range [0,1]. This is separate from sampling, and is done in order to allow the neural network to perform well when guessing large or small numbers. When training a neural network to predict values which vary from one another by several orders of magnitude, it is important to scale the inputs and outputs such that some values are not artificially favored by the loss algorithm due to their scaling. Additionally, when calculating the gradients during training, the line search algorithm assumes all of the inputs are roughly order 1. Another consideration is that the initial values of neural networks are generated under the assumption that the inputs and outputs will be approximately equal to 1. When calculating the loss associated with a misrepresented value, in the case of diffusivities where the target value is 1e-14, any value near 0 will be considered extremely close to accurate. It is important to note that this normalization is completely separate from the sampling distribution, its only purpose is to force the data to present itself as the neural network engines expect.

The voltages, already fairly close to 1 in value, were left unscaled and unmodified for

simplicity. The nonlinearity comes from the eLU activation function[73], which exponentially approaches -1 below 0 and is equal to the input above 0. This is similar to leaky ReLUs[81], or LReLUs, which simply have two lines of different slopes that intersect at 0. The point of having a nonzero value below an activation input of 0 is to prevent a phenomenon known as ReLU die-off, in which the weights or biases can become trained such that a neuron never fires again, as the activation function input is always below 0. This can effect up to 40% of neurons during training. Giving the activations functions a way to recover when an input is below 0 is a way of combatting ReLU die-off, and often results in faster training and better error convergence. Additionally, having an activation function which can output negative numbers reduces the bias shift that is inherent to having activation functions which can only produce positive numbers, as with ReLUs, which can slow down learning[82]. Additionally, non-saturating activation functions like eLUs and ReLUs combat vanishing gradients, which can cause significant slowdowns during training when using saturating activations functions like hyperbolic tangents[83].

The training protocol for this work was adapted from Deepchess[71], which created a new training set for each epoch by randomly sampling a subset of the data in a comparative manner. This process is leveraged here, where a new training set is generated at each training epoch. For large enough data sets, this can greatly limit overfitting, as the majority of training data given to the neural network is only seen once. The process is described visually in Figure 3.7, and involves 2 sets of parameter inputs and model outputs, A and B. In the neural network formulation, set A is the current initial guess, while set B is a simulated representation of experimental data, where the numerical model inputs would be unknown, but the desired simulated discharge curve shape is known. The DNN inputs are the scaled parameter set A and the time series difference between the discharge curves generated by parameter sets A and B. The DNN output is a correction factor, calculated as Output= B-A. In this way, knowing the correction factor and numerical model inputs A allows for the reconstruction of the desired numerical model inputs, B. During training, the DNN learns to associate differences in the error between the output curves and the current numerical model

input parameters with the desired correction factor which will approximate the numerical model input parameters used to generate the second discharge curve.

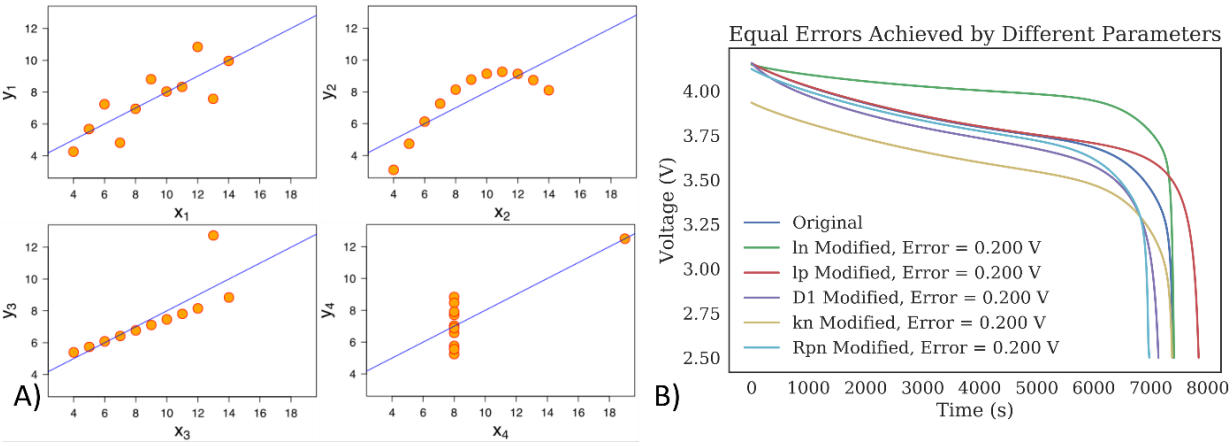


Figure 3.5: a) Anscombe's Quartet, a set of 4 distributions which have identical second-order descriptive statistics, including line of fit and coefficient of determination for that fit. b) An electrochemical equivalent where the RMSE is equal to 0.200V, but the curves are qualitatively very different. This was achieved by modifying different single parameters to achieve the error.

For the 9-dimensional problem, the input dimension was 1009 the values of the current scaled parameters and two concatenated discharge curves, each of length 500. The size of the training data was varied between 500 and 200,000 samples for the same 9 parameters with the same bounds. For each new training set, the model hyper-parameters remained the same a batch size of 135, trained for 500 epochs with a test patience of 20, meaning that if test error had not improved in the past 20 epochs, the training ended early, which was used to combat overfitting. The smaller data set models had 50% dropout applied to the final hidden layer in order to fight overfitting, which was present in the smaller data sets. In order to enforce the constant-length input vector requirement of the neural network, the discharge data was scaled with respect to time according to $t = (4000/i_{app})$ uniformly spread over the 500 times steps. Local third-order fits were used to interpolate between the numerical time points. In this text, iapp values of 15, 30, 60, and 90 A/m^2 are referred to as 0.5C, 1C, 2C,

and 3C, respectively. Additionally, although the neural networks are labeled by the size of the generated data sets, ranging from 500 to 200,000, only $3/4^{ths}$ of the data were used for training, which is occasionally alluded to in the text. The neural network labeled as needing 200,000 sets of data will have trained on 150,000, while the remainder is held for validation. For all neural networks, the same static test set of 10,000 parameter pairs was used so that the results are directly comparable.

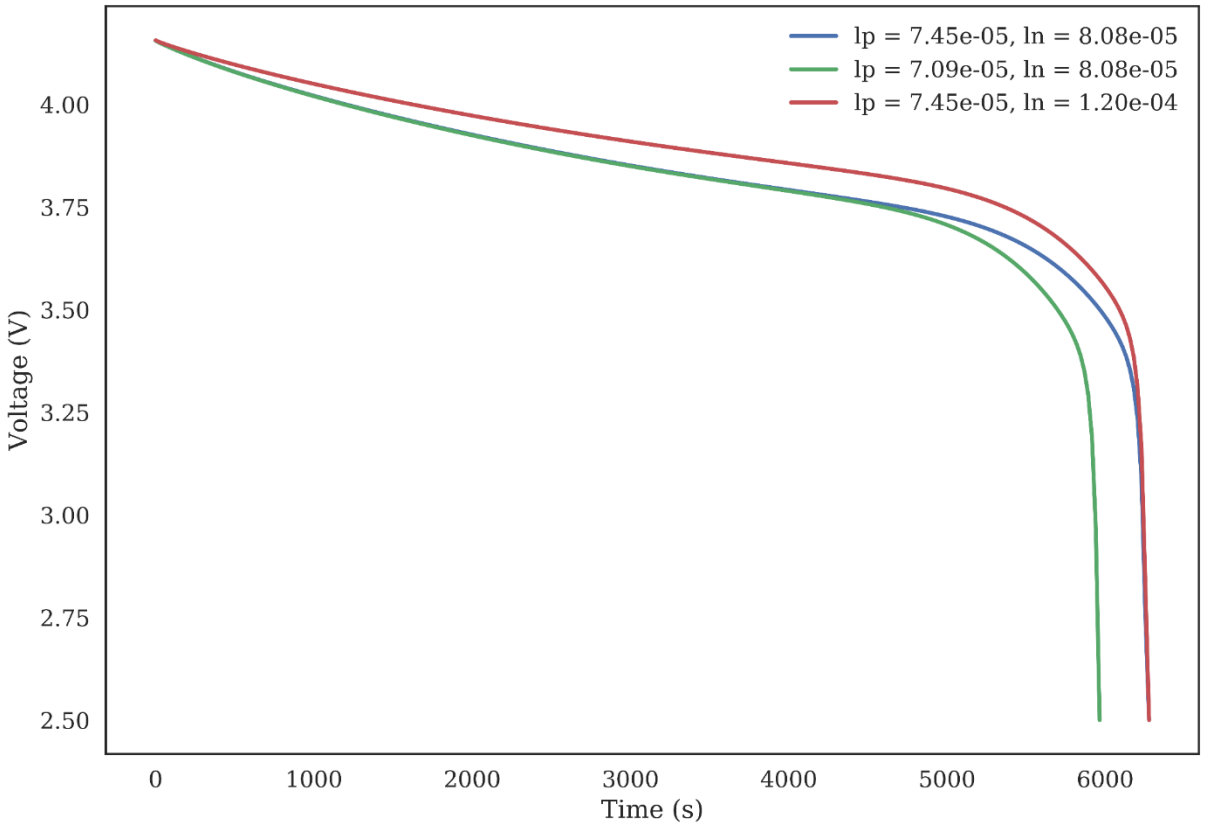


Figure 3.6: Differences in time series from the 2D example. Changing the positive thickness extends the discharge curve with respect to time, but increasing the negative thickness raises the average voltage without extending the length of the discharge curve.

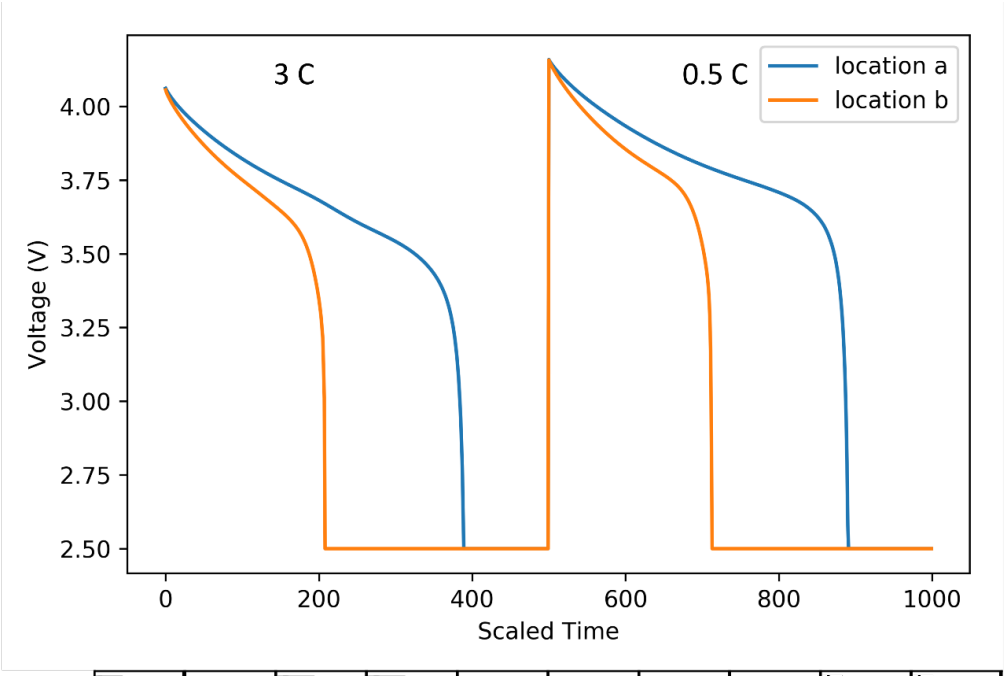
The purpose of this formulation comes down to two practical considerations information extraction and physical self-consistency. If a forward model were to be trained on the

same data, the model would be responsible for producing physically self-consistent results for instance, for a constant applied current, the voltage would need to be monotonically decreasing. This requirement is extremely difficult to achieve, especially with finely time-sampled data the noise will quickly climb above the voltage difference between adjacent points, resulting in an a-physical discharge. In the inverse formulation, where a discharge curve is mapped onto the input parameters via a neural network, each data point only gets to be used once. This results in significantly poorer guesses for a data set of the same size when compared to this formulation. This is due to the fact that the neural network gets many unique examples where the parameters to estimate are varied for a data set of 2000 simulations, there are 2000 unique error-correction mappings for every value. This allows the neural network to more intimately learn how each parameter varies as a function of the input error by squaring the effective size of the data set.

There is an important consideration which can be easily illustrated. In the 2D example, an error contour plot was generated which demonstrated the RMSE between curves as a function of their electrode thicknesses with the neural networks mapped corrections superimposed. At first, the performance may look relatively poor, even though it beats the lookup table performance, but it is important to note that this could have been replicated using any of the points and the performance would have been comparable. That is to say that the neural network is not simply learning to navigate one error contour, it is learning to navigate every possible constructed error contour and is able to interpolate between training points in a way that a lookup table of the training data cannot.

3.5 9D Application and Analysis

Once the model was trained, it was used as a first function call in a series of optimization schemes. Using three different optimizers, a pre-determined set of 100 A-B pairs was created which mimicked the initial and target discharge curves of a classic experimental fitting optimization problem. Here, however, the concerns about the model being able to adequately match the experimental data are removed, meaning that any measurable error between the



Set	D1	Dsn	Dsp	ctp	ctn	en	ep	lp	ln
A	1.89E-09	2.74E-15	1.35E-12	3.21E+04	5.29E+04	3.66E-01	3.58E-01	3.85E-05	3.71E-05
B	1.33E-08	6.15E-12	1.26E-15	2.86E+04	4.96E+04	2.07E-01	2.69E-01	7.45E-05	5.74E-05

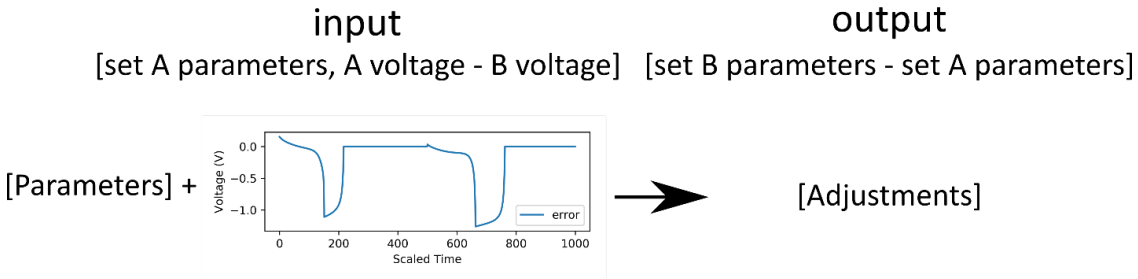


Figure 3.7: A visual representation of the proposed training paradigm. Given some set of model inputs A and another set B, there will be some error between the simulated discharge curves. The goal is to traverse the 9-dimensional space to arrive at the model inputs B. Numerical model inputs A are scaled to be on the set [0,1], concatenated with the error between the discharge curves, and the neural network output is calculated as parameter values A subtracted from parameter values B. In this way, to reconstruct an estimate for parameters B, the NN output must be added to the initial guess A. The flat lines after simulated discharge are artifacts of the constant-length input vector requirements of neural networks, and are not from the model simulation.

Table 3.2: Prediction error at unseen 1C current rate after optimization convergence at 0.5C and 3C rates for various black box optimizers, both with and without neural network initial guess refinement.

		SLSQP		Nelder-Mead		L-BFGS-B		GA	
Optimizer	**Initial Error (V)**	**Function calls**	**RMSE at 1C (V)**	**Function calls**	**RMSE at 1C (V)**	**Function calls**	**RMSE at 1C (V)**	**Function calls**	**RMSE at 1C (V)**
no Neural Network	0.3750	78	0.2236	750	0.0875	418	0.1064		
200k samples	0.0928	46	**0.0727**	577	**0.0218**	489	**0.0283**	17852	0.0470
50k samples	0.0936	71	0.0727	589	0.0277	430	0.0320	4783	0.0501
20k samples	0.0896	51	0.0862	596	0.0264	442	0.0341	2081	0.0511
5k samples	0.2970	111	0.0895	670	0.0533	448	0.0528	765	0.0547
2k samples	0.3273	67	0.1285	648	0.0766	450	0.0751	519	0.0560

converged value and the target curves are due to optimizer limitations and not due to an inability of the model to describe the experiment.

As shown in Table 3.3, passing the initial simulation through the neural network optimizer can drastically reduce the error at convergence, improving the final converged error by 4-fold. The optimization methods tested here include Sequential Least Squares Programming (SLSQP)[84], Nelder-Mead[85], the quasi-newton, low-memory variant of the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) algorithm (L-BFGS-B)[86], and a genetic algorithm (GA)[87]. Each of these optimization methods works fundamentally differently, and they were selected in order to adequately cover the available optimizers. All of the methods are implemented by Scipy[79] in Python. SLSQP and L-BFGS-B are bounded optimizers, while Nelder-Mead cannot handle constraints or bounds.

The first column of Table 3.2 shows the average initial error between points A and B, which equals 375 mV. After passing these relatively poor initial guesses through the neural networks once, the errors are demonstrably improved. Interestingly, although the 20k sample neural network creates guesses with a lower initial error on the unseen data, the converged results after optimization are consistently worse. This could be a coincidence of sampling, and can likely be attributed to the fact that the parameters given to the DNN do not have identical sensitivity in the physical model. Looking at the values of the relative error of the

parameters, as in Table 3.3, the models stack according to intuition, with the initial guesses being extremely wrong, and the neural networks performing significantly better, ordered in performance by the size of their training data. Note that these massive parameter errors are possible because the diffusivities vary by three orders of magnitude. These values were calculated on the parameters after they had been descaled, and as such, the effects of the accuracy in the diffusivities are likely dominant due to their massive variance relative to the other values.

The genetic algorithm is implemented in the differential evolution function through Scipy[87] and the number of generations and size of the generations were varied to reflect the training size divided by the number of iterations, leaving between 20,000 and 200 function evaluations per optimization. Unfortunately, the genetic algorithms could not be seeded with any values, which means that the neural network outputs could not be used. The limitation on the number of viable function calls was the compromise for restricting information for the genetic algorithm. In this instance, polishing, or finishing the genetic algorithm with a call of L-BFGS-B, was set to False. All other defaults were left, other than the maximum number of iterations and the population size. The number of iterations and population size were not optimized for the best performance of the genetic algorithm. For SLSQP, in order to convince the optimizer that the initial guess was not a convergent state, the step size used for the numerical approximation of the jacobian was changed from 10-8 to 10-2. Everything else was left as default.

In general, it is apparent that refining a relatively poor initial guess using the neural network optimizer can improve convergence and reduce the error, which was calculated at unseen data. A typical optimization pathway is demonstrated below in Figure 3.8, which shows the error between error of the optimizer and the target data as a function of the number of function evaluations. The method used below was Nelder-Mead, an algorithm which is not extremely efficient in terms of the average number of steps needed for convergence, but it is relatively robust to local minima and generally produces the best results with this model, as seen in Table 3.2. It is apparent from Figure 3.8 below that improving an initially relatively

Table 3.3: Test set loss and mean relative error of descaled parameters.

Method	Mean Relative Error of Parameters (%)	Test Set Error
Initial	2958	
200k NN	56	.0324
50k NN	58	.0391
20k NN	79	.0393
5k NN	143	.0634
2k NN	343	.0707

poor guess with the neural network transports the optimizer across a highly nonconvex space, allowing it to converge to a much more accurate local minimum in fewer function calls than without using the neural network optimizer. Interestingly, there are several points where the unrefined error is lower than the refined error, which hints at the relative number of local minima in the optimization space. Considering Figure 3.5b, it is apparent that given the number of different ways the parameters can change the shape of the discharge curve, it is not surprising that there are multiple avenues for the reduction of error, even if the result is not the globally optimal result.

While the average case is improved, it is important to note that the cost of having a small neural network which is also generalized implies an accuracy tradeoff if the initial guess is very good, the neural network may suggest changes which make the guess worse, in which case, the output can be ignored and the total opportunity cost of attempting to improve an initial guess with the neural network was only a single extra function call. Additionally, while all of the training data is physically meaningful, there is no guarantee that the per-parameter corrections given by the neural network optimizer will result in physically meaningful parameters. Although the training data bounds are sensible, there is no guarantee that an output from the neural network will fall within these bounds, and no meaningful way

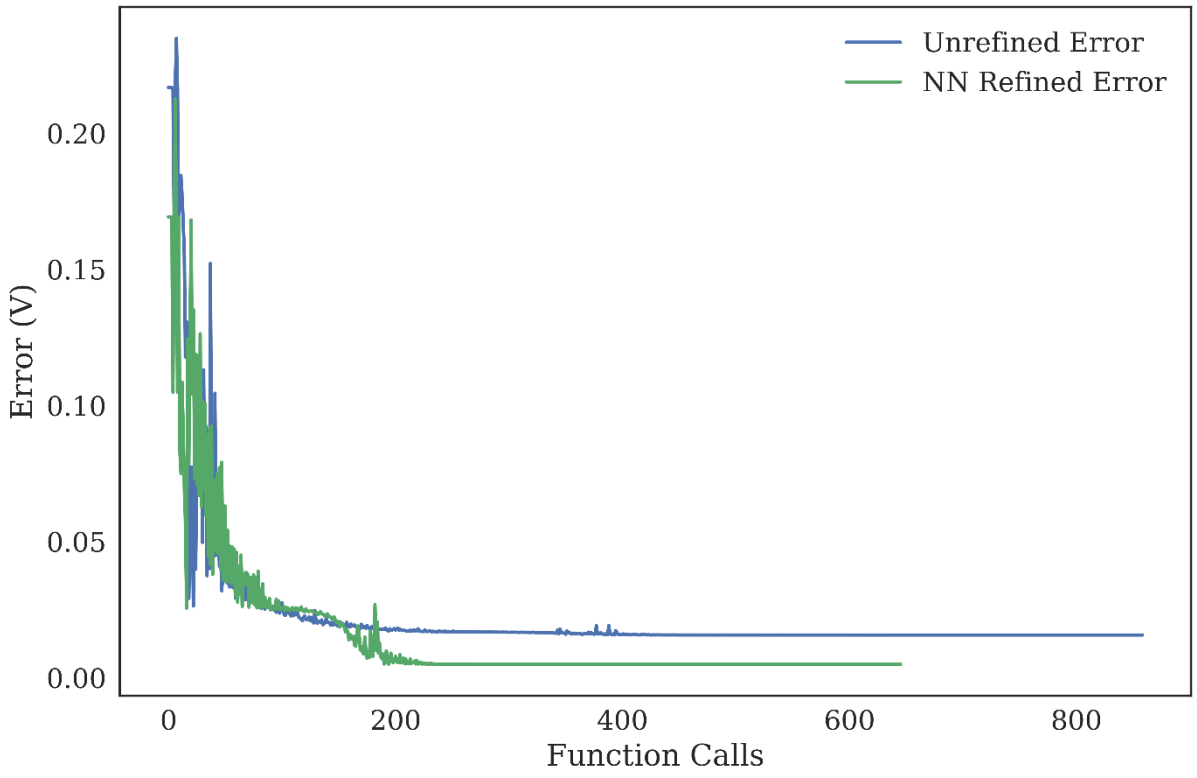


Figure 3.8: Typical optimization pathway for Nelder-Mead, where the current estimate error is displayed vs the function call number. Starting from the same initial guess with an error of 220mV, the guess refined by the neural network converges to a significantly lower error than the unchanged guess.

to predict when it will happen. Aside from changing the initial guess and estimating with the neural network again, there is no way to quickly repair an a-physical estimate. For example, the neural networks trained on smaller data sets have recommended negative diffusivities. In this instance, the guess was thrown out.

Lookup table performance is examined in the next section, which directly compares the neural networks with genetic algorithms in terms of using best-of-set sampling for generating initial guesses. The neural network size selected was chosen to compromise between simplicity and accuracy a smaller network will prefer generalizability to overfitting, but with an

extremely large data set, better test error may be achieved despite the tendency to over-fit. There are additional considerations as well, such as size on disk and time to compute. A look-up table is O(n), and as such takes around 2.5 seconds for the training set of 150,000 simulations, stored in a Numpy array, loaded into memory. When stored in a Hierarchical Data Format version 5 (HDF5) array using H5Py[88], the array does not need to be pulled into memory, but can be loaded one chunk at a time. While this is extremely beneficial for large datasets, it is also considerably slower, taking 10 minutes to look through the table once. Additionally, the compressed size of the data on disk is 1.04 GB. Time to evaluate the neural network is roughly 1e-6 seconds, and the size on disk is 1.2 MB significantly faster and smaller than keeping the compressed data. While modern hardware can handle these larger files easily, if model estimation is attempted on cheaper or more limited hardware, these may not be acceptable.

While this work focuses on using this formulation to fit discharge curves of simulated lithium ion batteries, any multi-objective optimization which is compressed to a single objective for an optimizer could stand to benefit from framing a neural network as a single-step optimizer rather than compress the multiple objectives into a single value and pass that to a traditional black box optimizer. This technique does not eliminate the value of traditional optimizers, however; the best result comes from using the neural network to refine a poor guess, as is often the case when starting an optimization problem. The neural network can never achieve arbitrary precision the same way a traditional optimization algorithm can. The goal of the neural network optimizer is to traverse the highly non-convex space between an initial guess and the target parameters, and give an easier optimization problem to the traditional, theoretically proven optimization methods.

3.6 9D Application - Genetic Algorithm Comparison

While the above method is useful for analyzing the performance for navigating in a 9-dimensional parameter space and trying to get from some point A to some point B, those who wish to arrive at the best result for an optimization problem often do not place much

value on the initial guess. In these instances, other methods may first be used to explore the parameter space, and the best result from that will be used as an initial guess for the new optimization problem using a more traditional optimizer.

There is an analog for this deep neural network optimization method as well, where after the neural network is trained, instead of starting from some unseen data point, a random sampling of training or test points are fed in as initial guesses, where the target curves remain the original targets. This has the added advantage of not requiring simulations to get a parameter guess, only to check the error of the guess. This technique leverages the idea that the neural network is only hyper-locally accurate, and that an accurate guess at the parameters cannot be guaranteed from any point in the 9-dimensional space. However, by sampling several points, it is possible to end up with an extremely performant guess for the cost of a few simulations.

To understand how this method is in direct competition with a genetic algorithm, that approach must first be explained. For each generation, a series of randomly generated points are created and evaluated. At the end of each generation, the best few results are kept and either randomly permuted, or combined with other results in order to create the next generation, which is then evaluated. This technique is popular in the literature, but it tends to be very function-call inefficient, converging only after many evaluations.

For this optimization, the formulation is equivalent sets of input parameters are either randomly generated or are randomly selected from a list of pre-generated, unseen parameters. These are then offered to the neural network, along with the associated error between the discharge curves, and a refined guess is generated by the neural network. The value of this guess is very difficult to determine without examining the result, and can be significantly better or significantly worse depending on many factors, including the sampling density of the parameter space, the size of the parameter space, and the initialized weights of the neural network. Although it may take several function calls to end up with a good guess, this guess is often very accurate, and for very sparse sampling it can compete with the lookup table performance.

Table 3.4: Average root mean squared errors of the best-of-set sampling results from each initial guess generation technique, fitting simulations at 0.5C and 1C. The error reported here is the sum of RMSE of voltage at 0.5C and 1C.

NN training size	NN test error	From initial (V)	10 samples (V)	20 samples (V)	50 samples (V)	100 samples (V)	Lookup Table (V)
200k	0.0412	0.0895	0.0482	0.0285	0.0150	0.0106	0.0072
50k	0.0419	0.0960	0.0704	0.0460	0.0305	0.0226	0.0083
20k	0.0423	0.0897	0.0408	0.0285	0.0190	0.0132	0.0128
5k	0.0526	0.1129	0.0723	0.0500	0.0260	0.0193	0.0205
2k	0.0644	0.1275	0.0918	0.0646	0.0371	0.0283	0.0292
500	0.0839	0.1960	0.1605	0.1108	0.0646	0.0494	0.0485
		18 samples	81 samples	144 samples	990 samples	2079 samples	
GA		0.2044	0.0784	0.0568	0.0275	0.0256	

Using the same data as the previous section, a new static set of 100 final target values was created by randomly sampling from the test data. There are two main differences between this analysis and the previous analysis rather than interpolating the currents, a new set of neural networks were trained on discharges at 0.5C and 1C, and the reported RMSE values are the summed result of calculation at these two currents. From the results in Table 3.4, it is clear that the neural networks perform significantly better than the genetic algorithm per function call, and that 100 random, unseen samples is sufficient to approximate the lookup table performance after passing the guesses through the neural network. While the lookup table seems to outperform the neural network, the converged results examined in Table 3.5 reinforce the idea that the root mean squared error metric does not tell the whole story, and the neural networks outperform the lookup table at low sampling densities.

In Figure 3.9 below, the resulting RMSE as a function of number of random guesses is examined for each of the neural networks and compared with a genetic algorithm. The neural network errors were sampled at 1, 10, 20, 50, and 100 random inputs, and the genetic algorithm was sampled as closely to 20 and 100 samples as math would allow, as the number of function calls scales as $\mathrm{O}(len(x) * (maxiter) * popsize)$. Since the length of x is 9 in this instance, it was not possible to perfectly hit the desired sampling values.

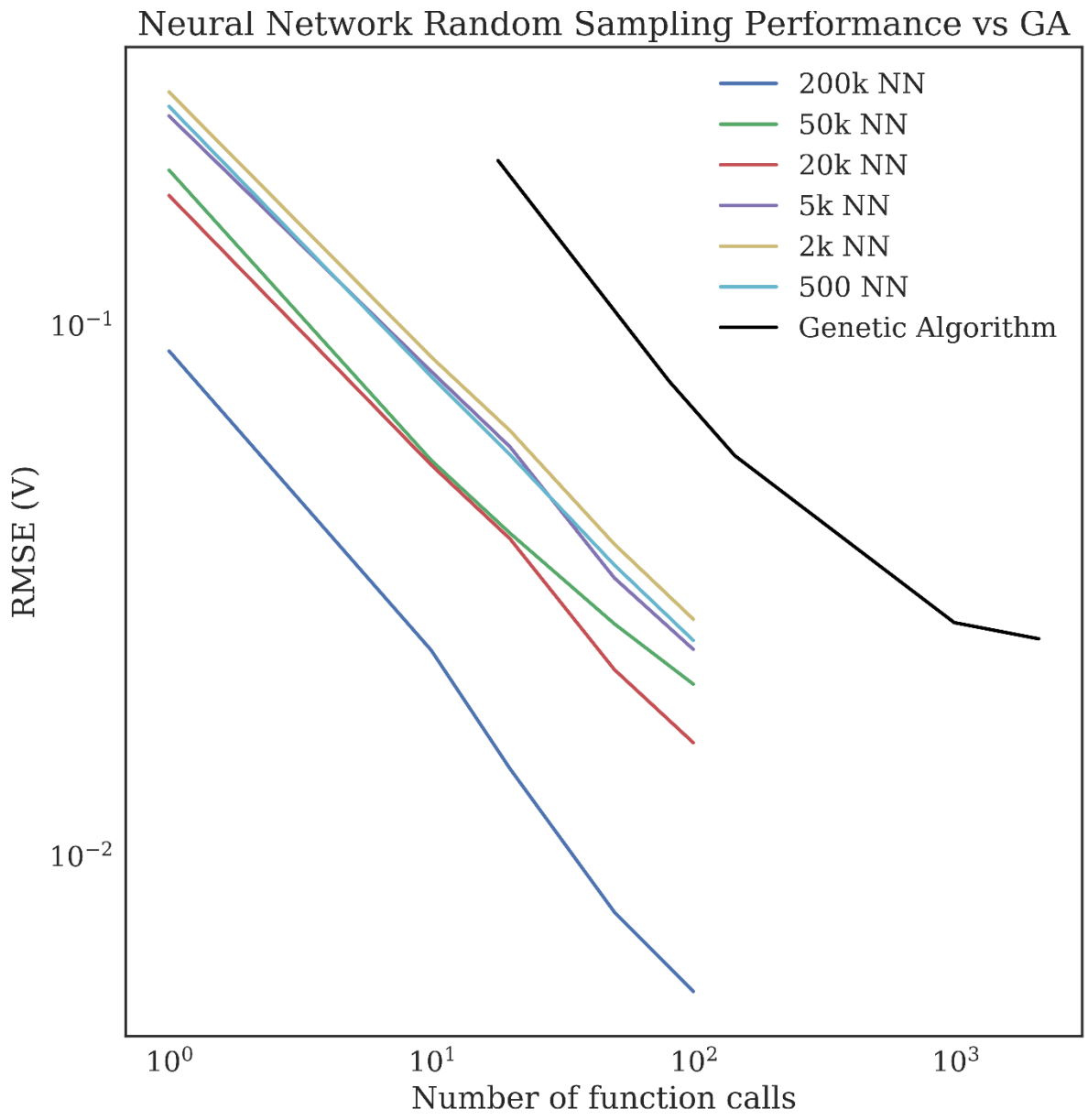


Figure 3.9: The root mean squared error between the best estimate from the neural networks and a genetic algorithm as a number of function calls. The neural networks are all similar, with errors decreasing with increasing training size. The genetic algorithm shares a qualitatively similar relationship, but the error begins to plateau by 1000 function calls, while the error is still higher than that of the neural networks.

Looking at the results in Table 3.4, some clear patterns emerge. The error of the guesses from the neural network is extremely high for the more coarsely sampled training sets, resulting in very poor error for the first function call, all of which were taken from the same initial guesses. After this initial function call, however, the guess points were randomly sampled from unseen data and the accuracy of the improved points was examined. The best results from each sample range are kept.

The reason this method is superior is that the neural networks are fairly small for this size of input data, where the input dimension is 1009, but the largest internal node size is only 95. This was done in order to force the neural network to generalize more aggressively, which can often improve the performance on unseen data when compared to a larger network which can memorize the dataset, but tends to over fit. This is classically known as the bias-variance tradeoff[89]. The size of the neural network was kept constant across sampling rates, in order to increase interpretability, meaning the network size is likely too large for the very coarse sampling and too small for the very fine sampling, and changing the neural network hidden layer dimensions to suit the sampling would result in better performance. It is worth mentioning that genetic algorithms are iterative in nature, meaning that the population generation and evaluation is embarrassingly parallelizable, but the number of iterations is inherently serial. For the neural networks, no action is serial, so the entire sampling is embarrassingly parallel, which can drastically decrease the time to an accurate estimate for long function calls.

After these initial guesses were collected, Nelder-Mead was used to polish the optimization result and the number of function evaluations needed to accomplish this task were recorded, along with the average final value. The parameter-level tolerance for convergence was set to 1e-6, and the maximum number of iterations was set sufficiently high to allow convergence. These results are compared to the output values from the genetic algorithm and are compared in Table 3.5. It is important to note that the initial errors between the static starting points and ending points was 0.9004V, and Nelder-Mead converged at an average RMSE of 0.0327 V when starting the optimization from that point. By refining the initial guess with a neural

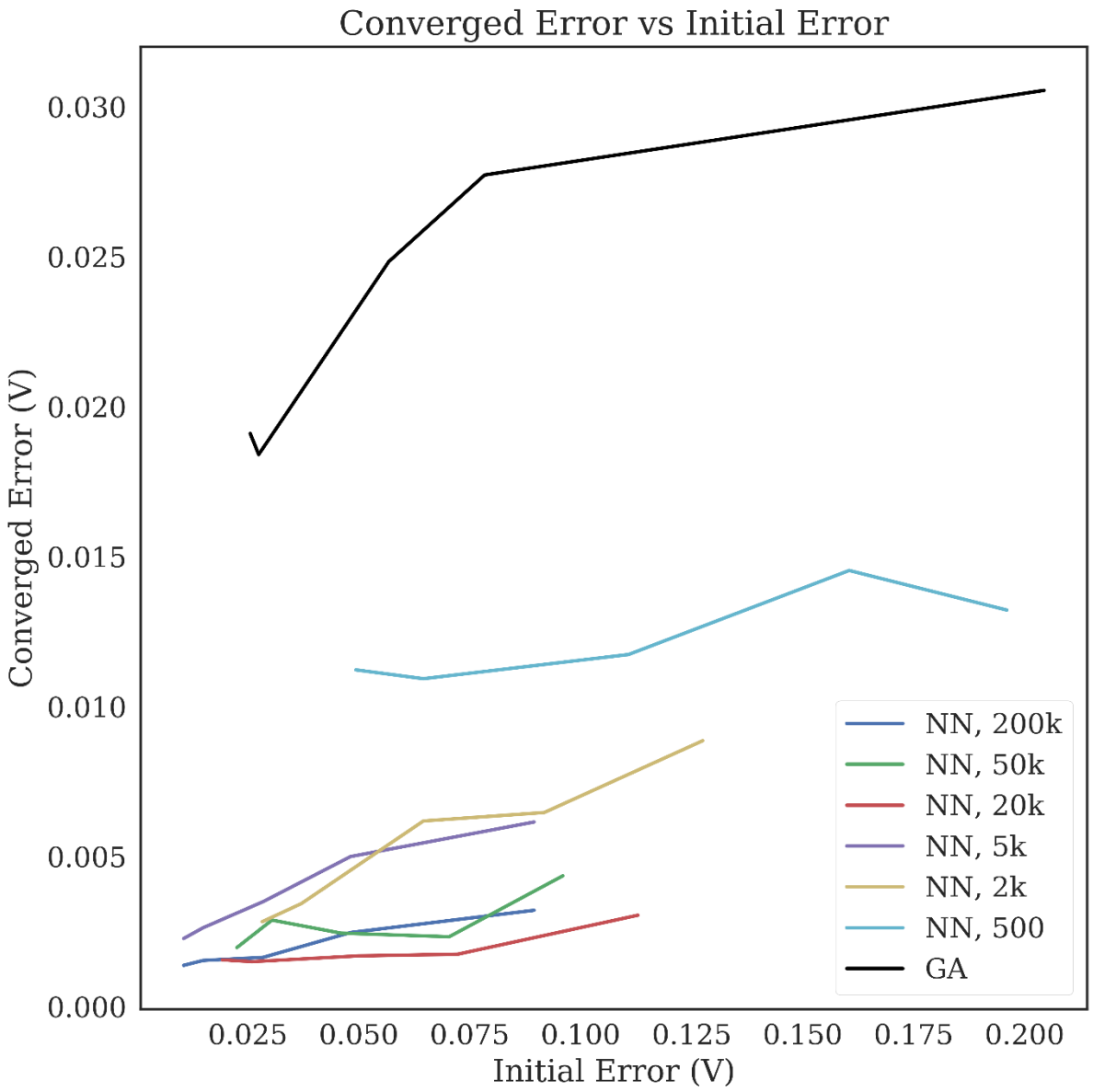


Figure 3.10: The converged error vs the initial error, grouped by method of guess generation. The neural networks clearly perform significantly better than the genetic algorithm, despite having many points which have comparable initial errors to the genetic algorithm.

network, it was possible to drastically reduce this error to 0.0032 V, representing a 10-fold reduction in error. However, by forgoing this constraint entirely, it was possible to get an

even lower 0.00137 V error after getting a very close initial guess by randomly sampling the neural network.

This performance offers a significant improvement over the converged error after polishing the output of the genetic algorithm, which had errors which were comparable in value to the neural network sampling, but which resulted in significantly worse convergent error. For example, by generating an initial guess using a genetic algorithm over the same bounds used for the generation of the training data, limiting the number of function calls to 990 spread across a population size of 11 and 9 iterations, an initial error of 8.3mV can be achieved. A comparable initial guess error can be found either using a neural network trained on 20,000 simulations and sampled 50 times, or a neural network trained on 5,000 simulations and sampled 100 times. However, after optimization, the initial guess from the genetic algorithm averages to 2.6 mV error, while the initial guesses provided by the neural network converge to 0.48 mV and 0.6 mV error, respectively a 5-fold improvement over the existing technique, despite comparable initial errors.

These results are summarized in Figure 3.10, which serves to showcase the idea that RMSE is not a sufficient metric when considering the difficulty an optimization algorithm will face when attempting to minimize the error from a given set of initial conditions. If a vertical line were drawn on Figure 3.10, it would indicate an equal initial error. Plotting intuition on this graph would likely look like a line with a 45-degree slope, indicating that any guess with a lower initial error would result in a converged result with lower error. Instead, it is clear that guesses generated by the neural networks and those generated by the genetic algorithm perform significantly differently, even when the value of the initial error is identical. This suggests that there is something which the neural networks are doing extremely well which the genetic algorithm is doing poorly, and that the guesses generated by the neural networks result in much easier optimization problems than those generated by the genetic algorithm. Even in the data-starved or data-comparable conditions of 500 and 2000 samples for training, the neural network outperforms the best-performing genetic algorithm.

Although it is popular to use a genetic algorithm to get an initial guess followed by a

Table 3.5: Final converged root mean squared errors as a function of number of samples, neural network training data size, and genetic algorithm function evaluations. The error is the sum of RMSE for 1C and 0.5C discharges.

NN Training Size	From Initial		10 Samples		20 Samples		50 Samples		100 Samples		Lookup Table	
	Num Calls	Final Error (V)	Num Calls	Final Error (V)	Num Calls	Final Error (V)	Num Calls	Final Error (V)	Num Calls	Final Error (V)	Num Calls	Final Error (V)
200k	705	0.003205521	674	0.002464	662	0.001637	659	0.001533	654	0.00137	1752	0.001129
50k	1137	0.004354459	1745	0.002323	1710	0.002444	640	0.002879	643	0.001964	663	0.001116
20k	1822	0.003040758	685	0.001741	695	0.001687	652	0.001493	646	0.001554	715	0.002721
5k	736	0.006148043	1777	0.004997	3902	0.003494	682	0.002625	671	0.002265	2863	0.004176
2k	1841	0.008858207	765	0.006464	758	0.006183	2214	0.003428	2218	0.002825	756	0.007579
500	787	0.013215034	1873	0.014534	784	0.011733	765	0.010926	772	0.011221	797	0.01185
GA	18 evaluations		81 evaluations		144 evaluations		990 evaluations		2079 evaluations			
	Num Calls	Final Error (V)	Num Calls	Final Error (V)	Num Calls	Final Error (V)	Num Calls	Final Error (V)	Num Calls	Final Error (V)		
	2867	0.03054	1790	0.02772	713	0.02483	2792	0.0184	657	0.0191		

traditional optimization technique, genetic algorithms are extremely inefficient in terms of performance per function call. The neural networks trained on Sobol-sampled data outperform the genetic algorithm, even with small amounts of data the smallest neural network trained on only 500 sets of generated data and sampled 100 times matches the genetic algorithm performance for 600 total function calls, compared to 990 for the genetic algorithm. This means that the neural network will outperform the genetic algorithm, even when starting from scratch. In addition, neural networks have the added benefits of being entirely embarrassingly parallelizable and are inherently reusable for new optimization problems. The ease of deployment and speed of neural networks make them a convenient way to compress the information from a large number of generated data points into a compact and easily accessible size, trading off some linear algebra for gigabytes of compressed data.

This discrepancy between the success of the optimization algorithm and the initial errors of the guesses is extremely interesting and serves to both demonstrate the inadequacy of a single value to represent the deviation between two discharge curves and to emphasize the importance of building parameter sensitivity into an initial guess generation method. For the current implementation of the genetic algorithm, no parameter scaling was done, which likely led to oversampling of large values in the range of diffusivities, which vary by three orders of magnitude. However, the lookup table results also feature a large error on the

Table 3.6: The RMSE of the previous converged results at the unseen condition of a 2C discharge. This represents an extrapolation in terms of current values, as 2C is higher than the 0.5C and 1C used for curve fitting. The NN and lookup table significantly outperform the GA, and the neural network outperforms the lookup table, especially when sampling is coarse.

NN Training size	Initial (V)	converged (V)	GA	initial (V)	converged (V)	Lookup Table	initial (V)	converged (V)
200k	0.01204	0.003065	**2079**	0.04358	0.03711	**200k**	0.0111	0.003117
50k	0.01841	0.003403	**990**	0.0438	0.03373	**50k**	0.01102	0.003569
20k	0.01462	0.003453	**144**	0.05362	0.03681	**20k**	0.01594	0.00493
5k	0.01963	0.004788	**81**	0.06227	0.03894	**5k**	0.02299	0.00648
2k	0.02536	0.0048644	**18**	0.1084	0.03911	**2k**	0.02916	0.01083
500	0.04137	0.01804				**500**	0.04202	0.01611

diffusivity-related parameters, which serve to demonstrate the extremely low sensitivity to these parameters when lithium diffusion is not the limiting factor that shapes the discharge curve.

An additional analysis was done using the lookup table results for the training sets of each of the neural networks. While randomly sampling 100 sets of parameter values was sufficient to approximate the error performance of the lookup table for each neural network, an interesting trend results from optimizing based on those recommended values. For coarser sampling, the RMSE of the converged values is improved by two fold compared to using the best fit from the training data, as shown in Table 3.6. Examining the error between the target parameters and the estimated parameters reveals that the neural networks are significantly better at estimating the parameters than using a lookup table, shown in Figure 3.11. A clear pattern emerges, wherein the optimization results based on the neural networks output outperform the training data until the sampling becomes so fine that the small neural network size tends to limit the accuracy of the model outputs, and the resulting errors well below one millivolt begin to converge. While this problem was done with 9 dimensions, the full dimensionality of the P2D model is 26, which would require significantly more samples to adequately explore.

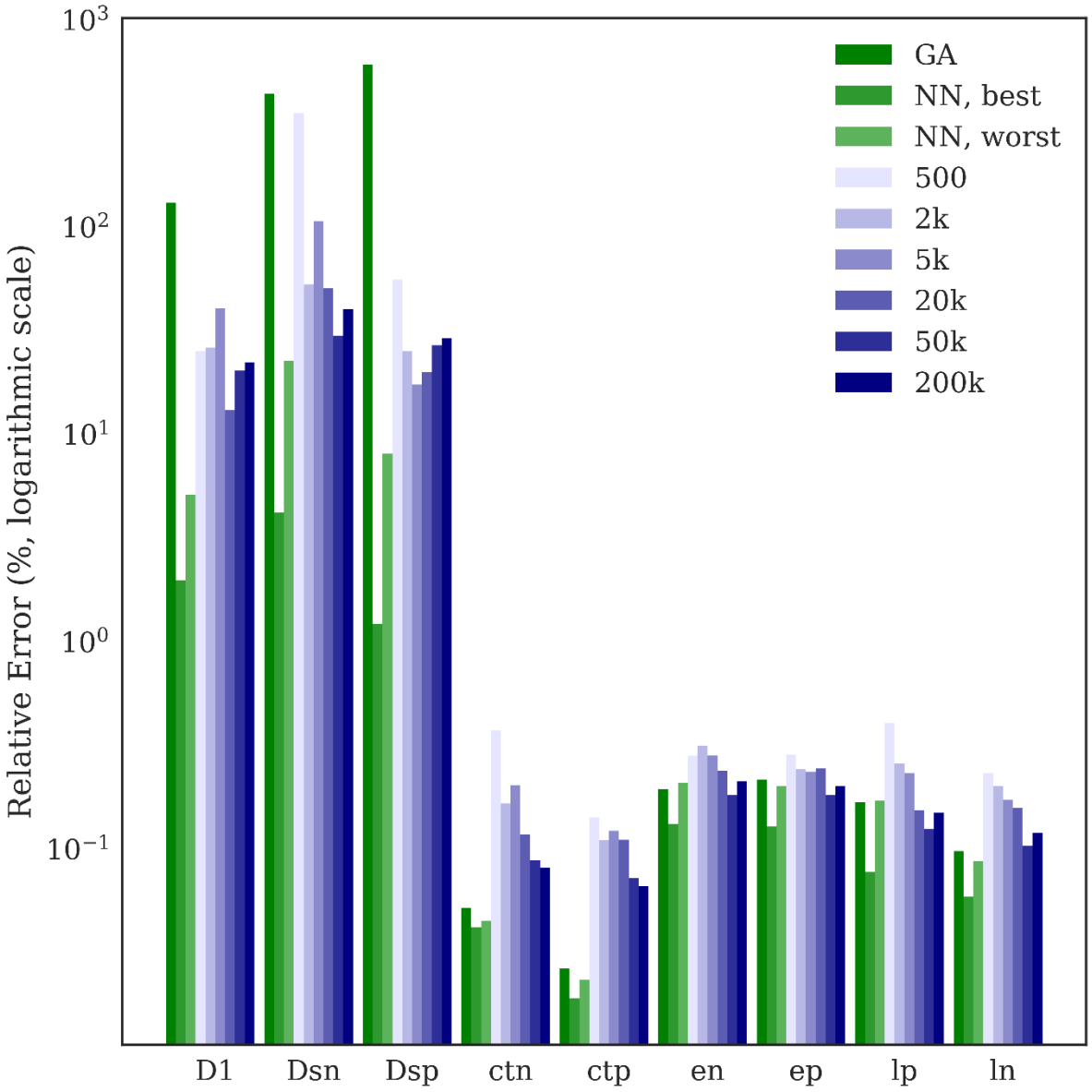


Figure 3.11: Relative per-parameter error for best GA, neural network, and training data. The neural network clearly offers the best performance, demonstrating significantly better performance than a lookup table of the training data.

It is clear that the RMSE between two curves does not fully predict the ease with which an optimizer will converge to an accurate solution. Two different problems have been ana-

lyzed, one of which was 2-dimensional for the purposes of visualizing the process, and the other of which was 9-dimensional for the purposes of exploring a practical set of optimization problems in higher dimensions. Potential future work would include extending this analysis to higher dimensions, using battery simulations with differing sensitivities to the parameters, or perhaps combining these approaches with Electrochemical Impedance Spectroscopy measurements for increased sensitivity to other parameters. Additionally, the same underlying assumptions used when generating the data are applied to the applicability of the model. For example, to assume that the parameter space is uniformly varying across a parameter range is likely false, in particular with electrode thickness it is likely closer to a bimodal distribution, as some cells are optimized for energy and others for power. Sampling the types of batteries to be fit and using the distributions of parameters can increase the chances of success when calibrating the model to experimental batteries.

3.7 Conclusion

In this work, a deep neural network was used both to refine a poor initial guess and to provide a new initial guess from random sampling in the parameter space. For the execution cost of one additional function call, it is reasonable to improve the final converged error on unseen data by 100-fold when compared with random model parameterization, often with fewer total function calls. It should be noted that this performance is on an exceptionally poor guess, indicating the difficulty optimizers have with this model. However, by randomly generating data and feeding these points into the neural network, it is possible to get an extremely good fit for under 100 function calls, which improves final converged error by 5-fold compared to generating an initial guess using a genetic algorithm, and improves the error by 10-fold when evaluating model performance at higher currents. This framework is generally applicable to any optimization problem, but it is much more reliable when the output of the function to be optimized is a time series rather than a single number. Additionally, the outputs of the neural network after 100 function calls outperform the lookup table of the training data, indicating value added by the ability to interpolate between data points

in high dimensional space, while taking significantly less space on disk than the training data. The deep neural network can easily be implemented in any language which has matrix mathematics, as was done with Numpy in Python. In this instance, the neural network acts as a much more efficient encoding of the data, replacing a lookup table with a few hundred element-wise operations and replacing a gigabyte of data with a megabyte of weights and biases.

The optimization formulation of the neural network leverages the increased informational content of a difference between time series over an abstracted summary value, like root mean squared error, which is required by many single-objective optimizers. Additionally, the ability of neural networks to take advantage of data shuffling techniques allows the algorithm to efficiently combat overfitting for only minimal computational overhead during training. This formulation also allows the neural network to extract the maximum amount of information from the generated data when compared to the inverse formulation, in which each discharge curve and input parameter pair is only seen once. The source code for this work can be found at https://github.com/nealde/EChemFromChess, along with examples and code for generating some of the images.

Acknowledgements

The authors would like to thank the Clean Energy Institute (CEI) at the University of Washington (UW) and the Data Intensive Research Enabling Clean Technologies (DIRECT), a UW CEI program funded by the National Science Foundation (NSF), in addition to the Washington Research Foundation (WRF) for their monetary support of this work. The battery modeling work has been supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Vehicle Technologies of the U. S. Department of Energy through the Advanced Battery Material Research (BMR) Program (Battery500 Consortium).

Chapter 4

CREATING PHYSICALLY CONSISTENT ARTIFICIAL NEURAL NETWORK SURROGATE MODELS FOR THE SINGLE PARTICLE LITHIUM-ION BATTERY MODEL

4.1 Introduction

Lithium-ion batteries are nearly ubiquitous, ranging from cell phone to grid scale applications. These complex electrochemical devices are valued for the high energy and power densities afforded by the facile kinetics and large magnitude electrochemical potential of lithium. In order to accurately describe the behavior of lithium-ion batteries, a wide range of models are available, ranging from simple empirical equivalent circuit models up to molecular dynamics simulations. In between these two extremes are continuum-scale models, of which the single particle model (SPM)[1,5] and pseudo two-dimensional (P2D)[2,3,4,5,6,7] models are two of the most popular, selected for their compromise between accuracy and solution speed. The SPM tracks lithium concentrations inside spherical particles in the positive and negative regions, but does not include any electrolyte effects, and so is only applicable at low currents. The P2D model includes electrolyte effects as well as concentrations which are a function of depth in the electrodes, greatly increasing the region of accuracy, and remains valid at high currents, with some models including temperature effects.

While these models are useful in the design and control of lithium-ion batteries, they must be calibrated to a particular battery in order to be used for control[8]. This process of model calibration is extremely time consuming, and the models are too expensive to make global black-box packages viable solutions in the high dimensional space, which can vary between 12 and 26 input parameters, depending upon the model[17].

Data science, hailed as the fourth paradigm of science[11], is a large field that includes

methods of data analysis, data storage, and data visualization. Machine learning, a powerful data analysis tool, provides methods for creating universal approximation functions which seek to map given inputs to given outputs. Artificial neural networks, and a popular form known as deep neural networks (DNNs), are extremely powerful learning machines which can approximate very complex nonlinear functions. However, during training, the outputs of the neural network are calculated independently, meaning that enforcing physical consistencies can become difficult when predicting values which are coupled. As the neural network is forced to extrapolate, the signal:noise ratio falls out of favor and a-physical predictions can arise, including violations of monotonicity of curves, which are relevant for constant-current operation of batteries.

In order to combat this, it is necessary to couple the outputs of the DNN. This is done by having the DNN estimate the inputs to another function which can help with this physical consistency in this case, a hyperbolic tangent function. The details of how and why this method was selected are examined in the next section. An example application is then presented on the SPM in the rest of this work.

4.2 Physical Consistency with Neural Networks

Deep neural networks (DNNs) are extremely popular for surrogate model creation of more expensive computational models, in particular in fields like quantum chemistry and molecular dynamics[90,91,92,93,94,95], where the models are extremely costly to run. When using DNNs on physical systems, or systems which should ensure some level of self-consistency, care must be taken both in problem formulation and in model penalization in order to ensure that this physical consistency is maintained.

Earlier works[24] have demonstrated that even when the application of DNNs to predictions on a physical system result in higher accuracy, they can also result in physical inconsistency. Namely, when applied in the task of lake water temperature estimation as a function of depth, situations can arise where temperatures are such that lower density pockets of water are estimated to lie below higher density pockets of water, which is physically inconsistent.

By explicitly calculating the density associated with the temperature estimate and punishing the DNN for arriving at physically inconsistent predictions, this can be mitigated to a large extent, even improving the accuracy of the model. It should be mentioned that this is often not the case in general, punishing physical inconsistencies will mean an increased overall mean squared error.

The same concepts have been applied to a simple Butler-Volmer Nickel Electrode model, described in equations 4.1 through 4.4. In this model, F, R, and T are Faraday's constant, the universal gas constant, and temperature, in Kelvin, respectively. Φ_1 and Φ_2 are the equilibrium potentials of the cathode and anode. W, V, and ρ represent the mass of active material, volume, and density, respectively. The exchange current densities are represented by i_1 and i_2 for the cathode and anode. i_{app} represents the total applied current density, in $\frac{A}{m^2}$.

$$J_1 = 2i_1[(1-y_1)e^{\frac{0.5F}{RT}y_2-\Phi_1} - y_1 e^{\frac{-0.5F}{RT}y_2-\Phi_1}] \tag{4.1}$$

$$J_2 = i_2[e^{\frac{F}{RT}y_2-\Phi_2} - e^{\frac{-F}{RT}y_2-\Phi_2}] \tag{4.2}$$

$$\frac{dy_1}{dt} = \frac{J_1}{F}\frac{W}{\rho V} \tag{4.3}$$

$$J_1 + J_2 + i_{app} = 0 \tag{4.4}$$

On this simpler model, a DNN is tasked with predicting a time-series of voltages rather than a length-series of water temperatures. In this instance, voltage should be monotonically increasing, and any instance of lower voltage occurring after a higher voltage could be interpreted as a physically inconsistent prediction. The model input is the applied current density i_{app}, which changes the time of the voltage response. By changing this input, it is possible to generate a simple 1-dimensional data science problem, where a DNN will attempt to recreate the physical model output, shown below. In order to accomplish this, the parameter must be normalized to be on the range (0,1), as the DNN weights and biases are

initially parameterized in anticipation of these magnitude inputs. In general, it is a good practice to also normalize the model outputs to be on the order of 1, although the voltages in this instance are already close enough to a value of 1 that this was not done, for simplicity. While successful results may be accomplished without parameter scaling, isolating the shape of the desired output from the magnitude generally results in better performance[96]. The parameter space was randomly sampled 100 times, and the neural networks were trained on 80 samples and tested on the remaining 20. For all neural networks, the exponential linear unit was used as the activation function of choice, and the Adam optimizer was used for weight and bias fitting.

In order to evaluate the effects of different problem formulations, the DNN attributes were changed in accordance with Table 4.1 below. The effect of changing the dimensionality of the neural network namely modified the bias-variance tradeoff, where smaller neural networks are forced to generalize more, which leads to worse training accuracy, but more maintained test accuracy, while larger neural networks will tend to overfit the training data if trained exhaustively. The traditional mean squared error (MSE) loss function penalizes incorrect outputs from the model in accordance with the L^2 norm, but does not enforce any relative constraints. The custom loss function includes the MSE, for accuracy, and also explicitly penalizes the network whenever a monotonistic constraint is violated - for instance, the voltage at each subsequent time point should be greater than or equal to the voltage at a previous time point.

The Recurrent network is recurrent only in problem formulation and not in model design – it is a traditional DNN, but in addition to the normalized applied current density, it also takes the voltage at any particular time as an input and estimates the voltage at a future time step. By iteratively calling this model, it is possible to rebuild a forecast of the curve. It is important that the entire forecast be shown, rather than the estimates at any single point in time – the first approach highlights any accuracy drift in the network, while the second would artificially inflate the performance of the model.

Table 4.1: The physical consistency is presented below, along with the framework used to create that estimate. Each of these methods represents a way of potentially combating physically inconsistent predictions.

Neural Network	Hidden Layers	Loss Function	Physical Inconsistency
Forwards NN	(75, 75, 75, 75)	MSE	35.6%
Physics NN	(75, 75, 75, 75)	Custom	24.4%
Physics NN Large	(250, 250, 250, 250)	Custom	10.1%
Recurrent DNN	(75, 75, 75, 75)	MSE	3.1%
Subsampled NN	(75, 75, 75, 75)	MSE	0.0%

$$MSE(ya, b) = \frac{1}{n}\sum_{i=1}^{n}(a_i - b_i)^2 \tag{4.5}$$

$$loss(a, b) = MSE(a, b) + max(a_i - a_{i+1}, 0) \tag{4.6}$$

The Subsampled network is trained on identical data to the other non-recurrent networks, but the time series is sampled at $1/10^{th}$ the typical rate, decreasing the output dimensionality. This larger gap between points greatly reduces the effect of noise on the monotonicity enforcement, which generally results in better physical consistency, but worse accuracy and a qualitatively much worse shape. Each of these formulations offers a method of trading off physical consistency for accuracy or complexity in problem formulation, and can act as potential remedies to systems which have physical consistency requirements.

4.3 Expanding to the Single Particle Model

Traditionally, a deep neural network (DNN)-based surrogate model could have one or more of several design goals. The surrogate model may seek to replicate some particular

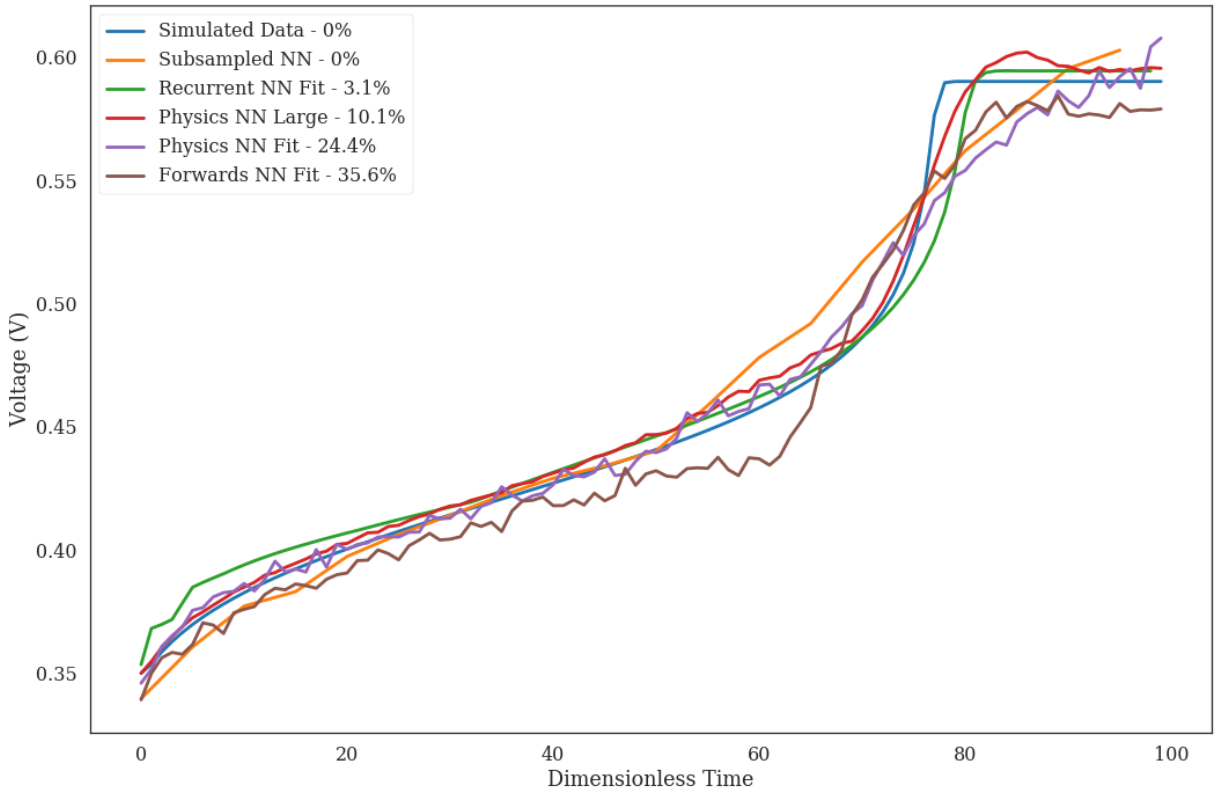


Figure 4.1: Physical consistency, or voltage monotonicity, of various approaches to an identical voltage prediction task for the simple Butler-Volmer Nickel Electrode model.

metric of the expensive computational model, but at reduced computational cost – for example, predicting time-series of voltages for the purpose of fitting to experimental data. The surrogate model may be created in order to restructure the informational content of the model – for instance, to map voltage time series onto model inputs for inverse optimization, or to create a recurrently formulated variant for the purposes of control at some time horizon[17]. In each of these cases, information which is necessary for the computational model, but not the approximate model – that is, the internal states of the physical system – can be discarded, as it is unnecessary for the task of voltage prediction.

While each of these potential goals may have a useful niche application, there is value in also tracking the internal states of the model for control purposes. As others have shown,

Table 4.2: Single Particle Model Equations

Ordinary Differential Equation	Boundary Condition
$\frac{\partial c_{s,i}}{\partial t} = \frac{D_{s,i}}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial c_{s,i}}{\partial r}\right), i = (p,n)$	$D_{s,i}\frac{\partial c_{s,i}}{\partial r}\|_{r=0} = 0$
	$D_{s,i}\frac{\partial c_{s,i}}{\partial r}\|_{r=R_i} = -J_i$
Algebraic Equations	
$J_i = k_i c_e^{0.5}(c_{s,i,max} - c_{s,i})^{0.5} c_{s,i}^{0.5} sinh(\frac{0.5F}{RT}(\phi_i - U_i(\frac{c_{s,i}}{c_{s,i,max}}))$	
$J_i = \frac{-i_{app}}{a_i F l_i}$	
$V_{cell} = \Phi_p - \Phi_n$	

modifying external control values to accommodate internal values can greatly extend the operational life of the battery[8,97]. Specifically, looking at the overpotential at the anode can provide enough information to double cycle life at high current rates by reducing the propensity of the lithium plating side reaction[9]. Models which only predict voltage forfeit this capability, and with it, a significant portion of the added value of the model.

In this work, the neural networks are formulated as recurrent problems, and attempt to map the current state not to the next values in the time series, but to the derivative of the states, as calculated by subtracting the differences in the state change. This allows the neural networks to act as ordinary differential equations, and the outputs can be scaled to allow for arbitrarily large or small time steps, explicitly trading computational expense for accuracy at runtime. The neural networks will be applied to the single particle model (SPM), whose equations and boundary conditions are listed in Table 4.2. The concentration inside the particle is tracked using a constant diffusivity term, $D_{s,i}$, using the radial symmetry boundary condition at the center and the change in concentration is equal to the molar flux at the surface of the particles. The values and ranges of the parameters are given in Table 4.3.

The most naive approach involves framing the entire model output in a recurrent for-

Table 4.3: Single Particle Model Parameters

Parameter	Lower Bound	Upper Bound	Units
D_p	1e-15	1e-13	$\frac{cm^2}{s}$
D_n	1e-15	1e-13	$\frac{cm^2}{s}$
$c_{s,p,max}$	30555	-	$\frac{mol}{m^3}$
$c_{s,n,max}$	51555	-	$\frac{mol}{m^3}$
l_p	1e-5	1e-4	m
l_n	1e-5	1e-4	m
a_p	8.1e5	9.3e5	$\frac{m^2}{m^3}$
a_n	8.1e5	9.3e5	$\frac{m^2}{m^3}$
R_p	8e-7	5e-6	m
R_n	8e-7	5e-6	m
k_p	8e-12	5e-11	$\frac{m^{2.5}}{mol^{0.5}s}$
k_n	8e-12	5e-11	$\frac{m^{2.5}}{mol^{0.5}s}$

mulation, feeding in all current states and returning either the derivative at that time step or the value at the next time step, depending on the intended use. This results in poor and self-inconsistent results, where a self-consistency analysis similar to that used by a time solver reveals very large deviations from the solve tolerance of 10^{-6} error.

Rather than using the most naive approach, another formulation which begins to break the problem into pieces can be found which is still ineffective. When forecasting the future states of the battery, either the derivative at the current time or the next state can be estimated. Choosing the latter results in significantly poorer results, as shown in Figure 4.2 below. This is due to the scale of the noise when direct estimation of the state at the next time point is done. By framing the problem as a derivative estimation, reducing the magnitude of iterative change at each time step, it is possible to significantly reduce the impact of this noise, although the stability at longer predictions may still be suspect.

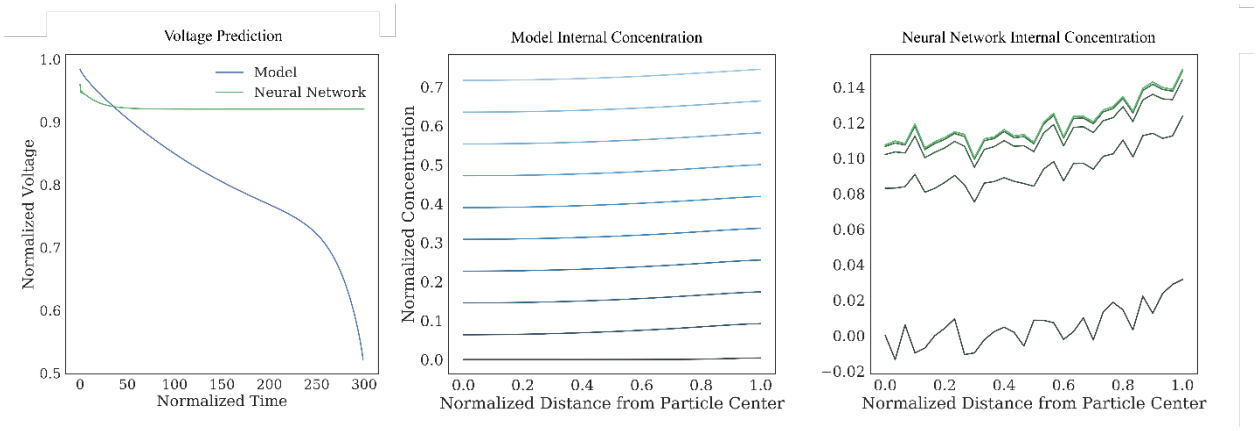


Figure 4.2: Looking at an unseen physical model parameter input combination, the neural network is forced to extrapolate from the training data. This, combined with the fact that the neural network has never been trained on its own physically-inconsistent predictions, causes the network to produce meaningless results.

Original attempts sought to map the recurrent inputs to these derivative directly, but these would generate small errors which would accrue to very large errors by the end of a simulated discharge. They are also subject to physical consistency considerations, and

the concentration inside a particle is used as an example. During discharge, lithium ions leave the spherical particles through the surface, and the concentration inside the particle changes subject to the constraints of the particle radius, the diffusivity inside the particle, and the particle concentration gradient. As such, during a constant current discharge, the concentration within the positive lithium particle should be monotonically decreasing. The large scale of the noise in Figure 4.2C represents a violation of this physically consistent condition.

These physical inconsistencies are caused by random noise inherent to neural network extrapolation from fitted values. The above formulation was time-constant, meaning that the model did not truly predict the derivative, but the value at the next time step. This changes the order of the magnitude of the signal and noise to immediately be on-par with the expected values, resulting in very short effective forecasts. By changing the problem formulation to estimate the derivative rather than the next value, the order of magnitude of these small errors can be greatly reduced, generally producing a better result. While the effective forecast time is significantly improved, the random noise buildup can still cause the neural network forecast to diverge from the correct prediction, as the neural network is forced to extrapolate more from the smooth training data. This formulation is examined in the next section.

In order to avoid the accrual of these small, physically inconsistent errors, it is necessary to wrap the neural network output in a function which can guarantee some level of relative consistency. To this end, a custom loss function is used which wraps the neural network predictions in a fully-described hyperbolic tangent function, which can accurately mimic the derivative of the concentration inside the particles at any time point. There are two ways this problem can be formulated: the data can be externally fit to effectively embed the informational content of the data through the lens of the hyperbolic tangent, using some black-box optimization algorithm, or the neural network can simultaneously fit the hyperbolic tangent function to the simulated data as well as train itself to become an accurate meta-estimator. The second formulation was chosen and is described below.

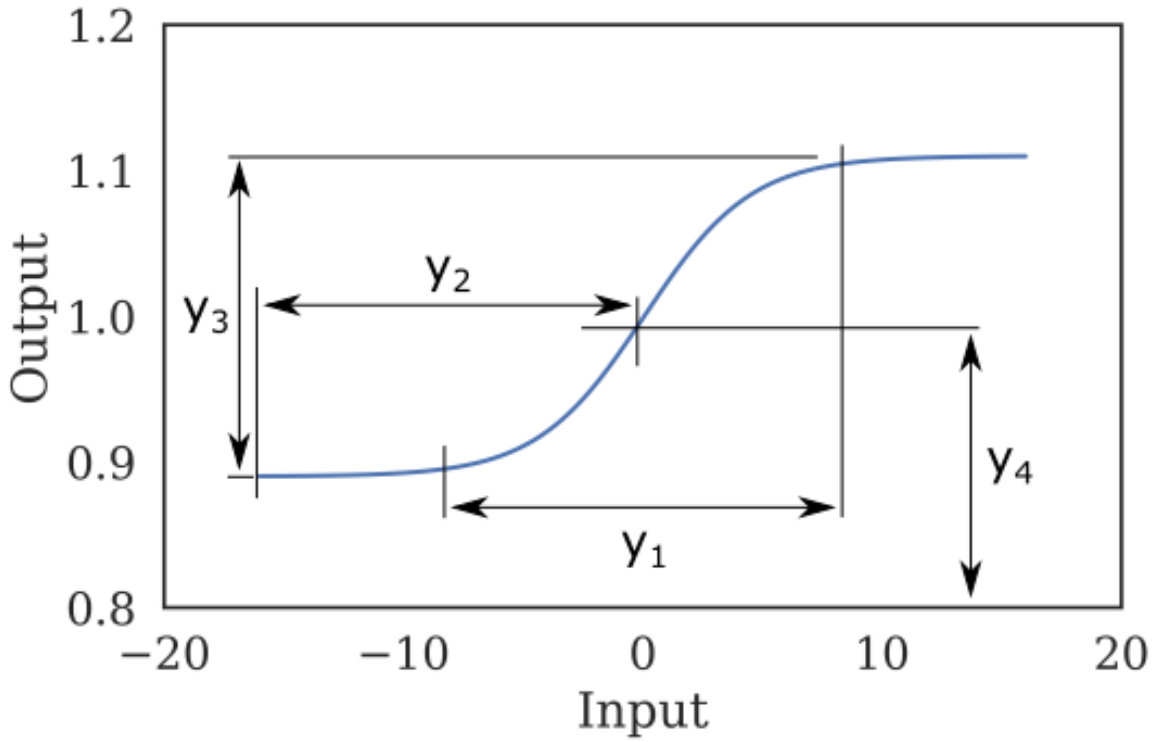


Figure 4.3: Here, a fully described hyperbolic tangent function is described, with the effects of each parameter shown. y_1 modifies the skew of the function, y_2 modifies the right-left placement of the function, y_3 modifies the scale of the function, and y_4 modifies the zero point of the function in the output direction. These levers allow the neural network to simultaneously fit the hyperbolic tangent function and act as a meta estimator, generating new values which are described by this shape.

$$y_{tanh} = tanh(e^{y_1} \cdot x + y_2 y_1) y_3 + y_4 \tag{4.7}$$

$$loss = MSE(y_{tanh}, y_{pred}) \tag{4.8}$$

Due to the dimensionality of the data, there are some practical implementation nuances which should be addressed. Using Keras[98], a front-end for Tensorflow[99] in Python, there are certain tests which evaluate the self-consistency of the models prior to training. One

of these constraints is that the given dimensionality at the last layer of the neural network must be equal to the dimensionality of the data which will be predicted – in this instance, that dimensionality is the number of nodes in the positive and negative electrodes of the Single Particle Model. In this work, 30 internal nodes were selected and are were evaluated to be self-consistent for the relevant discharge currents, giving 32 total nodes, including the central and surface concentrations.

Effectively, during each training epoch, the neural network will be predicting 32 values, which will never be directly compared to the target values. Instead, the first 4 will be applied as per equation 4.7 above, and the rest will be completely ignored. Later, during benchmarking, once the model weights and biases have been extracted from the neural networks, the final matrix dimensions can be reduced such that the total output dimensionality of the neural network is 4 when the model is evaluated in NumPy[100].

In this framework, there are a minimum of 4 neural networks which are called completely independently of each other. Two neural networks track the internal particle concentration, and two neural networks estimate the solid-phase potential at the positive and negative electrodes. At each time step, these neural networks are fed the current concentration inside the particles, scaled by $c_{s,i,max}$, and the input model parameters. These two neural networks are fit to the derivative at each time step, calculated from the model values using the forward difference formula. Using the surface concentration, the open circuit potential is explicitly calculated.

The surface concentration and model inputs are fed into the two remaining neural networks, which predict the difference between the open circuit potential and the solid-phase potential – in effect, they estimate the local overpotentials at the surface of the particles. By estimating these states, it is possible to rebuild the solid phase potential, Φ_i, at the positive and negative electrodes, and back out the exact values used to calculate V_{cell} in the model. Framing the problem in this manner allows the neural networks to excel at the task of estimating unknown states from known states, while outsourcing cheap calculations to the actual math that generated those values in the first place, leading to a hybrid formulation.

4.4 Applications with 2 Varied Parameters

Initially, when looking to create a team of neural networks which can accurately reproduce the internal states of the Single Particle Model, with the intent being to rebuild the external states directly from those, several different problem formulations were attempted. The constant time recurrent approach was the most obvious, but had several drawbacks – the noise created by physically inconsistent measurements would very quickly overwhelm the signal, and cause the neural network to fail, as shown in Figure 4.2 above. Next, the recurrent derivative model was explored, which effectively fit the neural network to the form of an Ordinary Differential Equation, where the derivative at the current time is predicted, and a simple explicit time integration method can be used. This has several advantages, including being able to explicitly trade computational time for accuracy, assuming the neural network has a high accuracy, in the form of taking larger time steps.

However, after looking at the form of the derivative for particle internal concentration, the shape was instantly recognizable, and is shown in Figure 4.4 below. Initially, it was apparent that the data could be fit with a hyperbolic tangent and then the fitting parameters for that could be estimated directly, but fitting these concentration gradients, which varied wildly both in shape and magnitude, was a very difficult endeavor and accurate results could not always be guaranteed. Other works[95] have used neural networks to directly solve boundary value problems (BVPs) and other similar problems, effectively using the neural network as a nonlinear solver. In other domains[101], it is popular to use a neural network as a means to solve a single problem rather than a generalized problem, such as controlling a human model to teach it to walk. The goal of this task is to maximize a score, or minimize a penalty, effectively using the neural network as a very robust optimization framework. Since these nonlinear fitting capabilities are built-in to the neural network framework, it should be possible to simultaneously have the neural network learn to act as an estimator and a fitting algorithm.

Qualitatively, the initial point, or the derivative at time 0, is extremely different from the

rest of the derivatives, which all seem to pivot around the same point and approach a flat line as the battery reaches a pseudo-steady state operation, where the concentration within the particles is changing at a constant rate due to the nature of a constant current discharge in combination with the simplicity of the model. Having a single neural network which excels at both the initial condition estimate as well as all other points in time is exceedingly difficult, as the initial time points are underrepresented, making up only 0.3% of all data points. This unbalanced data set makes it very unlikely that the errors at those points will carry enough weight to effectively influence the training direction of the neural network. In order to correct for this, the initial values were duplicated 30 times.

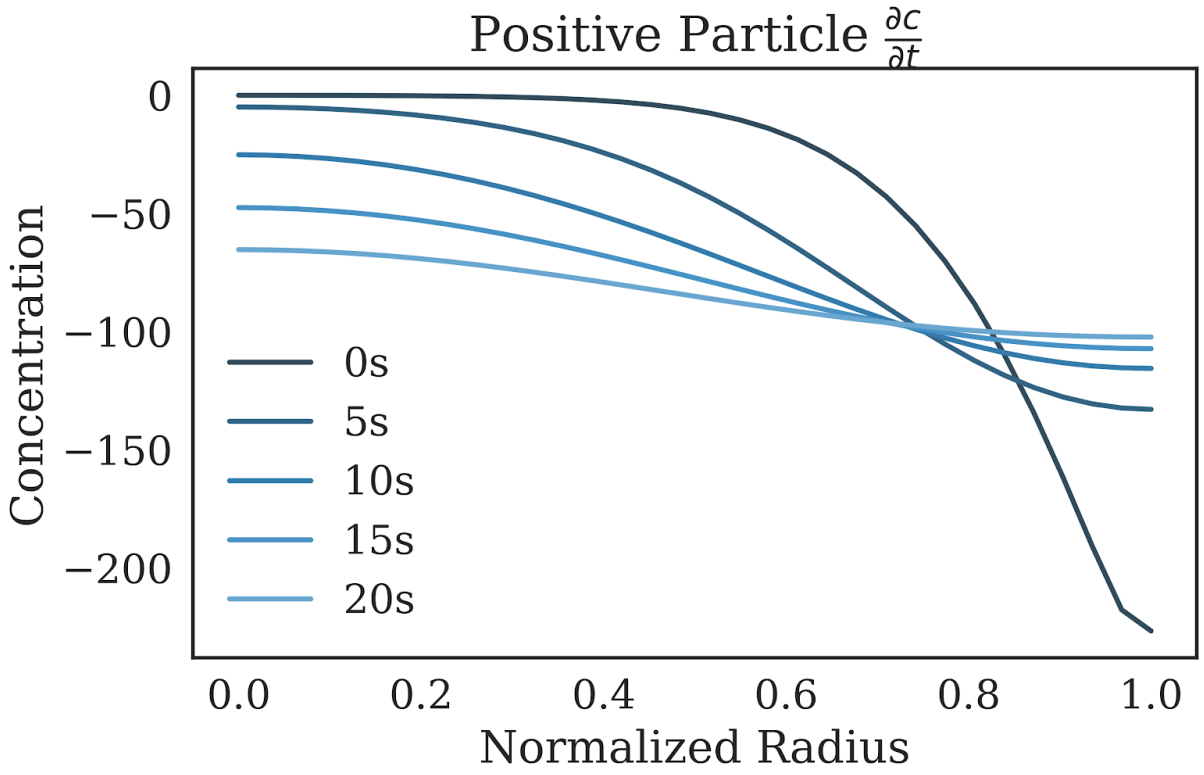


Figure 4.4: Positive particle derivative shape across the normalized radius of the particle, viewed at distinct time stamps. The shape of these curves are qualitatively similar to that of a hyperbolic tangent function, which is already built into machine learning libraries.

When examining the final internal positive particle concentrations, the effect on the stability of the estimation is apparent, as shown in Figure 4.5. While the values are relatively even, the model which was fit through a hyperbolic tangent function results in a significantly smoother concentration, which also creates more stable behavior, which comes with higher accuracy, especially during extrapolation outside the samples parameter values. The curves below were created by sampling the positive and negative electrode thicknesses using Sobol[77] sampling, with 50 examples for the training set and 50 examples for the test set. Figure 4.5 was generated using samples from the test set. As the sampling gets finer, the performance of both neural networks converge to perfectly replicating model outputs, but the tanh neural network is significantly more accurate and stable at this level of sampling. This will become important in the next section, when the sampling will remain coarse by necessity as the dimensionality increases and n^6 sampling is no longer feasible. In this analysis below, over-trained neural networks were used for the prediction of overpotentials and negative electrode concentrations, which were able to replicate the model values to within self-convergent values, and only the neural network tracking the positive concentration was modified, to better isolate the effects of this training style.

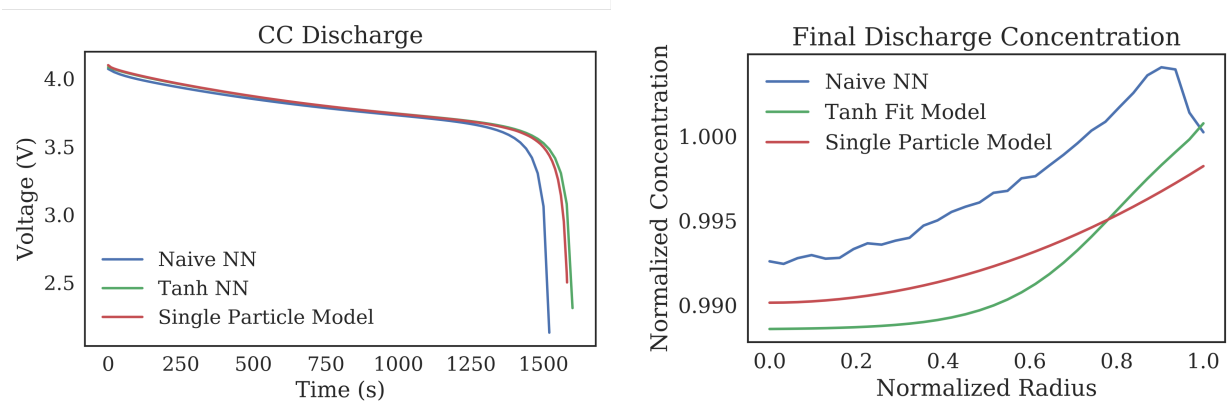


Figure 4.5: Looking at an unseen combination of model input parameters, it can be seen that the meta-estimator which wraps a hyperbolic tangent function outperforms direct value estimation. The naive implementation includes concentrations above 100% relative saturation, which are physically impossible.

It should be pointed out that in Figure 4.5b, part of the stopping criteria for the discharge is the surface concentration of the particle, which results in near-perfect match between the model, Tanh neural network, and Naive neural network. The discharge is finished when the voltage hits 2.5V, and this is largely a function of the concentration of lithium at the surface of the positive and negative electrode particles, in combination with the local overpotentials. However, the internal concentrations will affect future performance, so it is really the other values which should be used as the measuring stick, and it is clear that the Tanh NN outperforms the Naive NN in this regard, especially at relatively coarse sampling.

4.5 Applications With All Varied Parameters

Breaking the problem of voltage estimation up into a team of neural networks has several distinct advantages, namely access to internal states and the ability to decouple sampling from various regimes. For example, the applied current is strictly defined as an absolute current, which remains constant for both the positive and negative electrodes during constant-current discharge. The available parameters, namely a_p, R_p, and l_p, the length-normalized electrode surface area, particle radius, and electrode thickness, respectively, serve to scale this absolute current relative to each electrode. In this implementation of the SPM, the amount of active material is not held constant by default, as the active surface area is assigned and not calculated. This changes the effective material loading when any of the parameters are changed, and the surrogate model inherits this implementation detail. If direct effects of single-parameter modifications are desired, the surface area must be modified to account for these changes.

In the task of surrogate model creation, it is often up to the user to decide how many samples to create and to decide how to create them. While this can offer a great amount of flexibility, there are a myriad of ways in which decisions at this point can hinder the training results. For instance, ideally, a full factorial sampling scheme would be the most ideal, offering perfect information of the influence of each input parameter globally across the space. However, this sampling scheme scales as O(levels^n), where n is the number of

dimensions and levels is the number of distinct values for each parameter. For small numbers of levels and dimensions, this may be possible, but for 3 levels in the 16-dimensional space of this problem, 43 million simulations would be required.

Even once a sampling scheme is decided, level distribution must also be selected – popular choices are gaussian distribution, uniform random, equal-spaced, and logarithmic[10]. While an exhaustive review of sampling spacing techniques and sampling framework techniques is outside the purview of this work, a discussion is still warranted. For this section, as the dimensionality has grown over the previous section, Sobol[77] sampling will be used. Sobol sampling is a technique which pairs well with surrogate model development[10,17], as it originated as a technique for statistical sampling. While it is slightly less rigorous than its cousin, Saltelli[78] sampling, which is effectively a fractional factorial design, the rate of level duplication in sobol sampling is much lower, which generally gives better results for data science endeavors.

When no distribution of values is known, uniform random is a popular choice. However, in high dimensionality, uniform random sampling can result in a high degree of clustering and duplicated points[17]. To combat this, Sobol sampling uses a more structured approach, which significantly reduces exact point duplication in high dimensions, while still offering an effectively uniform random sampling paradigm. Previous work[10] has demonstrated that it is possible to map this uniform random spacing onto any other point distribution through the use of kernel density functions, which enables supports logarithmic, gaussian, bi-modal, or simple point mapping onto another (min,max) value set other than the stock (0,1).

Additionally, the explicit training of the concentrations and local overpotentials as a function of battery state decouples, to some extent, the effect of open circuit potentials on the discharge. This is shown in section 4.7, where the neural networks have the open circuit potential function modified and still track reasonably well with the model, which has also had its open circuit potential function modified. As the plots demonstrate, the results at these completely new circumstances align very well. It is worth mentioning explicitly that a model which attempts to predict only voltage as a function of model inputs, current, and

time would fare extremely poorly in this endeavor, as the open circuit potential is effectively hard-coded into the voltage estimates.

4.6 Applications with Charging

While demonstrating proof-of-concept implementations of surrogate models on constant-current discharge simulations is easier than doing the same for charing simulations due to the static nature of the current, the models offer significantly more utility when applied in a charging context. During discharge, the models are useful for estimating time remaining, estimating state of charge, and accurately forecasting performance based on varied operating conditions. However, in general, not much control can be exerted over the discharge state – the load for a phone, car, or computer may be throttled, but if certain performance is desired, the battery must deliver it.

Meanwhile, for charge, the device is typically not in use, as is the case with an electric car, or there is a surplus of energy that nets an effective charge of the battery, even during use, as with cell phones and laptop computers. In either case, the primary objective of the charge controller is to deliver as much energy to the battery as is reasonable within some upper and lower bounds and while maintaining safe operation of the device. It is worth noting that, in the context of phones and laptops, while the device may have some influence on the upper bounds available to the charge controller, the charge controller does not have to satisfy any current user constraints. Put another way, if a laptop is charging with 90 watts of available power and the users power request changes from 30 watts to 60 watts, nothing about the charging of the battery changes other than the upper bound placed on the current available during charge. This flexibility represents an opportunity to leverage the information inherent to these physical models in order to improve performance and extend life.

Pathak et. al.[8] have demonstrated that, by incorporating lithium precipitation kinetics into the pseudo two-dimensional model and properly calibrating the model to a particular cell chemistry, geometry, and scale, it is possible to more than double the effective useful life of a

battery simply by changing the dynamics of the charge cycle without any changes to the time constraints or upper current bounds. Effectively, by changing the charge paradigm based on information gleaned from the examination of internal states of the battery, specifically the overpotential at the negative electrode, it is possible to avoid a significant amount of lithium plating during charging at high currents and to extend battery cycle life through this avoidance.

In principle, the same physics could be incorporated in a more sophisticated version of the surrogate model demonstrated in this work, one which incorporated side-reaction kinetics based upon localized overpotentials. The current implementation explicitly tracks these overpotentials, meaning that this information would be available to any control scheme that implemented an equivalent surrogate model. In much the same way, a charge paradigm could be constructed which sought to maximize the amount of charge stored inside the cell while also keeping the plating overpotential above 0 volts, limiting the effects of lithium plating.

However, in order for the model to be able to examine those effects, this simpler version must be validated in the charge case, as has been demonstrated for the constant current discharge case. Furthermore, to implement the surrogate model in an optimization algorithm, it would be beneficial to train the surrogate model not just at different parameter combinations, as has been done here, but also at varying currents, such that the results might be valid at unseen charge currents.

For the charging simulation using a surrogate model, in order to get accurate results, it was necessary to change the default x values for the custom loss function from [0...32], which was used for discharge, to [-32...0], effectively priming the neural network to be able to fit the shape of the derivatives with the hyperbolic tangent function. As in the discharge section, the problem is formulated as a neural-ODE, where the team of neural networks predicts the derivative of the internal particle concentration and this value is applied for the given timestep.

One of the main challenges in modeling the charge of lithium ion batteries is the switching of charge modes, from constant-current (CC) to constant-voltage (CV). In the single particle

model (SPM), this is incorporated through boolean logic, which changes one of the algebraic constraints and modifies the informational flow through the model. During typical operation, the current is specified and the voltage is allowed to float. In Table 4.2, another equation is present for CC, which forces the actual applied current equal to i_{app}. During CV operation, this equation is modified such that the voltage is held at a constant 4.2 Volts, allowing the current to decay as the overpotentials shrink, reducing the rate of reaction at the positive and negative electrodes.

This is inherently a difficult task to ask of a single neural network, as the current is given explicitly as an input, and so must be valid for the duration of that time step for the estimated derivatives to be accurate. Wrapping this neural network in a nonlinear solver and modifying the current at each time step such that the voltage is equal to 4.2 Volts would achieve this task, as is shown below. However, this requires creating a consistent state across the whole team of 4 neural networks, which is extremely slow, as each iteration requires a separate call to each neural network, and convergence generally requires on the order of 15 function calls. Additionally, this method was not particularly effective, as the surface concentration was already close enough to the final concentration to trigger an immediate stop to the current, cutting the CV portion of the charge simulation extremely short.

However, by implementing another neural network which, given the internal concentrations of the positive and negative particles, the voltage, and the input parameters, sought to predict the self-consistent current for the CV portion of the charge, high accuracy was achieved, with constant-time optimization occurring at each time step. The voltage and current agreement between the surrogate and physics-based models are shown in Figure 4.6.

While timing these models, the native C solution for the Single Particle Model runs in 16ms for the above discharge simulation. Each of the neural networks, when implemented in dot products and summation of in-place NumPy arrays, take 20us to execute. A typical time step is 10 seconds, meaning that for the above simulation, roughly 500 time steps will need to take place. Most of the time is taken during the constant current portion of the charge, so an optimal implementation in Cython would be expected to take roughly 40ms, slightly

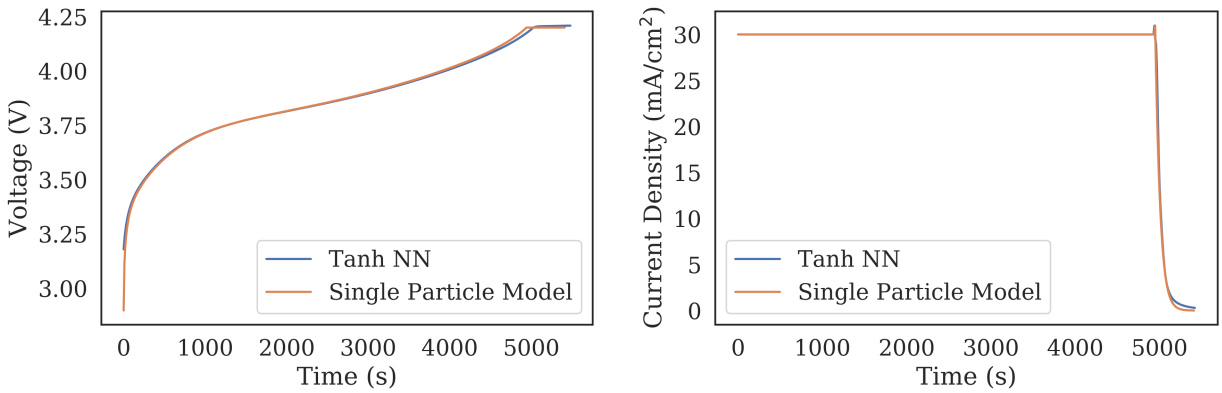


Figure 4.6: By implementing another neural network, tasked with current decay prediction, it is possible to enable constant-time operation during constant-voltage charging, eliminating the need for a nonlinear optimization algorithm and giving accurate results.

slower than the highly optimized IDA solver requires to solve the numerical model. However, when this method is applied to a more computationally expensive model, such as the pseudo two-dimensional (P2D) model. The numbers shift. The solve time for a grid-independent P2D model solution in Ampere is 2.13 seconds, whereas the same scaled-up application of these models would take only 380ms. These relationsihps are shown in Figure 4.7 below.

4.7 Modifying Open Circuit Potential

The open circuit potential, or OCP, is a thermodynamic property that relates the potential of the lithiated material relative to Li^+. As the cathodic material becomes depleted of lithium, the OCP will tend to decline, although the relationship between the potential and degree of lithiation is material specific. As the anodic material becomes more lithiated, its potential will tend to rise. As the open circuit voltage of the cell is calculated as the difference between the voltage at the cathode and the voltage at the anode, the relative degrees of lithiation of these two electrodes can drastically change the shape of the discharge curve, even with unchanged OCP. This relationship is demonstrated in Figure 4.8.

Importantly, this scheme of surrogate modeling decouples many of the phenomena that

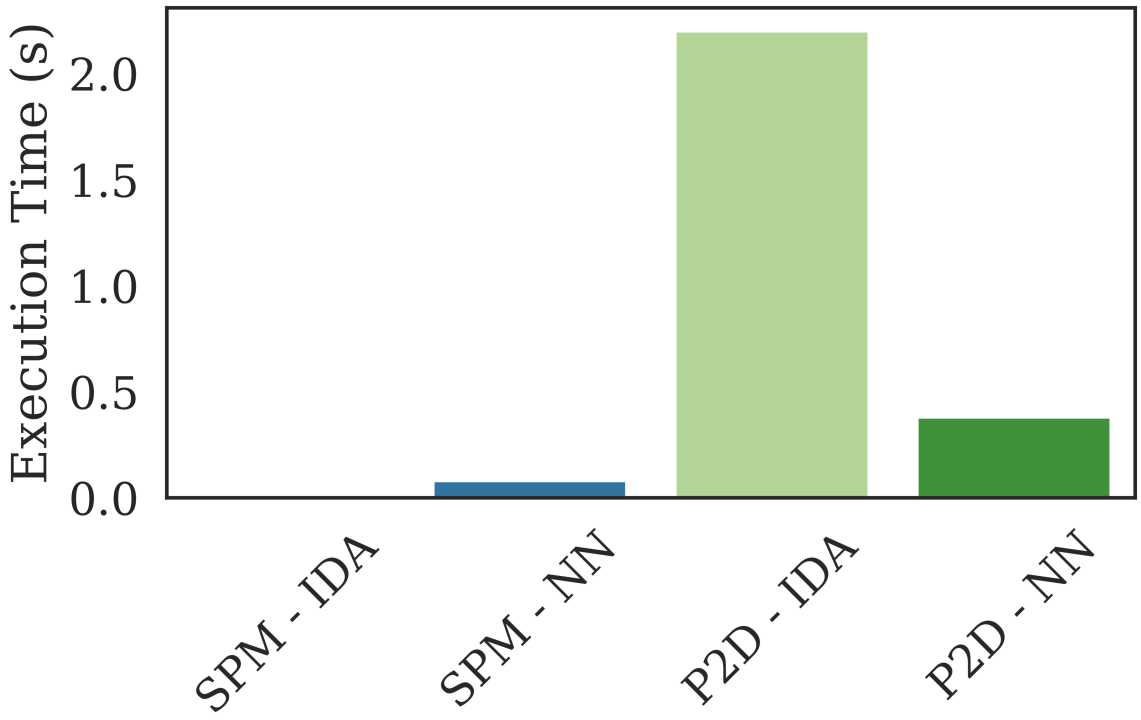


Figure 4.7: While smaller models, like the Single Particle Model, can outperform the equivalent surrogate model, more complex models will see a notable performance benefit.

occur in the cell, while still retaining the accuracy associated with self-convergent values at each of these unseen states. For instance, the particle concentrations are being directly tracked and calculated, as are the surface overpotentials associated with the reaction kinetics. This means that the OCP of each material can be modified arbitrarily and, as long as the voltage value has been seen during training, it is highly likely that the model will remain valid. To demonstrate this, the OCP of the cathode was lowered by 0.5 volts at all degrees of lithiation. This representation was fed into the numerical and neural network-based models, and the results are shown in Figure 4.9. Even at these unseen OCP values, the neural network model is indistinguishable from the numerical model, indicating the generality afforded by this problem formulation.

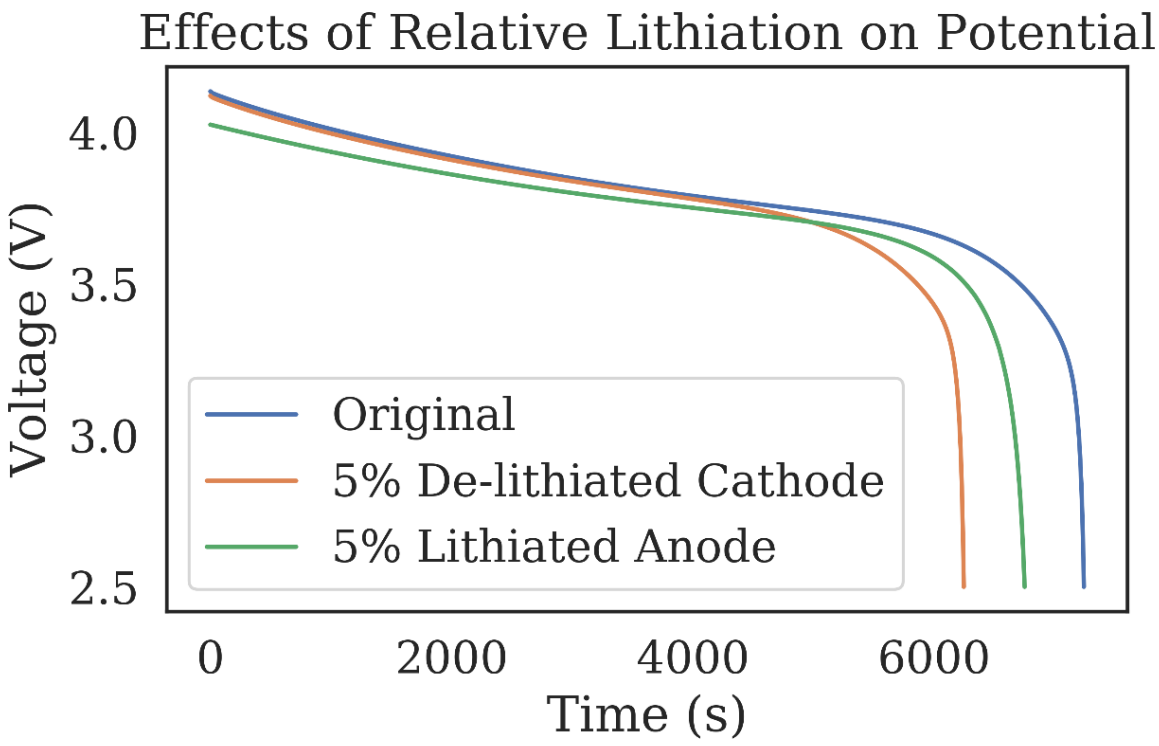


Figure 4.8: By modifying the initial conditions to slightly de-lithiated anodic and cathodic conditions, the resulting shape of the discharge curve is changed, even with identical open circuit potentials. Modifying the open circuit potentials adds more variation.

It is possible to create arbitrary waveforms, fit them using methods available in SciPy and NumPy, and apply these waveforms to the model. In order to replicate these results in the numerical model, it is necessary to generate a function to be fit, call an optimizer on that, and then take those values and replicate both the values and the function in C. Working with the models natively in Python offers significant advantages in terms of flexibility, ease of use, and choice of representation. For instance, it is popular to represent the complex shapes of open circuit potentials with polynomial ratios or ratios of hyperbolic tangent functions, which are difficult to fit due to their nonlinearity. Using Python, these same curves can be fit with high accuracy using splines, which solve in the same amount of time as these simple

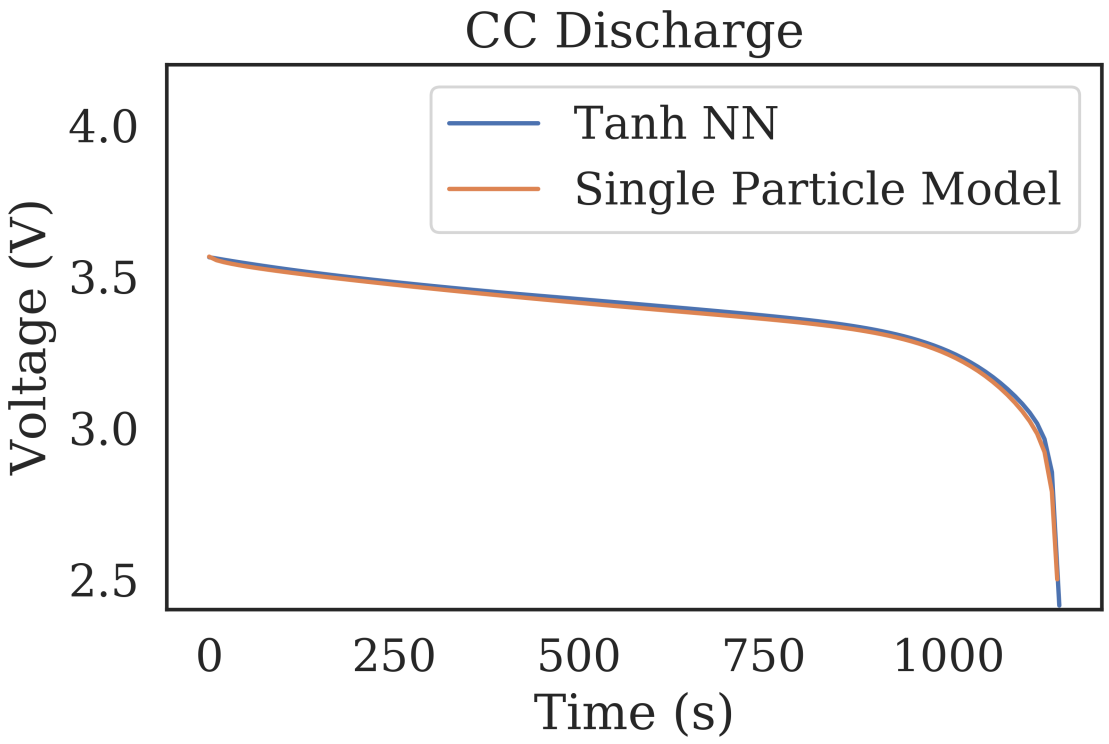


Figure 4.9: By modifying the open circuit potential of both the numerical model and the neural network model, it is possible to demonstrate the generality of this problem formulation. Even with unseen open circuit potentials, the models are in excellent agreement at unseen input parameter combinations.

functions, but are much more consistent in their ability to represent experimental data.

4.8 Conclusion and Future Work

By creating teams of neural networks, which forecast future self-consistent states, it is possible to rebuild expensive numerical models using relatively cheaper neural networks. In this work, the Single Particle Model (SPM) was used as the base for this neural network framework. During iterative forecasting tasks, such as iteratively predicting the future concentrations inside of anodic or cathodic particles, accuracy and self-consistency are of

paramount importance. If the noise beings to overwhelm the signal, the neural networks can quickly become divergent and estimate inaccurate future states.

In order to decrease the scale of this noise, the problem was framed as a neural-ordinary differential equation, in which the neural network was tasked not with predicting the future states, but with predicting the derivative at the current time point. By framing the problem in this manner, it is possible to take larger time steps if lower accuracy is needed, or more time steps if higher accuracy is required, explicitly trading computational runtime for accuracy. Additionally, the noise can be reduced significantly and physical consistency can be maintained if the neural networks are tasked not with predicting the derivatives themselves, but acting as meta-estimators in order to simultaneously predict and fit a hyperbolic tangent function, which can accurately represent the shape of the derivative. By mapping the derivative through this function, the output space is compressed from 30 dimensions to five dimensions, and physical consistency is maintained.

In order to see the computational benefits of a surrogate model as expensive as iteratively calling several neural networks for each time point, the numerical model must be sufficiently complex. While the highly optimized IDA solver can outperform the neural networks on the SPM, it is projected to be unable to do so with the more complex pseudo two-dimensional (P2D) model, which needs 2.3 seconds to solve, while neural networks, which can represent the same data present in the model, could be solved in only 400 milliseconds. Additionally, the neural networks offer significantly more flexibility in terms of open circuit potential (OCP) fitting, as the powerful curve-fitting tools available in Python are available natively, and the neural networks are implemented in matrix multiplication in NumPy. This also makes these surrogate models significantly easier to distribute, as no C compiler is required in order to utilize them.

In addition to the constant-current discharge and constant-current, constant-voltage charge simulations performed here, it is possible to string together partial charges and discharges through Ampere, which would extend the usefulness of the surrogate models to mimicking drive cycles. These much more realistic scenarios would require more generaliza-

tion, and may require something like a switch function as the current goes from negative to positive, as the charging implementation required modifying the hyperbolic tangent function input range from [0,32] to [-32,0] for acceptable performance. The code used to generate the neural networks mentioned in this work, as well as Jupyter notebooks which walk through some of the thought processes, are available at https://github.com/nealde/nn_spm.

Chapter 5

AMPERE - CREATING AN OPEN-SOURCE, HIGH-PERFORMANCE BATTERY SIMULATION PACKAGE FOR PYTHON

5.1 Introduction

Python is a high-level, interpreted programming language which has come to dominate the open-source scientific computing scene. This dominance is thanks, in large part, to Numpy[100], Scipy[79], Matplotlib[102], Sci-Kit Learn[25], Pandas[103], and other low-level packages which make high-performance computing accessible through Pythons high-level syntax and powerful classes. These packages use a combination of C, C++, Fortran, and Cython[104] libraries to bring high-performance computation to Python.

Originally created as a teaching tool, Python began with a few core philosophies, including: beautiful is better than ugly, explicit is better than implicit, and readability counts. Following these guidelines, Python uses whitespace, or space and tab characters, as official formatting for the code, forsaking the braces of C and Java. This forces proper formatting on the code, enhancing readability. While the interpreted nature of Python allows for some of its core strengths, such as variable type mutability, it also results in significantly slower execution of code. As such, very few effective differential algebraic equation solvers exist for Python, which causes a break in the data analysis pipeline, as data must be generated externally, even if it is analyzed using Python.

While calling external functions from Python is fairly straightforward using command line, doing so in a robust, parallelizable way is very difficult. Additionally, eliminating the overhead from serialization or writing data to a file is impossible, and memory is significantly faster than disk, even with the prevalence of solid state drives. As such, these methods prove

to be both slower and more brittle than a proper implementation, which will be described more in the second section.

5.2 Why Make a Python Package?

Data science, often hailed as the fourth paradigm of science[11], is a large subset of computer science which encompasses data visualization, machine learning, and data management. As data sets grow, it is important that the data be intentionally managed, if it is to be efficiently utilized. Previous works[10] have applied data science methods to numerical modeling of lithium-ion batteries, often with interesting and powerful results. These projects often required tens of thousands – if not hundreds of thousands or millions – of simulations to take place, creating gigabytes of data in the process. In order to make this process efficient and repeatable, a module-style format was the obvious choice.

Originally, the package was intended for personal use and not for publishing. However, as the project evolved, it became apparent that the difference in effort between a useful package and a useful, distributable package was fairly small, and that efficient implementations and powerful, simple syntax coupled with best practices for package development would improve the result in the long run.

Although some previous works[17] have not used this project, more recent efforts[10] have made heavy use of the Ampere package. While the details were not captured in those published works, they will be recounted here.

In On the Creation of a Chess-AI-Inspired Problem-Specific Optimizer for the Pseudo Two-Dimensional Battery Model Using Neural Networks[10], several different neural networks were trained as meta-estimators for the Pseudo Two-Dimensional (P2D) physics-based lithium-ion battery model. Rather than using the neural networks to replace the physics-based model, the neural networks were trained to become an expert on the model system, acting a input parameter recommenders to aid in the endeavor of model calibration to experimental data. This task is difficult, as the model is nonlinear, with varying parameter sensitivity across its 26-dimensional input space. However, in order to make use of the model

in design or control applications, the model must first be calibrated.

First, a static training set of 200,000 simulations was trained across the 9 input dimensions, which were selected for a combination of symmetry and high sensitivity. The parameter values at each of these points was selected in accordance with a Sobol[77] scheme, which gives more efficient parameter spacing in high dimensional applications than uniform random variance. The first goal of Ampere was to make this process less painful, as although a single data set may be included in the published work, many more will have been created and discarded along the way. Having this process as efficient as possible was a priority, as the creation of data is often the most computationally expensive portion of the data science pipeline.

In order to deliver a data generation function to Python, the model had to first be wrapped in a language with adequately robust solvers. The specifics of the implementation will be covered in the next section, but the thought processes will be covered here. Differential algebraic equations, or DAEs, are traditionally tricky to solve, as they are ordinary differential equations with strict algebraic constraints. In order to begin stepping forwards in time, the initial conditions must be satisfied, which often requires a solver with the ability to initialize these problems. In general, popular explicit solvers, including Scipys odeint, do not come with this capability, and as such are unsuitable for DAEs.

Julia, a new scientific computing language, is extremely powerful, offering a robust suite of solvers, optimization packages, and just-in-time compilation, all of which add up to a very capable scientific computing language. While it is possible to call Julia functions from Python, and vice-versa, this process is not elegant or smooth for user-defined functions, and when wrapping a command-line style Julia call from Python, the just-in-time compilation must be done for each function call, rendering any compilation advantages null. Due to the early stages of Julia when this package was implemented, static compilation was in the pipeline, but was not near completion. Methods for keeping the Julia kernel alive and communicating through UDP or text files were explored, but these methods were not suitable for distribution.

After exploring several options, a solution was found in the wrapping of IDA, an efficient time solver which is capable of handling and remedying inconsistent initial conditions. The exact details of how this powerful solver was wrapped will be covered in depth in the next section. The end goal was to have a workflow which allowed memory to be pre-allocated in Python, filled with relevant values by C, and passed back to Python, thereby eliminating any serialization overhead associated with static compilation and command-line calls, which can be significant, especially for fast programs with large amounts of data, as is the case in numerical modeling applications.

Once the base model was wrapped, other functions could be built and successively called, enabling flexible and powerful applications to be controlled in just a few lines of code. One of the most important to the discussed work was the fit method, which iteratively calls the numerical model, modifying the input parameters in order to reduce the error between some experimental data and the simulation. The nature of a modular package is what enabled the analysis at the scale of the published work.

In On the Creation of a Chess-AI-Inspired Problem-Specific Optimizer for the Pseudo Two-Dimensional Battery Model Using Neural Networks[10], once the neural networks were trained, it was necessary to evaluate their performance and compare it to the state-of-the-art for model initialization, the genetic algorithm. Ampere was instrumental in enabling analysis at this scale. In order to eliminate the chances of cherry-picking data, the fitting methods were repeated 100 times for each beginning-ending pair, for each neural network, at each level of sampling. The results of this analysis are shown in Table 3.5.

For this analysis, 100 pairs of initial and target parameter values were established. These were generated in order to replicate the process of fitting unseen experimental data while removing any uncertainty associated with model error. Effectively, any inability of the optimization algorithms to perfectly match the target curves would come solely from an inaccurate estimate for the model input parameters. For comparison, the best-performing optimization algorithm, Nelder-Mead, was able to reduce the initial error from an average of 375 mV to an average of 8.7 mV, compared to the neural network-based method, whose

best performance achieved an error of 0.13 mV.

For each of the 6 neural networks, which were trained on data generated by Ampere, for each of the 6 levels of sampling, and for each of the 100 A-B pairs created for the express purpose of optimization, it was necessary to start with an initial guess, optionally refine it with the neural network, and then call the Nelder-Mead optimization algorithm for between 600 and 3900 function calls as the optimizer iterated to convergence. Each of these optimization algorithms was fit over two simultaneous discharge curves, which doubles the number of effective simulations that are necessary. Ampere was instrumental in allowing this process to be parallelizable, repeatable, and error-free across the estimated 20 million simulations that were required for the complete analysis.

The previous chapter would have been significantly more difficult without Ampere. By default, the returned solution in Ampere is the state of each variable at each predetermined time stamp. It is only when the internal states are not requested at the package level that this information is discarded. This design decision was instrumental in extending the functionality and enabling teams of neural networks to slice up the model and iteratively rebuild the internal states, which is the major focus of the previous work.

While the first two sections, which focused on slicing up the internal states of the model and building out teams of neural networks to rebuild the external states at each time point, would have been possible using some other simple black-box formulation, the final section, which trained based on randomly generated currents for predetermined time slices, would have been considerably more difficult without the formal package structure and the foundation associated with properly modularized code. In particular, the piecewise current function added significant value to this section and enabled the generation of training data, the fitting of the models on the experimental drive cycle data, and the evaluation of the drive cycle data.

5.3 Implementation and Design Choices

The previous section outlined the motivation for the creation of a more formal and structured python package based on the anticipated and actualized use cases. This section focuses more on the specific implementation details and design choices which have gone into the package creation, including how the files are broken up, how the API was designed, how the low-level model talks to the high-level objects, and more specifics on why these particular tools were chosen.

The biggest inspiration for the design and, to a more limited extent, the implementation of this package was Sci-Kit Learn[25] and the associated publications which described the design choices made there. Sci-Kit Learn is an extremely powerful object-oriented package which offers world-class machine learning implementations in a friendly, well-documented API which natively supports Numpy and Pandas data structures. Its popularity is a testament to the demand for high-performance libraries in higher-level languages, and the design choices there, including documentation style, reasonable default selection, and available methods serve as guides for Ampere.

As a case study, the RandomForestRegressor object will be examined. Formally, a random forest is an ensemble of decision trees, all of which have been randomized such that they produce different structures, and the outputs of these decision trees are averaged in order to allow for interpolation between the training data points. Digging into the source code reveals an intuitive approach to functionally similar objects, namely, the instantiation of a base class for a bulk of the functional code, and particular objects for each desired final object.

Specifically, in the RandomForestRegressor, the final available methods are: apply, *decision_path*, *fit*, *get_params*, *predict*, *score*, and *set_params*. Of these, *fit*, *predict*, and *score* are the most commonly used, as they represent the core functionality of a statistical predictive model. Due to the nature of differences between regression algorithms and classification algorithms, there is another set of base classes – one for classifiers and one for regressors, which implement the remainder of the functionality. This leaves RandomForestRegressor

and ExtraTreesRegressor, an especially randomized variant of a random forest, as simply instantiations of the ForestRegressor base object, and the only things specific to one or the other are documentation and default settings. ForestRegressor builds upon BaseForest, which implements apply, fit, and decision_path, which are common to all forest objects. The relationship between object inheritance, methods, and particular instances illustrated in Figure 5.1 below.

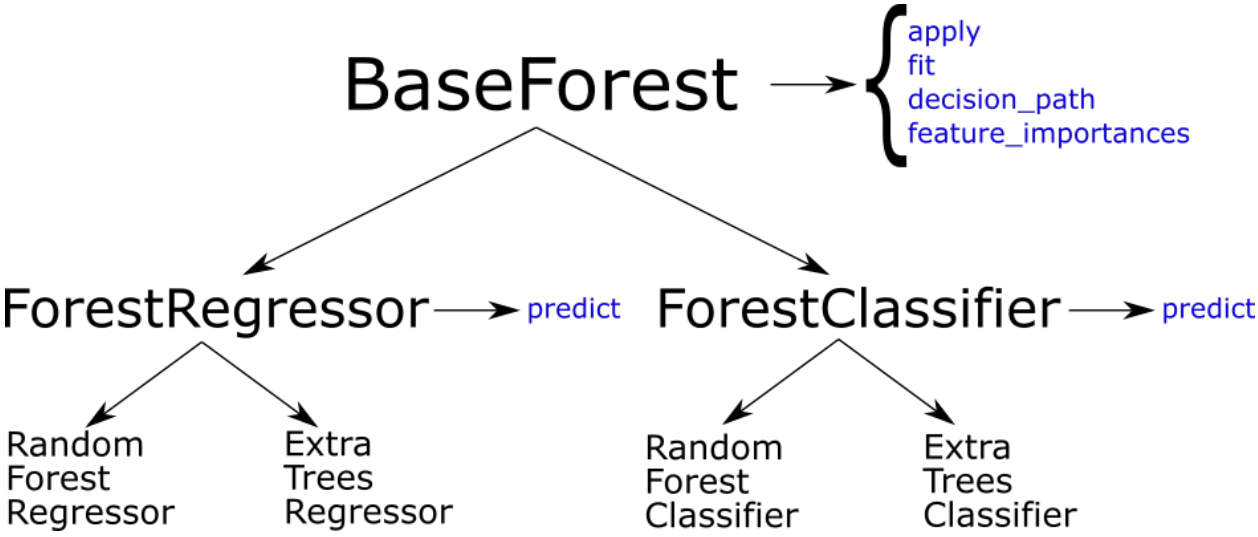


Figure 5.1: Most of the functional code is implemented in a base class, which substantially reduces variance between objects and increases maintainability of the code. Regressors and classifiers require different prediction functions, so those are implemented at one level above base.

This same approach is taken in Ampere, where a base battery object is created with a significant portion of the available methods and features, and the individual models are built on top of this base battery object with only minimal variable assignments necessary, varying mostly in default settings, available values, and specific functions. The hierarchy for Ampere is shown in Figure 5.2 below.

Due to the similarity of the functions, only a single base class is needed, which implements the core functionality of the package – *charge*, *discharge*, *fit*, and *generate_data*, in addition to cycle and *piecewise_current*, which are built directly on top of *charge* and *discharge*. Using

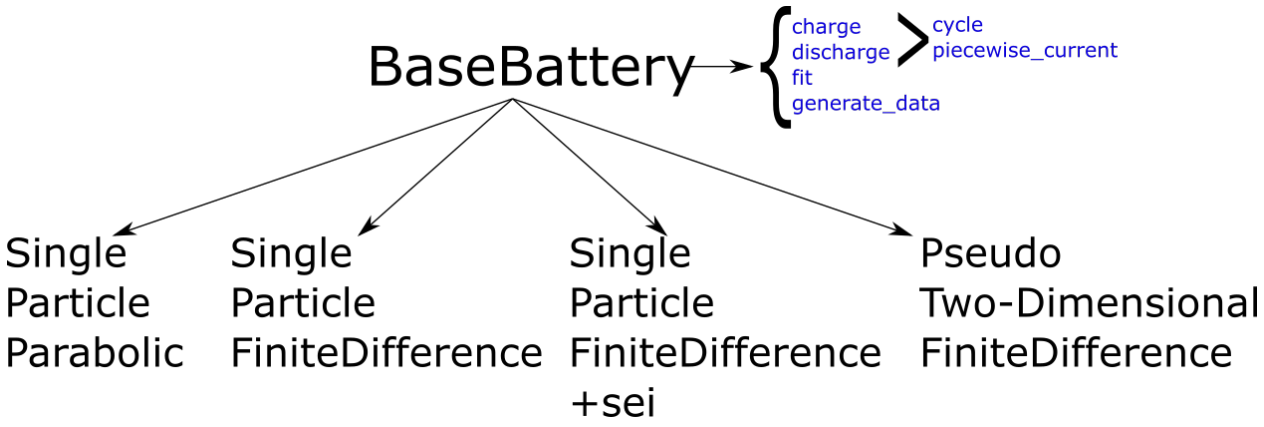


Figure 5.2: Similarly, the only functional differences for Ampere are instantiation attributes, such as lists of available parameters and functions to be called. As such, only a single base class is necessary.

these available functions, it is possible to initialize a model, calibrate it to a particular battery for which any amount of experimental data exists, and then evaluate the model at future states in order to predict the performance of the battery at unknown conditions. In addition, this same framework will support the high-accuracy surrogate models created in Chapter 3, which will require all the same methods, but which will interact with different functions under the hood.

In addition to these available methods, there are also attributes, which track the history of the object. The most important of these include history, an attribute which tracks the optimization pathway and can be useful for exploring local minima in the error landscape, *current_state*, which is heavily utilized in *piecewise_current*, as it allows for the chaining of multiple charge and discharge cycles, partial or otherwise. The ability to track the current internal state of the simulated battery and give this as new inputs into subsequent simulations was one of the major motivations for creating a proper Python package, as doing so manually would require a lot of specialized code, and the additional effort required to make the results reproducible, general, and robust is fairly small by comparison.

In addition to shaping the design from the perspective of the API, Sci-Kit Learn has

been instrumental in shaping the way the C functions are passed to Python. Originally, the C functions were precompiled and read in modified text files, outputting the results of the simulation to other text files. While this process is relatively fast, it is inherently unsafe, especially when multithreading is concerned, as the threads would overwrite the same input and output files, causing lock errors and stale reads. The next iteration called precompiled C functions from command line, printing the results to the console. This formulation was significantly safer than the previous iteration and supported parallel function calls, but it also required serialization and deserialization of the output array, which can be the bottleneck for fast simulations, extending computation time by a factor of 10.

In order to achieve the desired high performance, it was necessary to pre-allocate contiguous memory in the form of a Numpy array, pass a pointer to that array to the C function, and have the function populate that array with values as the solution proceeds. When the solution is finished, flow control passes back to Python, where the values can be read in-place. This formulation is thread-safe, fast, and inherently supports Numpy arrays, which are heavily utilized in downstream processing of the data within the Ampere package.

Initially, this required precompiled C functions, which are difficult to distribute, as the act of compilation is particular to the computational environment the function is created in. In order to make these functions distributable, it was necessary to wrap the C function in Cython, a compiled variant of Python, which can natively interface with C functions, ships with a C compiler, interfaces nicely with Numpy, and automatically compiles into a Python module, which can easily be imported into Ampere. Although distribution of these files in a package context can prove difficult, it is one of the most robust ways of interfacing Python and C functions that exists.

5.4 The Future of Ampere

Currently, Ampere features 4 numerical models – the Pseudo Two-Dimensional (P2D) model, the single particle model, the single particle model with SEI layer growth, and the parabolic profile approximation for the single particle model. Each of these models are

discretized using a finite difference scheme, which is the simplest to understand, and also one of the worst in terms of the order of accuracy. Practically, this means that in order to get a self-consistent, or grid-independent, result, many node points are needed. This is unfortunate, as the number of equations for the pseudo two-dimensional model scales as O(n*m), where n is the number of nodes in the X direction, and m is the number of nodes inside each particle. This results in extremely large systems for moderate numbers of nodes, which significantly increases the computational expense of the model, even with a high-performance solver like IDA.

Alternative, higher-order discretization schema exist, including various forms of Galerkin and Orthogonal collocation[3], these methods can be difficult for formulate without significant help from algebra engines like Maple or Mathematica. Other higher-order forms, such as Lobatto IIIA, can be applied to the solid-phase diffusion within the spherical particles, significantly reducing the computational time by cutting the number of node points necessary for grid independence[105]. Other options exist, including finite volume, which offer substantial improvements over the existing finite difference solution.

In addition to increasing the numerical efficiency of the existing models, it would be useful to extend the physics available in the models – including solid-phase diffusivities which vary with concentration inside the particle, adding stress-strain relationships in the particles, and potentially adding support for alternative particle geometries, including nanowires.

Several data science projects have been implemented using data generated by Ampere[17,10]. Each of these projects has created a useful tool which can be used to aid in the application of ampere, either through the creation of a surrogate model which can approximate the output of a more expensive computational model, or through the creation of a problem-specific optimization algorithm which can significantly reduce the computational cost associated with model calibration while also improving the final converged error. These frameworks have been successfully generalized, and their addition to Ampere would extend the current capabilities of the package.

In addition to improving models, extending the range of simulated physics, and including

the machine learning-based tools which have been developed using simulations from Ampere, important metrics for batteries exist outside the scope of single charge and discharge curves. While the single particle model with SEI growth is a good start to modeling degradation, the framework for Ampere could be enhanced in order to easily support the tracking of various metrics as a function of cycle life. For instance, initial capacity could be calculated and tracked as a function of cycle life, which will change only if a degradation mechanism exists in the model. Popular metrics to track over the lifetime of a battery include power density, energy density, internal resistance, and capacity retention. The addition of these metrics, coupled with an accurate degradation mechanism inside the models, would greatly extend the applicability of Ampere for scientists looking to investigate battery longevity.

The outlined future projects vary wildly in scope, as some could be implemented in a few lines, and others would require a substantial effort for an extended period of time. While the timelines for these features have not been defined, the package will continue to be improved and supported by its creator and maintainer, and performance and usefulness will continue to improve.

5.5 Conclusion

Open source scientific computing tools are essential to the acceleration of scientific analysis. Powerful analysis libraries, including NumPy, SciPy, Pandas, and now Sci-Kit, have brought high performance data analysis capabilities to Python, an otherwise slow interpreted language. By having high-level interfaces to these low-level libraries, it is possible to get the best of both worlds: maintain an elegant, intuitive, and well-documented interface in an interpreted environment, while giving the performance associated with efficient implementations in compiled libraries. By modeling Ampere, a library for high-performance lithium-ion battery simulations, after world-class open source software packages for Python, this same model can be applied to in-house implementations, making performant battery models more accessible to researchers. The open-source package is available at http://github.com/nealde/ampere.

Chapter 6

CONCLUSIONS AND FUTURE RESEARCH

6.1 Thesis Summary

Lithium-ion batteries are complex electrochemical devices whose internal states are challenging to infer from measurable external states. Understanding, quantifying, and tracking these internal states are of vital importance for proper monitoring, control, and design of future electrochemical cells. Physics-based models offer unique insight into the inner workings of these systems, but require careful calibration in order to accurately portray any cell in particular. In this thesis, several frameworks for leveraging the information inherent to these physics-based models have been outlined, with applications in design, control, and degradation analysis of lithium-ion batteries. Of the suite of physics-based models, two of the most popular, the single particle model (SPM) and pseudo two-dimensional model (P2D), are utilized. Chapter 2 outlined a framework for the creation and analysis of black-box forwards, inverse and inverse models, for the purpose of model calibration. Additionally, recurrent formulations were constructed which offered high accuracy in popular control tasks, such as voltage or state of charge prediction at a time horizon. Chapter 3 leveraged a problem formulation from a chess algorithm and generalized the result, training a neural network to act as an expert in the task of P2D model calibration, and improving the converged error by 30-fold. Chapter 4 examined the limitations of neural networks in a physically interpretable application, and offered some meta-estimator frameworks in order to enforce relative constraints on the output layer, which allowed for a higher degree of physical consistency. Chapter 5 explored the implementation of Ampere, Python package for high-performance lithium-ion battery models, which was created with the goal of making sophisticated analysis more accessible to electrochemical researchers.

There have been many lessons over the course of this corpus of work, but the most prominent of these is this; data science project success is a function of problem formulation, and proper problem formulation requires both creativity and domain expertise. In Chapter 2, tree-based models were used to create surrogate models of a computationally expensive model for the expressed purpose of fitting the surrogate model, as it solved several orders of magnitude faster than the physics-based model. This method was fundamentally flawed, as tree-based models are constant output, meaning that optimization was being attempted over previously seen values, rather than accurately interpolating between seen points. However, by modifying the target application, this data could be successfully leveraged to create predictors which were useful for control paradigms, both for voltage and state of charge prediction.

In data science, it is very difficult to come up with a solution which works only on one problem, as the tools are sufficiently generalizable that solving a problem in one domain naturally extends to other domains. In Chapter 3, inspiration was taken from artificial intelligence programs tasked with mastering the game of chess, which is effectively an integer optimization problem. By taking the comparative framework developed in Deepchess, generalizing it for any problem, and applying it to the P2D model, it was possible to develop an expert recommender system which worked similarly to a chess algorithm – given an initial guess, it would recommend modifications to the model input parameters which would minimize the error between the model and the given target data, for the cost of a single function call. A key insight in this project was the idea that the neural network could not replace a black box optimization algorithm, which can fit to arbitrary tolerance, but could augment it by excelling where these algorithms are the weakest – when the search space is large, and the error is high. Together, the neural network and optimization algorithm can outperform a genetic algorithm and the same optimization algorithm by 30-fold while also using fewer function calls.

The final aspect of problem formulation is physical consistency considerations, which can explicitly enforce constraints on the relative values of the neural network outputs during

training time. The logical extension of this practice is to use the neural network again as a meta estimator, where it does not fit the data directly, but generates a form which fits the data through another model whose outputs are limited to certain shapes. In Chapter 4, neural networks are used as meta estimators for hyperbolic tangent functions, whose shape closely resembles that of the change in internal particle concentration during charge and discharge of the single particle model. By mapping the neural network through this hyperbolic tangent function, it is possible to get higher accuracy and better generalization in coarse sampling scenarios while also maintaining greater physical consistency. Another key insight with this work was the framing of the problem as a neural-ode, where the predicted values are the instantaneous derivative, which mitigates the influence of the noise in both the direct estimation and meta estimation scenarios. By building teams of neural networks which track internal particle concentration and local overpotentials, it is possible to rebuild the external states of the battery with high accuracy while maintaining the internal states without the overhead of a complex physical model.

In order for tools to be useful, they must be implemented and distributed. Towards this end, Chapter 5 covered the implementation details of Ampere, an open-source Python package for high-performance lithium-ion battery models. By taking the same framework used to do the data science projects in Chapter 3 and Chapter 4 and documenting, distributing, and maintaining it, the barrier to replication is significantly lower, and the community as a whole benefits. It is towards this end that each chapter is accompanied by a github repository which hosts documented code necessary to replicate some or all of the results.

6.2 Creation of Surrogate Model for Pseudo Two-Dimensional Model

While the work in Chapter 5 is exciting, its application to the single particle model (SPM) limits the ultimate usefulness. The SPM, without electrolyte effects, is only relevant at relatively low currents. The pseudo two-dimensional (P2D) model contains significantly more physical phenomena, including electrolyte concentrations, volume-average, depth-varying concentrations and overpotentials, and internal particle concentrations which

also vary as a function of depth. The addition of these physical phenomena greatly extend the applicability and predictive power of the model.

The same framework introduced in Chapter 5, using teams of neural networks to rebuild internal states from computationally expensive, physics-based models, can also be applied to the P2D model. In this work, the internal particle concentrations and local overpotentials at the positive and negative electrode were tracked, with one neural network being used for each. This is shown in Figure 6.1 below. Effectively, each neural network specializes in a single physical phenomena, which allows for better generalization and more physical consistency than the naive approach, which would seek to use a single neural network to predict each of the tracked variables at each time step.

In order to extend this to the P2D model, more neural networks would need to be introduced, to account for the additional physical phenomena. Specifically, in the P2D model, the following extra states are tracked: depth-varying solid phase potential, liquid potential, and electrolyte concentration. In the model, at each point in X, or the depth of the electrode, a representative spherical particle exists, with its own independent concentration. Effectively, to replicate the applied approach for the single particle model, these concentration profiles could be used as replicate data for a single neural network, where the additional input of the solid phase and liquid phase potentials are given as inputs. These values, which change across the X dimension, are what drive the electrochemical behavioral differences across the electrode. As such, this problem is well-suited to leverage the increased informational content of the P2D model.

6.3 Creation of Hybrid Data-Driven Models

Flexible, powerful tools like those in the data scientist's arsenal are only going to become more useful, easier to apply, and able to solve more problems as time goes on. With this in mind, it is important to consider these data-driven models as almost the opposite of first-principles models – by generating or collecting a large amount of data, predictions can be made through nonlinear regression. This is an inefficient use of data, but as data as becoming

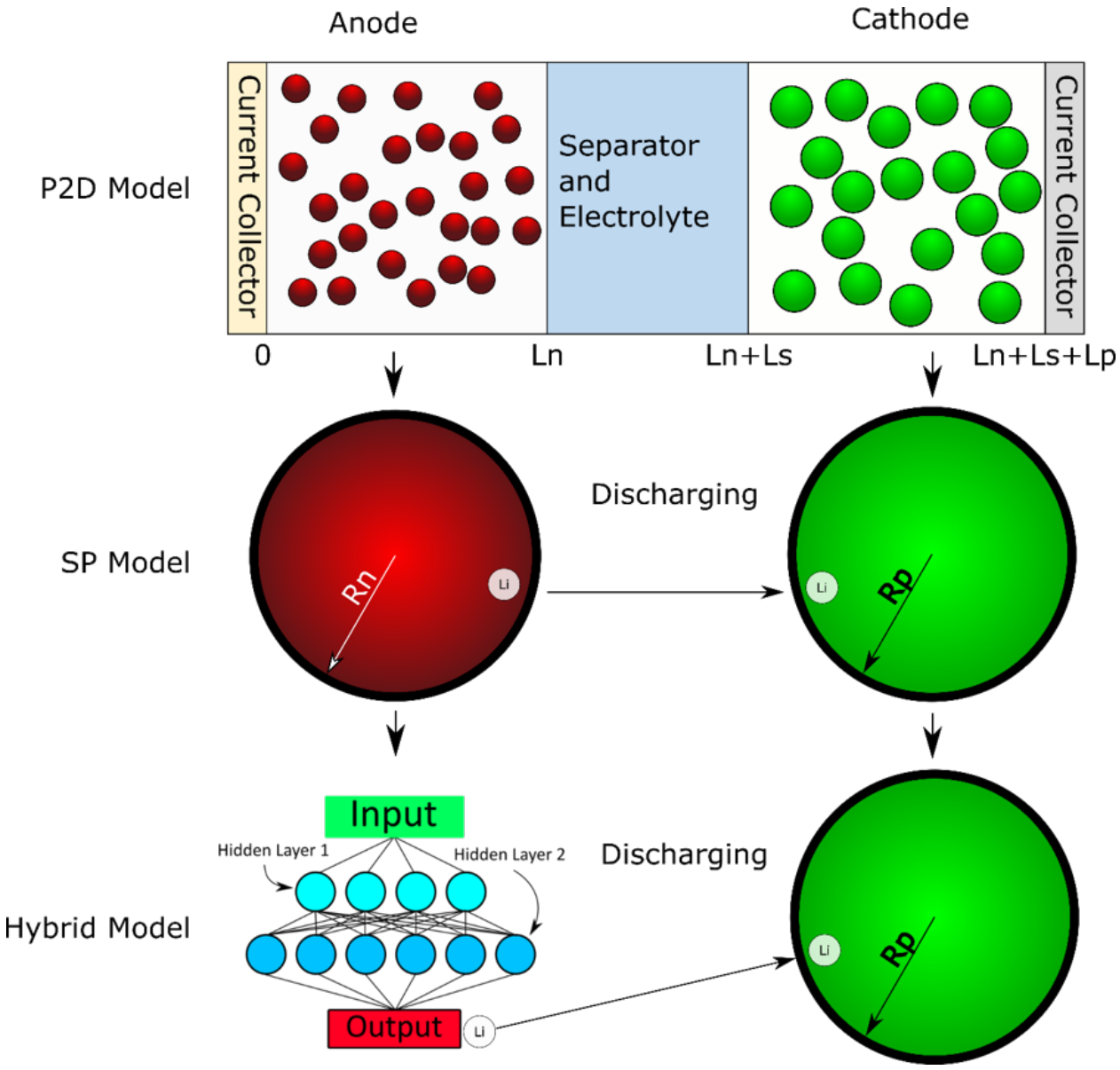


Figure 6.1: Using teams of neural networks, the individual particles in the Single Particle Model and the Pseudo Two-Dimensional Model can be replaced. These can then communicate with pre-created (oracle) data for validation, as is shown above, or can communicate amongst other neural networks for completely NN-driven prediction.

cheaper and easier to acquire, inefficiency here is no longer the limiting factor. By contrast, first-principles models take a small amount of data and fit a strictly defined model, offering

a high degree of predictability at unseen states, but requiring a large amount of effort to formulate and validate.

The future of scientific applications of machine learning tools lie at the boundary between first-principles models and data science techniques, where one can be used to enhance the other. Karpatne et. al.[24] summed this idea up very succinctly, recreated in Figure 6.2 below. This can come in many forms – for instance, perhaps a version of the single particle model does not contain effects from temperature. By fitting this model and training a machine learning technique on the deviations from experimentally collected values, it could be possible to use data science techniques to extend this model to applications where it was previously invalid. Others[106] have used machine learning bridge the gap between high-fidelity thermal models and less sophisticated analytical models, which has the benefit of being differentiable at all points, and fast to compute. By enhancing these simpler models with machine learning techniques, it is possible to get the best of both worlds – a fast, differentiable model, and an accurate model.

As data science becomes more popular and ubiquitous, scientists must take care to ensure that these tools are used to augment understanding, and not as a crutch to replace it. Many data science projects can seem like a scientist with a hammer looking for a nail, and applying data science techniques simply for the sake of doing so. It is imperative that the need drive the solution, and not the other way around.

One example of a current need in the battery world is solid-electrolyte interface (SEI) formation. The SEI is one of the most poorly understood parts of a lithium-ion battery, although this is changing rapidly. There is an opportunity to increase the predictive power of numerical models through the inclusion of a degradation model that mimics the effects of SEI, and it is plausible that this model could be data driven. As numerical models extend to higher dimensionality, higher physical fidelity, and more complex relationships, the advantages of hybrid models will become more and more pronounced.

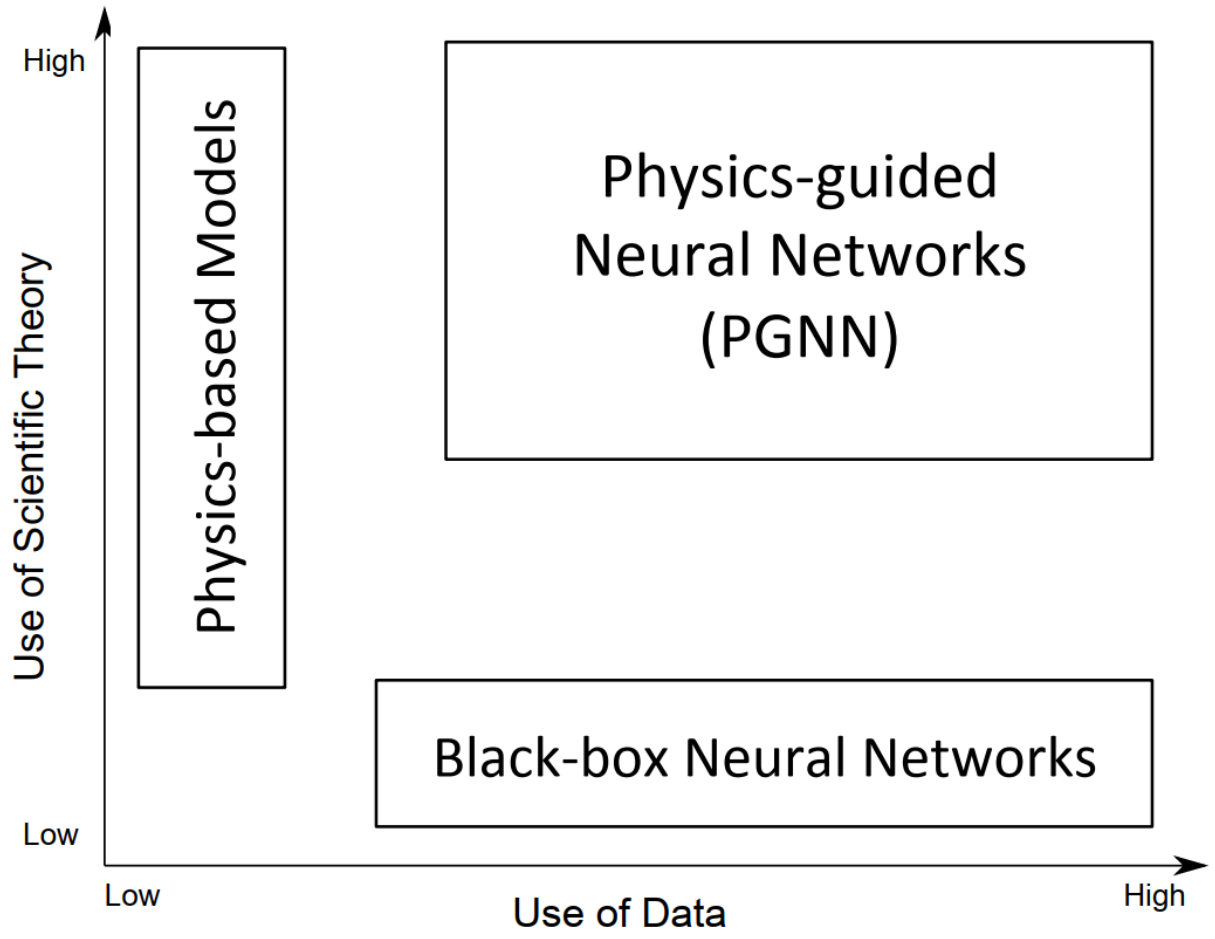


Figure 6.2: First-principles models leverage a large amount of knowledge and are generally fit to very little data, but demonstrate phenomenal extrapolation capabilities. Inversely, black box neural networks require no expert knowledge and require large amounts of data. By blending these two approaches, it may be possible favorably combine the strengths of each. Reproduced from Karpatne et. al.[24] for convenience.

BIBLIOGRAPHY

[1] Meng Guo, Godfrey Sikha, and Ralph E. White. Single-particle model for a lithium-ion cell: Thermal behavior. *Journal of The Electrochemical Society*, 158(2):A122–A132, 2 2011.

[2] Paul W. C. Northrop, Venkatasailanathan Ramadesigan, Sumitava De, and Venkat R. Subramanian. Coordinate transformation, orthogonal collocation, model reformulation and simulation of electrochemical-thermal behavior of lithium-ion battery stacks. *Journal of The Electrochemical Society*, 158(12):A1461–A1477, 1 2011.

[3] Paul W. C. Northrop, Bharatkumar Suthar, Venkatasailanathan Ramadesigan, Shriram Santhanagopalan, Richard D. Braatz, and Venkat R. Subramanian. Efficient simulation and reformulation of lithium-ion battery models for enabling electric transportation. *Journal of The Electrochemical Society*, 161(8):E3149–E3157, 1 2014.

[4] Paul W. C. Northrop, Manan Pathak, Derek Rife, Sumitava De, Shriram Santhanagopalan, and Venkat R. Subramanian. Efficient simulation and model reformulation of two-dimensional electrochemical thermal behavior of lithium-ion batteries. *Journal of The Electrochemical Society*, 162(6):A940–A951, 1 2015.

[5] Matthew B. Pinson and Martin Z. Bazant. Theory of sei formation in rechargeable batteries: Capacity fade, accelerated aging and lifetime prediction. *Journal of The Electrochemical Society*, 160(2):A243–A250, 1 2013.

[6] John Newman and William Tiedemann. Porous-electrode theory with battery applications. *AIChE Journal*, 21(1):25–41, 1 1975.

[7] Venkatasailanathan Ramadesigan, Paul W. C. Northrop, Sumitava De, Shriram Santhanagopalan, Richard D. Braatz, and Venkat R. Subramanian. Modeling and simulation of lithium-ion batteries from a systems engineering perspective. *Journal of The Electrochemical Society*, 159(3):R31–R45, 1 2012.

[8] Manan Pathak, Dayaram Sonawane, Shriram Santhanagopalan, Richard D. Braatz, and Venkat R. Subramanian. (invited) analyzing and minimizing capacity fade through optimal model-based control - theory and experimental validation. *ECS Transactions*, 75(23):51–75, 1 2017.

[9] Qianqian Liu, Chunyu Du, Bin Shen, Pengjian Zuo, Xinqun Cheng, Yulin Ma, Geping Yin, and Yunzhi Gao. Understanding undesirable anode lithium plating issues in lithium-ion batteries. 6(91):88683–88700.

[10] Neal Dawson-Elli, Suryanarayana Kolluri, Kishalay Mitra, and Venkat R. Subramanian. On the Creation of a Chess-AI-Inspired Problem-Specific Optimizer for the Pseudo Two-Dimensional Battery Model Using Neural Networks. *Journal of The Electrochemical Society*, 166(6):A886–A896, January 2019.

[11] Stewart Tansley and Kristin Michele Tolle. *The Fourth Paradigm: Data-intensive Scientific Discovery*. Microsoft Research, 2009.

[12] J. N. Hu, J. J. Hu, H. B. Lin, X. P. Li, C. L. Jiang, X. H. Qiu, and W. S. Li. State-of-charge estimation for battery management system using optimized support vector machine for regression. 269:682–693.

[13] Jiani Du, Zhitao Liu, and Youyi Wang. State of charge estimation for li-ion battery based on model from extreme learning machine. 26:11–19.

[14] Zhihao Wang and Daiming Yang. State-of-charge estimation of lithium iron phosphate battery using extreme learning machine. In *2015 6th International Conference on Power Electronics Systems and Applications (PESA)*, pages 1–5.

[15] Meng-Feng Li, Wen Chen, Hai Wu, and David Gorski. Robust state of charge estimation of lithium-ion batteries via an iterative learning observer. pages 2012–01–0659.

[16] M. A. Hannan, M. S. H. Lipu, A. Hussain, M. H. Saad, and A. Ayob. Neural network approach for estimating state of charge of lithium-ion battery using backtracking search algorithm. 6:10069–10079.

[17] Neal Dawson-Elli, Seong Beom Lee, Manan Pathak, Kishalay Mitra, and Venkat R. Subramanian. Data science approaches for electrochemical engineers: An introduction through surrogate model development for lithium-ion batteries. *Journal of The Electrochemical Society*, 165(2):A1–A15, 1 2018.

[18] Linxia Liao and Felix Kttig. A hybrid framework combining data-driven and model-based methods for system remaining useful life prediction. 44:191–199.

[19] Jing Yang. The remaining useful life estimation of lithium-ion battery based on improved extreme learning machine algorithm. 13:4991–5004.

[20] Kristen A. Severson, Peter M. Attia, Norman Jin, Nicholas Perkins, Benben Jiang, Zi Yang, Michael H. Chen, Muratahan Aykol, Patrick K. Herring, Dimitrios Fraggedakis, Martin Z. Bazant, Stephen J. Harris, William C. Chueh, and Richard D. Braatz. Data-driven prediction of battery cycle life before capacity degradation. *Nature Energy*, 4(5):383, May 2019.

[21] David A. C. Beck, James M. Carothers, Venkat R. Subramanian, and Jim Pfaendtner. Data science: Accelerating innovation and discovery in chemical engineering. *AIChE Journal*, 62(5):1402–1416, 5 2016.

[22] Blake R. Hough, David A. C. Beck, Daniel T. Schwartz, and Jim Pfaendtner. Application of machine learning to pyrolysis reaction networks: Reducing model solution time to enable process optimization. *Computers & Chemical Engineering*, 104:56–63, September 2017.

[23] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354, October 2017.

[24] Anuj Karpatne, William Watkins, Jordan Read, and Vipin Kumar. Physics-guided neural networks (pgnn): An application in lake temperature modeling. *arXiv:1710.11431 [physics, stat]*, 10 2017. arXiv: 1710.11431.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[27] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144 [cs]*, 9 2016.

[28] Claire Monteleoni, Gavin A. Schmidt, Shailesh Saroha, and Eva Asplund. Tracking climate models. *Statistical Analysis and Data Mining*, 4(4):372–392, 8 2011.

[29] 2016 at 3:30am Posted by Jacob Joseph on April 11 and View Blog. How to treat missing values in your data.

[30] Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):7887, 10 2012.

[31] Usama M. Fayyad, Andreas Wierse, and Georges G. Grinstein. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann, 2002.

[32] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 7 1959.

[33] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 3 1990.

[34] Kishalay Mitra and Mahesh Ghivari. Modeling of an industrial wet grinding operation using data-driven techniques. *Computers and Chemical Engineering*, 30(3):508–520, 1 2006.

[35] Aziz Nasridinov, Yangsun Lee, and Young-Ho Park. Decision tree construction on gpu: ubiquitous parallel computing approach. *Computing*, 96(5):403–413, 5 2014.

[36] Miao Liu, Mingjun Wang, Jun Wang, and Duo Li. Comparison of random forest, support vector machine and back propagation neural network for electronic tongue data classification: Application to the recognition of orange beverage and chinese vinegar. *Sensors and Actuators B: Chemical*, 177:970–980, 2 2013.

[37] I. K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, 10 1990.

[38] Jie Liu, Abhinav Saxena, Kai Goebel, Bhaskar Saha, and Wilson Wang. An adaptive recurrent neural network for remaining useful life prediction of lithium-ion batteries. Technical report, 10 2010.

[39] G. Capizzi, F. Bonanno, and G. M. Tina. Recurrent neural network-based modeling and simulation of lead-acid batteries charge and discharge. *IEEE Transactions on Energy Conversion*, 26(2):435–443, 6 2011.

[40] Marc Doyle, Thomas F. Fuller, and John Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of The Electrochemical Society*, 140(6):1526–1533, 6 1993.

[41] Wen-Yeau Chang. The state of charge estimating methods for battery: A review, 2013.

[42] Ali Jokar, Barzin Rajabloo, Martin Dsilets, and Marcel Lacroix. Review of simplified pseudo-two-dimensional models of lithium-ion batteries. *Journal of Power Sources*, 327:44–55, 9 2016.

[43] The julia language.

[44] Steven G. Johnson. *Sobol.jl: generation of Sobol low-discrepancy sequence (LDS) for the Julia language*. 6 2017.

[45] Didrik Nielsen. Tree boosting with xgboost - why does xgboost win "every" machine learning competition? 2016.

[46] Fabian Pedregosa, Gal Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and douard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 10 2011.

[47] Jeff Heaton. An empirical analysis of feature engineering for predictive modeling. *arXiv:1701.07852 [cs]*, 1 2017.

[48] Kazuya Nakagawa, Shinya Suzumura, Masayuki Karasuyama, Koji Tsuda, and Ichiro Takeuchi. Safe feature pruning for sparse high-order interaction models. *arXiv:1506.08002 [stat]*, 6 2015.

[49] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1):245–271, 12 1997.

[50] Blaine Paxton and John Newman. Variable diffusivity in intercalation materials a theoretical approach. *Journal of The Electrochemical Society*, 143(4):1287–1292, 4 1996.

[51] Venkat R. Subramanian, Vinten D. Diwakar, and Deepak Tapriyal. Efficient macro-micro scale coupled modeling of batteries. *Journal of The Electrochemical Society*, 152(10):A2002–A2008, 10 2005.

[52] Sumitava De, Bharatkumar Suthar, Derek Rife, Godfrey Sikha, and Venkat R. Subramanian. Efficient reformulation of solid phase diffusion in electrochemical-mechanical coupled models for lithium-ion batteries: Effect of intercalation induced stresses. *Journal of The Electrochemical Society*, 160(10):A1675–A1683, 1 2013.

[53] Sumitava De, Paul W. C. Northrop, Venkatasailanathan Ramadesigan, and Venkat R. Subramanian. Model-based simultaneous optimization of multiple design parameters forlithium-ion batteries for maximization of energy density. *Journal of Power Sources*, 227:161–170, 4 2013.

[54] Venkatasailanathan Ramadesigan, Vijayasekaran Boovaragavan, J. Carl Pirkle, and Venkat R. Subramanian. Efficient reformulation of solid-phase diffusion in physics-based lithium-ion battery models. *Journal of The Electrochemical Society*, 157(7):A854–A860, 7 2010.

[55] Joel C. Forman, Saeid Bashash, Jeffrey L. Stein, and Hosam K. Fathy. Reduction of an electrochemistry-based li-ion battery model via quasi-linearization and pad approximation. *Journal of The Electrochemical Society*, 158(2):A93–A101, 2 2011.

[56] Manan Pathak, Suryanarayana Kolluri, and Venkat R. Subramanian. Generic model control for lithium-ion batteries. *Journal of The Electrochemical Society*, 164(6):A973–A986, 1 2017.

[57] Long Cai and Ralph E. White. Lithium ion cell modeling using orthogonal collocation on finite elements. *Journal of Power Sources*, 217(Supplement C):248–255, 11 2012.

[58] K. A. Smith, C. D. Rahn, and C. Y. Wang. Model-based electrochemical estimation of lithium-ion batteries. pages 714–719. 2008 IEEE International Conference on Control Applications, 9 2008.

[59] Yi Zeng, Paul Albertus, Reinhardt Klein, Nalin Chaturvedi, Aleksandar Kojic, Martin Z. Bazant, and Jake Christensen. Efficient conservative numerical schemes for 1d nonlinear spherical diffusion equations with applications in battery modeling. *Journal of The Electrochemical Society*, 160(9):A1565–A1571, 1 2013.

[60] Saurabh Y. Joshi, Michael P. Harold, and Vemuri Balakotaiah. Low-dimensional models for real time simulations of catalytic monoliths. *AIChE Journal*, 55(7):1771–1783, 7 2009.

[61] Venkat R. Subramanian and Ralph E. White. New separation of variables method for composite electrodes with galvanostatic boundary conditions. *Journal of Power Sources*, 96(2):385–395, 6 2001.

[62] H. Perez, N. Shahmohammadhamedani, and S. Moura. Enhanced performance of li-ion batteries via modified reference governors and electrochemical models. *IEEE/ASME Transactions on Mechatronics*, 20(4):1511–1520, 8 2015.

[63] Srinivas Soumitri Miriyala, Venkat R. Subramanian, and Kishalay Mitra. Transformann for online optimization of complex industrial processes: Casting process as case study. *European Journal of Operational Research*, 264(1):294–309, 1 2018.

[64] Priyanka Devi Pantula, Srinivas Soumitri Miriyala, and Kishalay Mitra. Kernel: Enabler to build smart surrogates for online optimization and knowledge discovery. *Materials and Manufacturing Processes*, 32(10):1162–1171, 7 2017.

[65] Srinivas Soumitri Miriyala, Prateek Mittal, Saptarshi Majumdar, and Kishalay Mitra. Comparative study of surrogate approaches while optimizing computationally expensive reaction networks. *Chemical Engineering Science*, 140:44–61, 2 2016.

[66] Wei Zhang, Kazuyoshi Itoh, Jun Tanida, and Yoshiki Ichioka. Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied Optics*, 29(32):4790–4797, 11 1990.

[67] Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. page 255258. MIT Press, Cambridge, MA, USA, 1998. [Online; accessed 2019-01-07].

[68] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, page 26432651. Curran Associates, Inc., 2013. [Online; accessed 2019-01-07].

[69] Vivek Kumar Singh, Hatem A. Rashwan, Santiago Romani, Farhan Akram, Nidhi Pandey, Md Mostafa Kamal Sarker, Adel Saleh, Meritexell Arenas, Miguel Arquez, Domenec Puig, and Jordina Torrents-Barrena. Breast tumor segmentation and shape classification in mammograms using generative adversarial and convolutional neural network. *arXiv:1809.01687 [cs]*, 9 2018. arXiv: 1809.01687.

[70] Ronan Collobert and Jason Weston. A unified architecture for natural language pro-

cessing: Deep neural networks with multitask learning. ICML '08, page 160167, New York, NY, USA, 2008. ACM. [Online; accessed 2019-01-07].

[71] Eli David, Nathan S. Netanyahu, and Lior Wolf. Deepchess: End-to-end deep neural network for automatic learning in chess. *arXiv:1711.09667 [cs, stat]*, 9887:88–96, 2016. arXiv: 1711.09667.

[72] Bharath Raj. Data augmentation | how to use deep learning when you have limited data part 2, 4 2018. [Online; accessed 2018-10-24].

[73] Djork-Arn Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv:1511.07289 [cs]*, 11 2015. arXiv: 1511.07289.

[74] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. ICML'10, page 807814, USA, 2010. Omnipress. event-place: Haifa, Israel.

[75] Garrett B. Goh, Nathan O. Hodas, and Abhinav Vishnu. Deep learning for computational chemistry. *arXiv:1701.04503 [physics, stat]*, 1 2017. arXiv: 1701.04503.

[76] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs]*, 12 2014. arXiv: 1412.6980.

[77] I. M Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and Computers in Simulation*, 55(1):271–280, 2 2001.

[78] Andrea Saltelli, Paola Annoni, Ivano Azzini, Francesca Campolongo, Marco Ratto, and Stefano Tarantola. Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer Physics Communications*, 181:259–270, 2010.

[79] Jones E, Oliphant E, and Peterson P. Scipy: Open source scientific tools for python. 2001. [Online; accessed 2018-08-20].

[80] F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, 2 1973.

[81] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. page 6.

[82] Yann Le Cun, Ido Kanter, and Sara A. Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18):2396–2399, 5 1991.

[83] Junhong Lin and Lorenzo Rosasco. Optimal learning for multi-pass stochastic gradient methods. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, page 45564564. Curran Associates, Inc., 2016. [Online; accessed 2019-02-20].

[84] Dieter Kraft and Deutsche Forschungs-und Versuchsanstalt fr Luft-und Raumfahrt (DFVLR) Institut fr Dynamik der Flugsysteme. A software package for sequential quadratic programming. Technical report, Braunschweig, 1988.

[85] Fuchang Gao and Lixing Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Computational Optimization and Applications*, 51(1):259–277, 1 2012.

[86] M. Sarrafzadeh. Department of electrical engineering and computer science. *ACM SIGDA Newsletter*, 20(1):91, 6 1990.

[87] Rainer Storn and Kenneth Price. Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 12 1997.

[88] Hdf5 for python. [Online; accessed 2017-12-06].

[89] Bias variance decomposition. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 100–101. Springer US, Boston, MA, 2010.

[90] Zhenwei Li, James R. Kermode, and Alessandro De Vita. Molecular dynamics with on-the-fly machine learning of quantum-mechanical forces. 114(9):096405.

[91] Venkatesh Botu and Rampi Ramprasad. Adaptive machine learning framework to accelerate ab initio molecular dynamics. 115(16):1074–1083.

[92] Stefan Chmiela, Alexandre Tkatchenko, Huziel E. Sauceda, Igor Poltavsky, Kristof T. Schtt, and Klaus-Robert Mller. Machine learning of accurate energy-conserving molecular force fields. 3(5):e1603015.

[93] Tianyu Gao and John R. Kitchin. Modeling palladium surfaces with density functional theory, neural networks and molecular dynamics. 312:132–140.

[94] Chen Wang, Akshay Tharval, and John R. Kitchin. A density functional theory parameterised neural network model of zirconia. 44(8):623–630.

[95] John R. Kitchin. Machine learning in catalysis. 1(4):230.

[96] Kai-Yuan Cai, Lin Cai, Wei-Dong Wang, Zhou-Yi Yu, and David Zhang. On the neural network approach in software reliability modeling. 58(1):47–62.

[97] H. E. Perez, X. Hu, S. Dey, and S. J. Moura. Optimal charging of li-ion batteries with coupled electro-thermal-aging dynamics. 66(9):7761–7770.

[98] Francois Chollet. Keras.

[99] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal

Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[100] S. van der Walt, S. C. Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. 13(2):22–30.

[101] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. 37(4):143:1–143:14.

[102] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[103] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[104] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31 –39, 2011.

[105] Pierre Celestin Urisanga, Derek Rife, Sumitava De, and Venkat R. Subramanian. Efficient Conservative Reformulation Schemes for Lithium Intercalation. *Journal of The Electrochemical Society*, 162(6):A852–A857, January 2015.

[106] Zhen Liu and Han-Xiong Li. Extreme learning machine based spatiotemporal modeling of lithium-ion battery thermal dynamics. 277:228–238.