

Towards Explainable Neural Networks for Mathematical Programming

by

Fuat Can Beylunioğlu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Management Sciences

Waterloo, Ontario, Canada, 2025

© Fuat Can Beylunioğlu 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Efstratios N. Pistikopoulos Professor Dept. of Chemical Engineering Texas A&M University
Supervisor(s):	P. Robert Duimering, Associate Professor Mehrdad Pirnia, Associate Professor Dept. of Management Science and Engineering, University of Waterloo
Internal Member:	Samir Elhedhli Professor Dept. of Management Science and Engineering University of Waterloo
Internal Member:	Sirisha Rambhatla Assistant Professor Dept. of Management Science and Engineering University of Waterloo
Internal-External Member:	Vijay Ganesh Professor School of Computer Science Georgia Institute of Technology

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Mathematical programming has primarily been a study of numerical optimization where the solution is obtained following a procedure recursively until convergence. Recent applications of Neural Networks (NN) as surrogate models have challenged this view by treating optimization problems as approximating unobserved functions that map input parameters to optimal solutions, which is referred to as *solution functions*. In this thesis, we investigate properties of the solution function, and develop NN based methods for deriving explicit formulae of this mapping. Drawing from the principles of the Universal Approximation Theorem, we investigate the sources of NN approximation errors that arise when training models on dataset consists of input parameters and optimal solutions for previously solved problems.

Drawing on insights from multiparametric programming (MP), which explores piecewise-linear (PWL) properties of quadratic programs with linear constraints (QP), we demonstrate that a NN with ReLU activation functions is a general form of the solution function of QPs. Despite this fact, we show that achieving perfect representations of this function proves difficult when employing black-box models and standard NN training methods. We propose a semi-supervised NN model that learns explicit parameters of each linear segment of the QP solution function from problem coefficients analytically, but is only trained for a few number of model parameters to construct the solution function. Using IEEE DC optimal power flow test sets, we show that this approach accurately learns to represent the QP solution function with minimal error, ensuring optimal solutions for any right-hand side (RHS) parameters.

Building on the semi-supervised model, we further propose a closed-form NN model and a learning via discovery algorithm that learns an exact representation of the solution function without training, by applying algebraic operations to derive all model weights from problem coefficients without the need for training data. Our proposed learning algorithm begins with an optimal solution to initiate the NN model, then discovers each linear segment of the PWL solution function and expands the model parameters iteratively without using solvers. The proposed procedure yields a NN model that acts as the closed-form solution to QP with linear constraints, ensuring that optimal solutions for the discovered regions are both feasible and optimal. It is shown that the proposed CF-NN model offer significant advantages. The model takes seconds to learn the solution function and can seamlessly generalize outside the training distribution. Results on IEEE DC-OPF test cases significantly outperforms traditional deep NN training methods, achieving optimal solutions with near-zero calculation errors in all discovered critical regions.

In this thesis, we primarily focused on methodologies for uncertain parameters added to the right hand side of equality constraints. Further work is needed to examine the effect of uncertainty on inequality constraints, improve the scalability of the approach to larger systems by ensuring the learning algorithm to visit all critical regions of the feasible domain, reducing floating point precision errors, and addressing the issues related to degeneracy.

Acknowledgements

Void is what PhD felt like here. It is made of some principles glorifying the traditional. These are the ingredients of a perfect thesis.

I wrote an imperfect one and I insisted on it. I was attached to something that does not belong here. I was totally disconnected from this environment to write this thesis, acting irrationally, rejecting the treatment, and I finished my thesis. But I would not survive here if I didn't have Rob guiding me.

First and foremost, I have to thank my supervisors, Rob Duimering and Mehrdad Pirnia for their patience and support. I am sure it was hard for them as hell and am truly sorry for it. I also want to thank my thesis committee, Samir Elhedhli, Sirisha Rambhatla and Vijay Ganesh for their time and critique for a better work. I feel I have to give special thanks to Efstratios Pistikopoulos for participating in my thesis committee and his comments during my defense. I cannot imagine a better ending to my PhD studies than receiving the approval and praise from the founder of my research field. Finally, I cannot thank Houra Mahmoudzadeh enough for seeing the true value of my work when I needed it the most.

During this time, I met many Dostoevsky-esque characters that do not belong in here either. I will not forget any of our chats with Selçuk Onay that switch momentarily between levels of intellect but always end at wishing no one is hearing us. The timing of the coffee pods he brought reminded me that God has mysterious ways. I want to thank Safa Erenay for a lot of things, from his unconditional support in my time of need and his timeless academic advises, but his coffees reminded me God has mysterious ways as well.

I was once asked what the best thing in PhD life is. There is nothing. It is just bad. Living in the Costco-Walmart-Basement triangle is nothing to glorify. But I cannot exchange any of my time I spent in this unholy trinity thanks to the true friendships I made. I don't know what happiness is, but I think it has something to do with unapologetic trash talks I made with Atahan in secluded spots deep in provincial parks, late-night philosophical coffee chats with Onur, unhealthy ice cream retreats with Aniket and chats about everything but the course with Stoyan in our TA meetings. Speaking of happiness, I cannot imagine it without Shima anymore. I am so lucky to know you.

Canada is not for beginners. It took me four years to understand the people. Most are highly conservative with their values and look for some subtle social cues to decide whether you are worth to talk to. If not, they build walls to socially distance themselves from those freaks. How to know those social cues? I recommend skating for beginners and canoe trips for the advanced. So I want to thank all other Canadians and non-Canadians who didn't build those walls. In particular, I want to thank Goossen family for the blueberry hot sauce gift and the entire Mikitka family for homemade cookies. I am so thankful to Matt, Kevin, Hassan, Kyle and David for all of invaluable chats and Jeff for humble activities, like trying different guitars for hours in a desolate Long and McQuade and not buying any. I couldn't survive in the hip-hop-loving City of Waterloo without Lil Noir Movie club and its frequent members, Mario, Peter and Yuan. I won't forget their faces after seeing Old Boy together. Finally, I want to thank Bohdan for being with me in my darkest times. I don't want to say more, it was just bleak. My family goes without saying.

That's it I guess. I did my part.

Dedication

To those that disappear...

Table of Contents

Examining Committee	ii
Author’s Declaration	iii
Abstract	iv
Acknowledgements	vi
Dedication	viii
List of Figures	xiv
List of Tables	xvii
1 Introduction	1
1.1 Motivation	5
1.2 Contributions	7

1.3	Thesis Outline	9
1.4	Literature Review	10
1.4.1	Machine Learning for Optimization	11
1.4.2	Universal Approximation Theorem	12
1.4.3	Physics Informed Neural Networks	14
1.4.4	Multiparametric Programming	15
1.4.5	Integration of DNNs with Multiparametric Programs	18
1.4.6	Machine Learning in Power System Research	19
2	Background	23
2.1	Optimality Conditions for Constrained Optimization	24
2.1.1	Sensitivity Analysis	26
2.2	Multiparametric QP with Linear Constraints	27
2.2.1	Solution Function of Inequality Constrained mp-QP	29
2.2.2	Global Properties of mp-QP Solutions	31
2.3	Neural Networks	35
2.3.1	Neural Networks as Piecewise Linear Functions	37
2.4	Operational Models for Power Systems	41
2.4.1	Optimal Power Flow Problem	41

2.4.2	DC OPF Problem	43
2.5	Summary of the Chapter	45
3	Solver-Free Data Generation Approach for Training Models for Nonlinear Problems	47
3.1	Methodology	50
3.1.1	Power Flow Problem	51
3.1.2	Generator-Predictor Approach for Data Construction	52
3.1.3	Comparison of Newton Raphson-based Training and Solver-Free Approach	54
3.1.4	Dataset size and Model Performance	57
3.1.5	Maximum Model Prediction Error on Nonlinear Functions	58
3.2	Numerical Results	67
3.2.1	Comparison of NN and Solver-Based Solutions	67
3.2.2	Maximum Iteration on Optimization Performance	71
3.3	Chapter Summary	72
4	Semi-Supervised Neural Network Model For Multiparametric Quadratic Programming	73
4.1	Introduction	73
4.1.1	The Tradeoff Between Dataset Size and Model Complexity	75

4.2	A General Framework to Solve QP Using NN	78
4.2.1	Predicting Active Constraints and Shadow Prices	80
4.2.2	Filtering Active Constraints and Data Generation	83
4.3	Numerical Results	85
4.3.1	Test Results	87
4.3.2	Uncertainty	92
4.4	Scalability and Limitations	95
4.5	Chapter Summary	97
5	A Neural Network Framework for Discovering Closed-Form Solutions to Quadratic Programs with Linear Constraints	99
5.1	Multiparametric QP with Linear Constraints	102
5.2	Methodology	103
5.2.1	The Model	103
5.2.2	Modeling PWL Changes	105
5.2.3	Shadow Price Model: $f_{\mu}(z)$	107
5.3	Learning via Discovery Algorithm	110
5.3.1	Illustration of the Algorithm on a 2D Example	114
5.4	Numerical Results	118
5.4.1	Test Results	119

5.4.2	Speed Comparison	126
5.4.3	Line Limits	127
5.4.4	Test Results with Line Limits	131
5.5	Chapter Summary	134
6	Conclusion	136
6.1	Discussion and Future Work	138
	References	145

List of Figures

2.1	Sensitivity of μ against θ_1	33
2.2	Illustration of Shallow and Deep NNs	37
2.3	Illustration of NN as a Piecewise Linear Function	40
3.1	Generator-Predictor model to train solver	54
3.2	Comparison of two approaches to train a $NN(n_h = 1, d_h = 100)$ with varying d_d	56
3.3	Average test RMSE of $NN(ReLU, n_h = 1, d_h = 100)$ on d_p dimensional quadratic function	58
3.4	Two NN approximators to simplified PF equations in 3.3	59
3.5	Illustration of $f_{sin}(x)$ that minimize test MSE=0 against $\sin(x)$	61
3.6	Maximum error estimates for P^{inj} and Q^{inj}	67
3.7	Maximum iteration on solver performance	71

4.1	The tradeoff between dataset size and model complexity in estimating arbitrary PWL functions	76
4.2	The QP Solver Model Architecture	80
4.3	Sensitivity of μ against θ_1	82
4.4	NN Model to Predict μ^*	82
4.5	Submodel predicting μ^*	88
4.6	SSNN Architecture	88
4.7	Calculation Speed Comparison of SSNN and Gurobi (Solver) over 1000 test problems	93
4.8	Hourly distribution of optimal primal and dual prices	94
4.9	Illustration of Model Predictions	95
5.1	The QP Solver Model Architecture	104
5.2	Illustration of slope update rules on an example PWL function.	105
5.3	$f_\mu(\theta)$ model	108
5.4	Critical regions and connectivity graph for an mp-QP with two parameters (Adapted from [94, p. 59])	115
5.5	Illustration of slope update rules on an example PWL function.	116
5.6	Comparison of power injection violations of two methods in 57-bus system	125
5.7	Connectivity of critical regions for problems without and with line limits	129

5.8	Illustration of dataset distribution and the search pattern in Algorithm 4 .	129
5.9	Connectivity graph between critical regions	134
6.1	The issue arises in case of degeneracy	139

List of Tables

3.1	Maximum error estimates for $Q_i^g - \hat{Q}_i^g$, averaged for all i	66
3.2	Comparison of our model against alternative approaches on test systems .	70
3.3	Time (s) to calculate 4000 test set predictions for each model	71
4.1	KKT Conditions and corresponding formulae	89
4.2	Mean Squared KKT Errors on test set generated for a local demand range	90
4.3	Mean Squared KKT Errors on test sets generated for the extreme demand characteristics	91
5.1	Mean Squared KKT Errors on the test sets with Local Perturbations . . .	121
5.2	Mean Squared KKT Errors on the test set with Extreme Characteristics .	122
5.3	$C(\mathbf{P}^g) - \hat{C}(\mathbf{P}^g)$ calculated for datasets constructed via local perturbations and reflecting extreme characteristics	123
5.4	Performance Comparison of Closed-form and Gurobi on 57-bus system to calculate solutions to long-term planning problem	127

5.5	Mean Squared KKT Errors on the test set with Local Perturbations	133
-----	--	-----

Chapter 1

Introduction

In the last decade, there has been an increasing interest in using Neural Networks (NN) to predict the solutions to system of partial differential equations or to act as surrogate models to linear and nonlinear optimization problems that are typically solved with iterative techniques. Such approaches have been applied in areas such as physics [96], energy management [38], chemistry [58], and control theory [56]. Most applications treat the NN model as a black-box and employ standard training methods using datasets of input-output pairs, representing particular values of system parameters and corresponding optimal solutions, ignoring the mathematical structure of the underlying function. Such modeling approaches necessitate large, computationally expensive training datasets to achieve satisfactory performance, yet cannot guarantee the feasibility or optimality of the results.

Effectively, training NNs to solve optimization problems for various instances of input variables and parameters can be interpreted as solving multiparametric programs, which

are defined as optimization problems with uncertain parameters such as right hand side vectors of constraints and cost coefficients of the objective function. Multiparametric programming (MP) has been studied in parallel to surrogate NN models for optimization prior to the increasing popularity of NN research. The literature seeks to characterize the solutions of the optimization problem as functions of the problem parameters. Using the characteristics of MP, efficient algorithms can be developed to find optimal solutions to problems that are typically solved repeatedly, such as model predictive control [56] and power management applications [53].

Although there are notable overlaps between multiparametric programming and NN in the optimization literature, it was not until the 2020s when researchers proposed merging the two fields. Recent work has attempted to integrate multiparametric programming with DNNs, based on recognition of their similar underlying mathematical characteristics [56, 57]. Specifically, for multiparametric Linear Programming (mp-LP) and multiparametric Quadratic Programming (mp-QP) with linear constraints, the optimal primal and dual solutions are piecewise linear (PWL) functions of the parameters [94]. Similarly, a NN with ReLU activations function is a general form of PWL function with trainable weights and biases [79]. In theory, therefore, for any mp-LP or mp-QP, one should be able to find an optimal set of weights and biases such that a NN can exactly represent the corresponding solution function [56].

In practice, however, DNN applications of multiparametric programming face significant training challenges to guarantee global solution optimality and feasibility. First, obtaining the weights and biases for a NN that generates minimal loss is difficult due to the nonconvexity of the training loss function, whereby multiple sets of weights and biases

can result in different local minima for a given dataset. Second, NN prediction accuracy is highly dependent on the training set, making it difficult to generalize predictions outside of the training distribution. Overcoming these problems requires sampling very large datasets that cover all feasible regions of the optimization problem parameters, which is impractical for high dimensional problems. For example, constructing a dataset that includes only boundary points for an n dimensional parameter vector requires sampling 2^n points, which would still be insufficient because additional interior points are needed within each region [14].

There are attempts to overcome the training and generalizability challenges by incorporating the physical properties of the underlying problem within the training of DNNs with an additional term in the loss function that incorporates domain specific equations to be satisfied. This physics-informed loss function acts as a regularization term that restricts the parameter space of the NN model to discard any nonrealistic solutions. In other words, encoding information about the mathematical structure of the problem into the learning algorithm amplifies the information content of the data that the algorithm sees, enabling it to quickly steer itself towards the right solution and generalize well even when only a few training examples are available.

One of the widely studied applications of Machine Learning (ML) has been in the domain of power systems management, where large scale optimization models must be repeatedly solved for different parameters. Various ML approaches are exploited to speed up the operations of optimization models to solve such problems by using statistical learning models to predict active constraints [28], two stage solution approaches handling optimality and feasibility separately [118], reinforcement learning to speed up the convergence of

MILP [112], PINN for power applications [81] and more. Many power system applications such as the optimal power flow (OPF) problem, transmission network expansion problem (TNEP) [19], and microgrid planning [110] and operation problems involve optimizing the cost of energy production to satisfy a given demand, while obeying electricity distribution laws that are defined as equality constraints, and generator and line limits defined as inequality constraints.

Despite evidence suggesting significant improvements in the solution time of such problems, scholars and practitioners are skeptical about the usage of NN due to the unexplainability of the obtained solution and lack of optimality and feasibility guarantees. Attempts to develop models that address these concerns include additional layers for repairing predicted solutions to satisfy their feasibility [22], incorporating verification techniques to find the input that would generate the worst-case scenarios of the solution [81], and designing model architectures reflected by the problem structure [39]. However, these approaches still act as black-box models that do not clearly explain how NNs construct solutions.

The potential of neural networks to approximate a mathematical function has been formally described by the Universal Approximation Theorem (UAT), which suggests that there exists a neural network that can approximate any continuous function with less than a desired error for any input from a compact set [50]. Given this strong premise, it must be possible to train a neural network that can approximate the solution of any multiparametric program within an error threshold and thus guarantee the optimality and feasibility of the solutions for all feasible input parameters. However, existing studies that use NNs as surrogate models have not been able to deliver on this promise [80, 81, 71]. It remains unclear why it is difficult for machine learning approaches, which are based on training

a Deep NN to solve multiparametric programming, to achieve outstanding performance goals.

1.1 Motivation

Considering the shortcomings of previous research, this study aims to address the limitations of using NN-based surrogate models in optimization problems, where traditional approaches often fail to provide optimal and feasible solutions. One possible solution could be developing predictive models that incorporate uncertainty, such as Bayesian NNs, which can produce distributions of predictions. However, when there is a unique solution of a constrained optimization problem for every possible right-hand-side parameters, the output of optimization problems can be obtained using a deterministic continuous function [36], hence eliminating the need for models incorporating uncertainty. Therefore, since the UAT suggests that NNs are capable of approximating any continuous function, research is needed to explain the training of NN models for optimization problems.

It can be inferred from the UAT that the NN training set must include all boundary points and be dense enough for better accuracy and the domain of the model should have certain characteristics. A key challenge in using NNs to approximate the solution of optimization problems is their inability to capture the complex relationships between inputs and outputs when the problem dimensionality increases, making it impractical to generate training data representing all possible scenarios. Furthermore, the relationship between model size, dataset size, and performance remains unclear, necessitating further research.

In our preliminary experiments, we observed that when the target function is linear or PWL, NNs could provide results with very high accuracy, despite smaller training sets. This observation motivated this research to use DNNs to solve mp-QP problems with linear constraints, whose solutions form a PWL function. While existing literature highlights similarities between the solutions of mp-QP and NN properties, more research is needed to develop effective methods for training NNs to represent the solution of optimization problems.

The basic sensitivity theorem [36] provides valuable insight into the relationship between optimal solutions and changes in input parameters. Specifically, it suggests that the slope of the solution function is constant within each critical region defined by active constraints. By leveraging this information, it is possible to derive some parameters that the NN is expected to discover from the data, such as the coefficients of each linear surface at each critical region corresponding to a part of the solution function. Therefore, this research aims to integrate such coefficients into the NN model and use training to create exact NN solvers.

To advance our understanding of how to create NNs to solve optimization problems, further research is necessary to explore methods for adapting these models to capture complex relationships between problem parameters and optimal solutions. Specifically, we need to investigate ways to identify optimal model architectures, develop techniques for robust training, and establish a deeper understanding of the relationship between model size, dataset size, and performance.

1.2 Contributions

This thesis contributes to the literature by addressing the shortcomings in developing trustworthy NN models for solving nonlinear programs (NLPs). We examine the generalizability, scalability and explainability of NNs predictions to optimization problems and make connections between NN operations and optimization problem solution procedures to integrate DNNs with MP solvers. We examine NN models for optimization in two cases, a general class of continuous nonlinear convex problems, whose solution function is a nonlinear function that has to be approximated, and quadratic program (QP) with linear constraints whose solution function is PWL, which can be represented exactly by a NN model. In both cases the problem is assumed to have a unique solution for any problem parameter, i.e. the solutions form a deterministic continuous function.

The contributions of this thesis are therefore as follows:

- **Proposing a Scalable Data Generator for Nonlinear Programs:** We examine implications of the UAT in the context of surrogate NN models for Nonlinear Problems (NLP) in Chapter 3. The analysis provides insight into NN capabilities in approximating the solutions to NLP and introduces a cost-efficient data generator without using solvers that can sample very large and high quality datasets. Our data generator approach samples datasets of desired sizes directly from the problem equations without using solvers in a loop, inspired by their underlying physics. This approach is equivalent to dataset construction using quasi Newton Raphson solvers but avoids the inaccuracies that stem from the solver operations. The resulting DNN model, when trained using datasets constructed using our approach, outperforms

alternative approaches including the off-the-shelf nonlinear solvers.

- **Proposing a Semi-Supervised NN architecture with High Generalizability:** Motivated by the overlap between functional forms of mp-QP solutions and DNN models, Chapter 4 illustrates the intractability of representing mp-QP solutions exactly with DNNs using the existing approaches. To overcome the difficulties, we propose a novel architecture to solve mp-QP problems for finding optimal NN parameters that would align with mp-QP solutions and generalize outside the training distribution. The proposed method derives a large proportion of NN model parameters directly from the optimization problem coefficients and requires training on only a few parameters. We also propose an efficient procedure using solvers only to discover critical regions of the mp-QP solutions. To the best of our knowledge, this approach is the first to truly inject the physical properties of the solution into the NN model weights, rather than incorporating them as a part of training loss function as in PINNs.
- **Proposing a Method for Deriving Model Parameters from the Problem Coefficients:** The semi-supervised model suffers from a long and unstable training process that limits scalability to larger problems. Chapter 5 addresses this issue by proposing an unsupervised model architecture and an algorithm to derive all model parameters from the problem coefficients. The proposed algorithm starts with a NN that can calculate optimal solutions for one critical region and derives a new model when a new region is discovered. This chapter contributes to the literature by proposing standalone NN learning approach that is independent from using solvers

and can provide global optimal solutions. Our model learns to solve optimization problem without trained on labeled datasets.

1.3 Thesis Outline

This thesis is organized as follows. In the rest of this chapter, a review of the recent literature is presented. In Chapter 2, a general class of convex optimization problems, and its necessary and sufficient optimality conditions are outlined. Then the chapter discusses multiparametric problem, its global solution properties and the notion of critical regions, detailing the PWL nature of the global solution function that is essential to our thesis. Later in the chapter, a Deep Neural Network model will be formulated and the PWL property of a NN with ReLU activation function will be presented. The chapter will conclude with the background on operational models for power system problems, which details the power flow equations that are central to power systems research, the optimal power flow (OPF) problem and its linearization that will be studied in this thesis.

Chapter 3 details how our proposed data generation approach constructs accurate input-output pairs without using solvers in a loop by sampling the dataset directly from power flow equations. The chapter starts with a theoretical analysis of NN prediction error when the NN is trained on a dataset with a certain density, i.e., sampling data points from a uniform grid constructed on the feasible domain. Then a cost efficient data generation approach will be presented and compared with the Newton-Raphson method. The chapter presents the performance results of a NN trained by constructing dataset with our approach, compared against the conventional psuedo Newton-Raphson approaches.

In Chapter 4, a novel semi-supervised NN model to solve mp-QP with linear constraints is proposed. This chapter starts by discussing the intractability of training a NN model to solve mp-QP problem and the lack of generalizability power outside the training distribution. We analyze the ability of NNs to represent an arbitrary PWL function via conventional training procedures. Then we propose a NN model and a method to derive a large proportion of NN weights from the problem coefficients. We compare our approach to a commercial solver and conventionally trained NN model.

In Chapter 5, an unsupervised NN model and a procedure to derive model parameters is presented. This model advances the semi-supervised model with a NN design that can activate and deactivate a neuron, allowing for only the correct solution corresponding to every critical region. Our method is accompanied by a discovery algorithm that updates the model coefficients when a new critical region is discovered. Finally, in the Conclusion, we discuss the implications of our models, their potential applications and limitations.

1.4 Literature Review

Our research lies at the intersection of different domains of research, between multiparametric programming, ML for optimization to Physics Informed NNs. In this section, a review of the relevant literature will be presented.

1.4.1 Machine Learning for Optimization

Machine learning research for optimization can broadly fall into two main categories, studies using ML to improve optimization and training ML models for end-to-end optimization learning. The first group uses ML for heuristics to improve solution time and efficiency of existing optimization algorithms, such as using NNs for learning branching rules for MILP problems [45], reinforcement learning to predict optimal cuts for MILP problems [106], and optimal combinations for branching rules [6] for a general class of problems. In addition to the first group, researchers also employed statistical learning approaches as heuristics for combinatorial optimization, such as for learning active constraints thereby reducing the problem dimensions, providing more accurate predictions and quicker solutions [76, 82, 28, 23]. ML also offers significant computational advantages for column generation [102] as a heuristic model to predict the reduced cost of columns without directly solving the pricing subproblems or finding paths that would lead to less iterations in the master problem [25].

The second approach, end-to-end optimization learning, is an established field with various applications in engineering. The approaches leverage DNNs as universal function approximators to estimate the nonlinear relationship between the problem parameters and the optimal solutions, which has found success in power management [81, 38, 71], control [56] and chemistry (see [57] for a review). Despite their advantages as nonlinear solvers, guaranteeing feasibility and optimality is challenging, and finding the best model parameters from a large parameter space is data-greedy with significant computational training burdens.

Guaranteeing optimality and feasibility of solutions for all inputs poses a challenge due to the nonlinearity of the solution function. To improve feasibility, [118] and [22] proposed two-stage models that first predict optimal solutions with a NN and then post-process the solutions to enforce feasibility. [71] proposed a NN architecture that enforces optimal dispatch generator limits using a sigmoid activation function in the output layer. [38] processed optimal solutions through sequentially connected sub-networks reflecting problem characteristics. Instead of predicting the optimal solutions directly, in [23] the shadow prices of inequality constraints were predicted only. However, as the model generates noisy predictions, the output was only used for predicting whether the shadow price is positive, to indicate the set of active constraints to reduce optimization to solving a linear system of equations via matrix algebra.

Researchers have developed various frameworks that exploit end-to-end learning using Lagrangian duality approaches to predict approximate solutions for constrained problems, including continuous Nonlinear Programs [38, 109] and constrained prediction tasks such as transprecision computing and fair classification [39]. Other approaches involve injecting information about constraints from targeted feasible solutions to reduce constraint violation [29] or applying autodifferentiation techniques on KKT conditions to calculate exact gradients for training NN to solve QPs [2, 31] and LPs [113].

1.4.2 Universal Approximation Theorem

The Universal Approximation Theorem (UAT) is central to the proposed research. Briefly, the theorem states that there exists a number of hidden neurons that guarantees the NN

to fit any continuous function with less than a desired error. Originally the theorem was proposed by Cybenko [26] for NNs with sigmoid activation function. It was later shown by Hornik [50] that the approximation capability is not specific to the activation function but it is due to the multilayer (at least one hidden layer) architecture of feed forward neural networks. The original study and its extensions are now referred to as arbitrary width proofs for a fixed NN depth, where width refers to the number of hidden neurons per layer and depth the number of hidden layers. The complementary proof for the arbitrary depth case and a fixed width was also provided in [72].

Although the original proof explains the approximation capability of NNs, it does not provide a relationship between the number of hidden neurons/layers and the approximation error. Researchers studied this relationship and showed that the width can be extremely large for certain functions when the network is shallow, but could be very small in case of a deep neural network [107, 33]. Similarly others asked whether there is a critical width that guarantees an accurate approximation. Lu et al. [72] showed that if hidden layer size (d_h) is less than input layer size (d_x), then the NN cannot approximate Lebesgue integrable functions. Hanin and Selke [46] showed that ReLU networks $d_h = d_x$ cannot approximate a function $G(x) : \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_y}$, where d_y is the output layer size, but it is possible if $d_h = d_x + d_y$. Similar proofs for different activation functions exist in the literature. See [92] for a detailed review.

The UAT version proposed by Hornik [50] forms the theoretical basis needed for this research proposal. Our analysis targets real valued continuous functions, such as the Jacobian of the AC-OPF problem that satisfies $d_x = d_y$. UAT will be further discussed in Chapter 2.

1.4.3 Physics Informed Neural Networks

The studies that first proposed using NN for solving system of equations go back to 1990s when training NNs with high computation power was not available. Early studies [66, 64, 65, 75] leveraged the approximation power of NNs in order to solve ordinary or partial differential equations. Later researchers applied the model to problem specific domains such as Schrödinger equation [105], atomic and molecular physics [16] and other fields in sciences [88, 93]. Despite of their advantages as nonlinear solvers, training a NN to model a system of equations or to solve it is a challenging task because finding the best model parameters from a large parameter space is data-greedy and computationally challenging for optimization algorithms. Although computational challenges are eased with proliferation of high-performance computers and efficient NN libraries, traditional methods require enormous datasets to train well-performing networks.

Physics Informed Neural Network (PINN) was proposed to overcome the dataset requirements, and provide more accurate predictions with less violations and more generalizability power. The model was first introduced by Raissi et al. [96] although it has its roots in ML literature [98, 97] and partially in early NN studies in 90s [95, 64]. Like the above mentioned NNs, PINNs leverage the universal function approximator feature of NN to estimate or solve nonlinear partial differential equations without linearization or simplification. Unlike regular NNs, it additionally leverages the partial derivatives of the underlying problem function drawn from the knowledge domain by implementing it in the loss function as a form of regularization term to reduce the problem space and thus reduce the dependency of NN on large datasets.

PINNs have been applied to different physics problems, including fluid dynamics [96, 74], heat transfer [17] and Cardiac Activation Mapping [100]. Also, researchers applied the same concept to architectures different from Deep Feed Forward NN, to convolutional neural network and autoencoders [68] and recurrent neural networks [54]. The concept was introduced to power systems by Misyris et al. [77] and later applications include fault detection [68], wind power estimation [91], nonlinear system identification [103], stability analysis [104], and capturing power system dynamics [78]. PINN has recently become a popular field of research with many applications and alternative ways to solve partial differential equations that are beyond the scope of our study. The reader can refer to [14] for a deeper review.

1.4.4 Multiparametric Programming

Multiparametric programming approaches go back as early as 1952 when Markowitz derived his problem solutions as a function of a parameter, θ which intrigued others to study the problem in depth. Parametric programming was first coined by Manne [73], which was later generalized to multidimensional parameter vectors. One of the earliest multiparametric formulations of LP was proposed in 1972 in [42] by providing an algorithm to solve mp-LP problems.

Multiparametric programming became more popular in the 2000s when researchers showed that model predictive control laws can be expressed as multiparametric programming solutions [9], specifically in the form of a piecewise linear function of the parameter set. Studies proposed a geometric interpretation that considers the critical regions of

mp-LP solutions as polyhedra, where the parameter space is explored by moving the parameter set among critical regions [15]. However, it was later shown that this approach does not guarantee discovery of all critical regions [108]. The geometric approach was later generalized to solve mp-QP by Bemporad et al. [10] as it shares similar characteristics with mp-LP. In contrast to mp-LP, which relies on the simplex algorithm, multiparametric QP was theorized using Basic Sensitivity Theorem proposed by Fiacco [36], and it was shown that solutions to mp-QP are PWL functions and the optimal objective function is a piecewise quadratic function of the parameter set [9, 94].

Many multiparametric problems are typically defined with uncertain parameters on the right-hand-side (RHS) of the constraints, but there is research with left-hand-side uncertainty, i.e., varying constraint coefficients as well. This multiparametric setting is particularly challenging due to bilinear terms, which led researchers to relax the bilinear terms with McCormick based approaches [70, 115]. Khalilpour and Karimi [60] simplified the problem by allowing the univariate parametric case, which simplifies the partitioning of critical regions.

The geometric approach involves finding critical regions by solving the problem for an initial parameter set, θ_0 , identifying active sets and obtaining the parametric solution corresponding to the critical region where the solution is valid. Subsequently, the remaining critical regions are explored by moving outside the initial critical region. The original study by [9] proposes constraint reversal by successively reversing signs of each constraint, i.e., from $A_i x \leq b_i$ to $A_i x > b_i$. However, this approach creates too many artificial cuts that creates scalability challenges for larger problems. Two main modifications were proposed to overcome scalability. First is an active set inference approach [108] that adds or drops

one constraint from the active constraint set to move to an adjacent region. The second is variable step size approach [7] that chooses the facets defining the critical region, finding its centre and moving outside to discover an adjacent critical region. However, both methods fail to explore the entire feasible parameter set when facet-to-facet property does not hold.

The critical regions of an mp-QP problem are defined by a unique corresponding set of active constraints. Given this definition, the feasible space can be exhaustively explored by enumerating all combinations of constraints and removing infeasible combinations. As the number of constraint combinations grow exponentially, this approach does not scale up to higher dimensional problems. The number of potential combinations can be reduced when an infeasible set is discovered [44] since a power of an infeasible set is also infeasible. This motivated others to propose branch and bound like algorithms to discover the critical regions [35, 34].

The mixed integer versions of LP and QP problems can be viewed as combinations of multiple mp-LP and mp-QP problems. Therefore, strategies proposed to solve mp-MILP and mp-MIQP problems include enumerating all combinations of integer variables, solving the corresponding LP or QP problems, and combining solutions by comparison. To make the solution tractable, researchers again proposed branch and bound [1] and decomposition strategies [55].

Researchers found that mp-MIQP solutions have different characteristics than mp-MILP, mainly on the properties of the critical regions of the solutions. Unlike mp-MILP solutions whose critical regions are polytopes, obtaining optimal solutions by comparing multiple mp-QP solutions can result in quadratically constrained critical regions. Main-

taining critical regions in the form of polyhedra offers computational advantages, which motivated researchers to apply McCormick relaxations for linearizing the objective function or critical regions [85].

Other applications of multiparametric programming to optimization including multiparametric nonlinear programming are documented in the literature. These studies are outside the scope of our study and thus will not be detailed. The reader can refer to [84] for further review.

1.4.5 Integration of DNNs with Multiparametric Programs

Examining mathematical properties of mp-LP and mp-QP problems under a multiparametric setting led researchers to recognize their similarity to the PWL characteristics of DNNs. Based on this recognition, researchers formulated DNNs as MILP problem and proposed efficient formulation methods [57]. MILP formulations of DNNs were used to generate adversarial examples, such as adversarial attacks [40] and finding worst case guarantees for DNNs trained to solve OPF problems [80, 81].

Although analytical strategies were proposed to solve mp-QP problems [87], the functional relationship between inputs and optimal solutions is not trivial. Karg and Lucia [56] examined the functional similarities between DNNs and model predictive control (MPC) laws, and trained a model to estimate MPC solutions for a time variant system. Similarly, Huo et al. [53] proposed an approach to integrate MPC laws of a MILP problem for a microgrid system. Wu et al. [116] proposed a procedure to train DNNs for LP and their models can reduce primal variable prediction error below 1E-2 on 4 dimensional optimiza-

tion examples. However, none of these works compare optimality and feasibility between their models and traditional solver benchmarks or DNN based approaches. This dissertation proposes a custom NN architecture that closely aligns with the PWL structure of the global solution function to improve accuracy and generalizability of mp-QP predictions.

1.4.6 Machine Learning in Power System Research

Early studies in the OPF literature are concerned with using ML to improve execution time of solvers. In a series of studies (e.g. [76, 28]), researchers observed that it is possible to reduce the dimensions of the problem by predicting the active constraints. Specifically, the solution must always satisfy the equality constraints and predicting the binding inequality constraints reduces the size of the problem space. For this purpose, different types of classifiers were trained to find the solution faster [76, 28, 82]. Others [5] proposed faster solutions to the AC-OPF problem by removing inactive constraints to modify the feasibility space and downsize the problem.

Due to the highly nonlinear nature of the constraints, Newton-Raphson (NR) methods suffer from unstable convergence points, so finding the global optimal solution to AC-OPF with NR depends on choosing appropriate parameters, such as learning rate and initial solution. In addition, NR methods are computationally costly due to the inversion of the Jacobian term. This led researchers to reduce the cost of computing the Jacobian term and its inverse, e.g., training a function with synchronized voltage and current phasor data to approximate the term's output [24], or training a neural network that can imitate the term [48].

Rather than replacing any particular term, Baker [3] replaces all parameter updates by training a Recurrent Neural Network (RNN) on a dataset consisting of Newton Raphson iterations of previously solved problems. The method provides faster solutions than regular MIPS solvers with small error. For the PEGASE 1354 test system [41], it yields results 22 times faster than a baseline solver, with 1.16% mean percentage error from optimal solution and 9.95 per unit average constraint violation. However, the optimality and feasibility errors increase as system size increases.

Instead of boosting NR solvers, researchers discovered that NNs can also be trained to predict the optimal solution directly without any iteration. One study [21] reported high performance using a three layered Deep Neural Network (DNN) trained on input-output pairs consisting of active/reactive power demand and their corresponding solution variables. As in Baker’s study, the error grows with system size and it does not guarantee AC-feasibility. Nevertheless, it shows the potential of NNs in providing practical solutions to AC-OPF for use by system operators.

Recent studies addressed the problem of feasibility. Zamzam and Baker [118] proposed an algorithm that employs a ML model to predict a near optimal solution and then adjusts it to ensure feasibility. Biagioni et al. [13] proposed a ML aided Alternating Direction Method of Multipliers (ADMM) framework, in which a gated recurrent unit (GRU) network predicts the optimal dual parameters of the constrained problem and then continues with ADMM iterations to find the optimal solution. Li et al. [69] employed a Generative Adversarial Network (GAN) to train an DC-OPF solution generator and a discriminator that checks if the generated output is drawn from the actual solution distribution.

Researchers also have taken advantage of properties of the problem structure to improve feasibility. Based on hierarchical relations between different AC-OPF outputs (i.e. active and reactive power are determined by voltage magnitude and angle), Chatzos et al. [21] proposed OPF-DNN, a network consisting of four sequentially connected sub-networks to predict the outputs. The predictions of each sub-network are fed into the next one as an input along with the input demand data. Predicting the OPF results indirectly reduces the complexity and improves predictions. Huang et al. [52] observed that optimal generator dispatch could be directly drawn from the AC-PF equations if voltage magnitudes and angles were known. Therefore, they proposed a DNN that predicts these two variables and the rest are derived from AC-PF equations. While the model can naturally satisfy AC-feasibility, it does not do so for generator dispatch limits, which are later finetuned with a post-processing method.

An alternative approach proposed by [81], utilizing Physics Informed Neural Network (PINN), as an unsupervised model that is trained not by reducing Mean Squared Error between predictions and target values but reducing a loss function designed to involve Karush–Kuhn–Tucker (KKT) conditions, testing whether optimality and feasibility are satisfied. The study further proposes a method to measure the worst-case violations that the underlying network can produce. Although the results indicate less constraint violations, the model still cannot guarantee feasibility and can underperform in producing optimal results compared to training a supervised model [80].

One challenge for using NNs to solve AC-OPF is the existence of multiple equally optimal solutions for the same demand. In this case, it is not possible to train a Feed Forward NN properly. Kotary et al. [62] studied the connection between dataset and

NNs approximation power and proposed a preprocessing method before training. Huang and Chen [52] used AC-PF equations to train NN in an unsupervised fashion for a unique mapping from load and optimal dispatch. Pan et al. [86] proposed an augmented learning algorithm to augment the input (load) to create a unique mapping.

Overall, the literature claims large improvements, yet it would be fair to claim that the field is still underdeveloped. Ensuring optimal and feasible solutions is still an unsolved problem; while many studies claim fewer violations, they either trade off optimality and feasibility or simply cannot guarantee either one. Therefore, ML aided approaches are not ultimate solutions yet and are used as a warm start to regular solvers to satisfy the conditions as in [3, 30]. On the other hand, studies such as [13] can guarantee these conditions but underlying ADMM algorithm does not provide fast enough solutions.

Chapter 2

Background

This chapter provides the theoretical background underlying the proposed research. In Section 2.1, key concepts in convex optimization, including optimality conditions and sensitivity analysis for uncertain RHS vector is discussed. In Section 2.2, a general form of mp-QP problem is presented followed by a discussion on the critical solution regions and the global sensitivity of solutions. In Section 2.3, shallow and deep NN models are introduced, followed by a discussion showing that NNs can be viewed as a general form of PWL functions that can estimate any function in this form. Section 2.4 details the formulations of our primary application cases, the PF and OPF problems. It also derives the commonly used linearization of the OPF problem, known as the DC-OPF. The chapter will conclude with a discussion of the relevance and significance of these topics to this thesis.

2.1 Optimality Conditions for Constrained Optimization

A general form of continuous constrained optimization can be written as,

$$\text{Minimize: } F(\mathbf{x}) \tag{2.1a}$$

$$\text{s.t. } G(\mathbf{x}) = 0, \quad [\boldsymbol{\lambda}] \tag{2.1b}$$

$$H(\mathbf{x}) \leq 0, \quad [\boldsymbol{\mu}] \tag{2.1c}$$

where $F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$, $G(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^{m_1}$, $H(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^{m_2}$ are continuously differentiable functions¹, λ and μ are Lagrange multipliers of equality and inequality constraints, and m_1 and m_2 are numbers of equality and inequality constraints, respectively. The Lagrangian of the problem can be written as,

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = F(\mathbf{x}) + \boldsymbol{\lambda}^T G(\mathbf{x}) + \boldsymbol{\mu}^T H(\mathbf{x}). \tag{2.2}$$

For the problem in (2.1), an optimal and feasible solution must satisfy the following conditions.

¹The optimality conditions that will be defined here holds as long as both equations are continuously differentiable within the open set containing the local minimum.

$$\nabla F(\mathbf{x}^*) + \boldsymbol{\lambda}^{T*} \nabla G(\mathbf{x}^*) + \boldsymbol{\mu}^{T*} \nabla H(\mathbf{x}^*) = 0, \quad (2.3a)$$

$$G(\mathbf{x}^*) = 0, \quad (2.3b)$$

$$H(\mathbf{x}^*) \leq 0, \quad (2.3c)$$

$$\boldsymbol{\mu} \geq 0, \quad (2.3d)$$

$$\mu_j H_j(\mathbf{x}^*) = 0 \text{ for } j \in \mathcal{B}, \quad (2.3e)$$

where $\mathcal{B}$ is the set of binding constraints and H_j is the j th output of $H(\cdot)$. Here (2.3a) is known as the Stationarity condition, 2.3bc are Primal Feasibility conditions, 2.3d is the Dual Feasibility condition and 2.3e is the Complementary Slackness condition.

The conditions in (2.3) are known as KKT necessary optimality conditions, which are sufficient to guarantee the solution is globally optimal when $F(\mathbf{x})$ is convex [12, p. 285]. For a general class of optimization problems, however, the second order sufficiency conditions also necessitates the below to hold:

$$\mathbf{y}^T \nabla L_{xx}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \mathbf{y} \geq 0, \quad (2.4)$$

for all $\mathbf{y} \in \mathbb{R}^n$. While the addition of (2.4) is not necessary for the mp-QP problem addressed in this thesis, it is needed for the Theorem 1 discussed in the next section.

2.1.1 Sensitivity Analysis

The Basic Sensitivity Theorem [36] is foundational to the theory of MP programming, which will be discussed in the Section 2.2. The theorem provides insight into how optimal solutions change in response to perturbations in the RHS vectors, $(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c)$, within an ϵ -neighbourhood of the solution as shown in 2.5. It is formally stated as follows:

Theorem 1 (Sensitivity). [12, p. 384] *Let $\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*$ be a local optimal solution satisfying KKT conditions in (2.3) and second order condition in (2.4). Also, at the solution $\mathbf{x}^*$ the gradients of the active inequality constraints are linearly independent.*

$$\textbf{Minimize: } F(\mathbf{x}) \tag{2.5a}$$

$$\textbf{s.t. } G(\mathbf{x}) = \boldsymbol{\theta}_e, \quad [\boldsymbol{\lambda}] \tag{2.5b}$$

$$H(\mathbf{x}) \leq \boldsymbol{\theta}_c, \quad [\boldsymbol{\mu}] \tag{2.5c}$$

where $\boldsymbol{\theta}_e \in \mathbb{R}^{m_1}, \boldsymbol{\theta}_c \in \mathbb{R}^{m_2}$ are parameters of the problem. Then, for an open sphere $\mathcal{S} = \{(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c) \mid \|\boldsymbol{\theta}_e\|^2 + \|\boldsymbol{\theta}_c\|^2 < \epsilon^2\}$ centered around $(0,0)$ there exists a local optimal solution and associated dual variables $x(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c) \in \mathbb{R}^n, \lambda(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c) \in \mathbb{R}^{m_1}, \mu(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c) \in \mathbb{R}^{m_2}$ that correspond to each $(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c)$ drawn from this set.

Also, $x(\cdot), \lambda(\cdot), \mu(\cdot)$ are continuously differentiable functions in $\mathcal{S}$ with $\mathbf{x}^* = x(0,0), \boldsymbol{\lambda}^* = \lambda(0,0), \boldsymbol{\mu}^* = \mu(0,0)$.

Additionally, $\forall(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c) \in \mathcal{S}$, the below conditions hold:

$$\nabla_{\boldsymbol{\theta}_e} F(x(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c)) = -\lambda(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c), \quad (2.6a)$$

$$\nabla_{\boldsymbol{\theta}_c} F(x(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c)) = -\mu(\boldsymbol{\theta}_e, \boldsymbol{\theta}_c). \quad (2.6b)$$

The significance of the theorem is that it provides a mathematical framework for obtaining new optimal solutions using shadow prices when the right hand side is updated within a limited interval around original optimal points. The next section generalizes sensitivity analysis to outside the ϵ sphere using the notion of MP programming on QP with linear constraints. Particularly, it is shown that in the case of QP with linear constraints, functions defined in 2.6 can be expressed as linear functions of $\boldsymbol{\theta}_e, \boldsymbol{\theta}_c$ within an ϵ sphere of the initial solution and as a PWL function outside this region.

2.2 Multiparametric QP with Linear Constraints

Based on the findings of the sensitivity theorem, researchers studied multiparametric optimization problems that are defined with bounded uncertain parameters added to the RHS or cost function, and searched for solution strategies that can guarantee optimal and feasible solutions [84]. In this thesis, the quadratic problem with linear constraints is studied in the form of a MP program. This section details the problem and derives some abstract solution strategies to support the discussion in later chapters. Then, the piecewise affine nature of the global solution will be detailed in Section 2.2.2.

The mp-QP problem can be defined mathematically as

$$\underset{\mathbf{x}}{\text{Minimize}} \ z(\boldsymbol{\theta}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + (\mathbf{C} + \boldsymbol{\theta}_c)^T \mathbf{x} + \mathbf{C}_0, \quad (2.7a)$$

$$\text{s.t.} \ \mathbf{A}_e \mathbf{x} = \mathbf{b}_e + \boldsymbol{\theta}_e, \quad [\boldsymbol{\lambda}] \quad (2.7b)$$

$$\mathbf{A}_c \mathbf{x} \leq \mathbf{b}_c + \boldsymbol{\theta}_c, \quad [\boldsymbol{\mu}] \quad (2.7c)$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{C} \in \mathbb{R}^n$, $\mathbf{C}_0 \in \mathbb{R}^n$, $\mathbf{b}_e \in \mathbb{R}^{m_1}$, $\mathbf{b}_c \in \mathbb{R}^{m_2}$, $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is a semi-definite positive matrix, and $\mathbf{A}_e \in \mathbb{R}^{m_1 \times n}$, $\mathbf{A}_c \in \mathbb{R}^{m_2 \times n}$ and $\boldsymbol{\theta} \in \Theta_f$ where Θ_f is the set of all uncertain feasible parameters. Therefore, (2.7) is parametrized by $\boldsymbol{\theta} = [\boldsymbol{\theta}_c, \boldsymbol{\theta}_e, \boldsymbol{\theta}_c]$ where $\boldsymbol{\theta}_c \in \mathbb{R}^n$, $\boldsymbol{\theta}_e \in \mathbb{R}^{m_1}$, $\boldsymbol{\theta}_c \in \mathbb{R}^{m_2}$. Here, the subscript of $\mathbf{A}_c$ and $\mathbf{b}_c$ is the set of all inequality constraint indices, i.e. $\mathcal{C} = \{1, 2, \dots, m_2\}$. The Lagrangian function of the above problem can be written as,

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + (\mathbf{C} + \boldsymbol{\theta}_c)^T \mathbf{x} + \mathbf{C}_0 + \boldsymbol{\lambda}^T (\mathbf{b}_e + \boldsymbol{\theta}_e - \mathbf{A}_e \mathbf{x}) + \boldsymbol{\mu}^T (\mathbf{b}_c + \boldsymbol{\theta}_c - \mathbf{A}_c \mathbf{x}), \quad (2.8)$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{m_1}$, $\boldsymbol{\mu} \in \mathbb{R}^{m_2}$ are the dual variables associated with equality and inequality constraints, respectively.

In cases where the inequality constraints are not binding at the solution, it is known that $\boldsymbol{\mu}^* = 0$. Then, the solution can be calculated by setting $\boldsymbol{\mu} = 0$ in (2.8) and finding the point that sets the gradient of the Lagrangian to zero. However, if one or more inequality constraints are binding, then the solution can be attained in the same fashion if the set of binding constraints were known, i.e., by converting the binding inequality constraints to equality constraints and finding the critical point of the Lagrangian.

Assume that for any parameter set, $\boldsymbol{\theta}$, the set of indices of binding constraints $\mathcal{B} \subseteq \{1, \dots, m_2\}$ is known. Then, from KKT conditions in (2.3), the optimal solution to the problem can be found by solving

$$\frac{\partial L}{\partial \mathbf{x}} = 2\mathbf{Q}\mathbf{x} + \mathbf{C} + \boldsymbol{\theta}_c - \boldsymbol{\lambda}^T \mathbf{A}_e^T - \boldsymbol{\mu}^T \mathbf{A}_B^T = 0 \quad (2.9a)$$

$$\frac{\partial L}{\partial \boldsymbol{\lambda}} = \mathbf{b}_e + \boldsymbol{\theta}_e - \mathbf{A}_e \mathbf{x} = 0, \quad (2.9b)$$

$$\frac{\partial L}{\partial \boldsymbol{\mu}_B} = \mathbf{b}_B + \boldsymbol{\theta}_B - \mathbf{A}_B \mathbf{x} = 0, \quad (2.9c)$$

where $\mathbf{A}_B, \mathbf{b}_B$ consist of rows of coefficient matrices, $\mathbf{A}_C, \mathbf{b}_C$, corresponding to coefficient indices in $\mathcal{B} \subseteq \mathcal{C}$, and optimal shadow prices $\mu_i^* > 0 \forall i \in \mathcal{B}$ and $\mu_i^* = 0 \forall i \notin \mathcal{B}$.

Notice that $\boldsymbol{\mu}^*$ is a function of the parameter $\boldsymbol{\theta}$, which also determines which constraints are active at the solution. Following this observation, the next section outlines the solution function that generates the optimal solutions for any given $\boldsymbol{\theta}$.

2.2.1 Solution Function of Inequality Constrained mp-QP

Given the uncertain parameters, $\boldsymbol{\theta}_e, \boldsymbol{\theta}_C$, our goal is to derive a solution strategy that can calculate optimal solutions for any such parameter. For this purpose, we first assume that the function, $\boldsymbol{\mu}(\boldsymbol{\theta})$, that generates $\boldsymbol{\mu}^*$ is known, which later be relaxed in further chapters with the introduction of our proposed NN models. Under this assumption, optimal solutions to (2.7) can be attained by modifying (2.8) and writing the derivatives as follows:

$$\frac{\partial L}{\partial \mathbf{x}} = 2\mathbf{Q}\mathbf{x} + \mathbf{C} + \boldsymbol{\theta}_c - \boldsymbol{\lambda}^T \mathbf{A}_e^T - \mu^T(\boldsymbol{\theta}) \mathbf{A}_c^T = 0, \quad (2.10a)$$

$$\frac{\partial L}{\partial \boldsymbol{\lambda}} = \mathbf{b}_e + \boldsymbol{\theta}_e - \mathbf{A}_e \mathbf{x} = 0. \quad (2.10b)$$

The above system of equations can be rearranged in matrix form as follows:

$$\begin{bmatrix} 2\mathbf{Q} & -\mathbf{A}_e^T \\ -\mathbf{A}_e & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{C} - \boldsymbol{\theta}_c + \boldsymbol{\mu}^{T*} \mathbf{A}_c^T \\ -\mathbf{b}_e - \boldsymbol{\theta}_e \end{bmatrix}, \quad (2.11)$$

where $\boldsymbol{\mu}^* = \mu(\boldsymbol{\theta})$. Hence, optimal solutions, $\mathbf{x}^*, \boldsymbol{\lambda}$, for any $\boldsymbol{\theta} \in \Theta_f$ can be calculated as

$$\mathbf{J}^{-1} \begin{bmatrix} -\mathbf{C} - \boldsymbol{\theta}_c + \boldsymbol{\mu}^{T*} \mathbf{A}_c^T \\ -\mathbf{b}_e - \boldsymbol{\theta}_e \end{bmatrix} = \begin{bmatrix} \mathbf{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix}, \quad \text{for } \mathbf{J} = \begin{bmatrix} 2\mathbf{Q} & -\mathbf{A}_e^T \\ -\mathbf{A}_e & 0 \end{bmatrix}. \quad (2.12)$$

The mapping in (2.12) can be expressed as a linear function, $g : [\boldsymbol{\mu}^*, \boldsymbol{\theta}] \rightarrow [\mathbf{x}^*, \boldsymbol{\lambda}^*]$, which can be written as,

$$g(\boldsymbol{\mu}^*, \boldsymbol{\theta}) = \mathbf{J}^{-1} \begin{bmatrix} -\mathbf{C} - \boldsymbol{\theta}_c + \boldsymbol{\mu}^{T*} \mathbf{A}_c^T \\ -\mathbf{b}_e - \boldsymbol{\theta}_e \end{bmatrix}. \quad (2.13)$$

Given the known value of $\boldsymbol{\mu}^*$, equation 2.13 is a function that yields the the optimal primal and dual variables for (2.7a)-(2.7c) for any $\boldsymbol{\theta}$, i.e.,

$$g(\boldsymbol{\mu}^*, \boldsymbol{\theta}) = [\mathbf{x}^*, \boldsymbol{\lambda}^*]^T. \quad (2.14)$$

Chapters 4 and 5 propose models that provide exact predictions to functions, $\mu(\boldsymbol{\theta})$ and $g(\cdot)$ by representing them as subnetworks of a composite NN function.

As (2.14) is linear a linear mapping, the only nonlinearity can stem from $\boldsymbol{\mu}^*$. The next section discusses the piecewise affine nature of the solution function that produces optimal solutions to the problem in (2.7).

2.2.2 Global Properties of mp-QP Solutions

In this section, we present the theoretical background that primal and dual solutions are piecewise affine functions of $\boldsymbol{\theta}$ and show that this is due to the piecewise affine characteristics of $\mu(\boldsymbol{\theta})$. Then, we introduce the notion of critical region and present a theorem that details the properties of the binding set of constraints when moved among critical regions, which is essential to our proposed algorithm in Chapter 5.

The theorem below proves that the optimal primal solutions are piecewise affine and the optimal objective is a piecewise quadratic function of the parameters.

Theorem 2. *For the mp-QP problem 2.7, $\Theta_f \subseteq \Theta$ is a convex set, the primal solution function $\mathbf{x}(\boldsymbol{\theta}) : \Theta_f \rightarrow \mathbb{R}^n$ is continuous and piecewise affine. Also the optimal objective function $\mathbf{F}(\boldsymbol{\theta}) : \Theta_f \rightarrow \mathbb{R}$ is continuous and piecewise quadratic.*

Proof: The reader can find the proof of the theorem in [94].

Lemma 1. *Consider the mp-QP problem 2.7. The shadow prices of inequality constraints, $\boldsymbol{\mu}^* : \Theta_f \rightarrow \mathbb{R}^{m_2}$ is piecewise affine function.*

Proof: The mapping $g(\cdot)$ in eq. 2.14 is an affine mapping from $\boldsymbol{\mu}^* = \mu(\boldsymbol{\theta})$, and as shown in Theorem 2, $\mathbf{x}(\boldsymbol{\theta})$ is a piecewise affine function. As $g(\cdot)$ can be defined as a composite function, i.e., $(g \circ \mu)(\boldsymbol{\theta})$, the only input of the function has to be piecewise affine.

Theorem 2 and Lemma 1 demonstrate that the piecewise linearity of the primal and dual solutions is mainly due to the PWL characteristics of $\boldsymbol{\mu}^* = \mu(\boldsymbol{\theta})$, in contrast to the rest of the solutions that can be obtained with a linear mapping. As evident from (2.9), each $\mathcal{B}_i$ defines a feasible subset of $\boldsymbol{\theta}$, called a *critical region*, where solutions for all $\boldsymbol{\theta}$ from the same critical region can be obtained by solving the same system of equations since the set of binding constraints does not change at the solution. When moved away from this interval, the system of equations (thus the slope of $\mu(\boldsymbol{\theta})$) updates with a different $\mathcal{B}$, resulting in piecewise linear shape.

Figure 2.1 illustrates changes in the slope of $\mu(\boldsymbol{\theta})$ with respect to each critical region over an example. For this example, a quadratic optimization problem with $\mathbf{x} \in \mathbb{R}^8, \lambda \in \mathbb{R}^6, \boldsymbol{\lambda} \in \mathbb{R}^6$ was solved repeatedly for varying RHS of equality constraints. Specifically, θ_1 was increased from 0 to maximum feasible value and the other parameters set to 0. It can be seen that there are 5 critical regions depending on which constraint is active at the solution (i.e. $\mu_i^* > 0$), and at each region the slope of $\mu(\boldsymbol{\theta})$ is different.

Notice that as $\boldsymbol{\theta}$ passes from one critical region to another, either the shadow price of a constraint changes from 0 to positive or vice versa. In other words, at the intersection of two critical regions, the set of binding constraints either expands a new constraint or one of the binding constraints no longer binds. Moreover, at the intersection of the two critical regions, adjacent line segments produce the same value of the shadow price. This is an

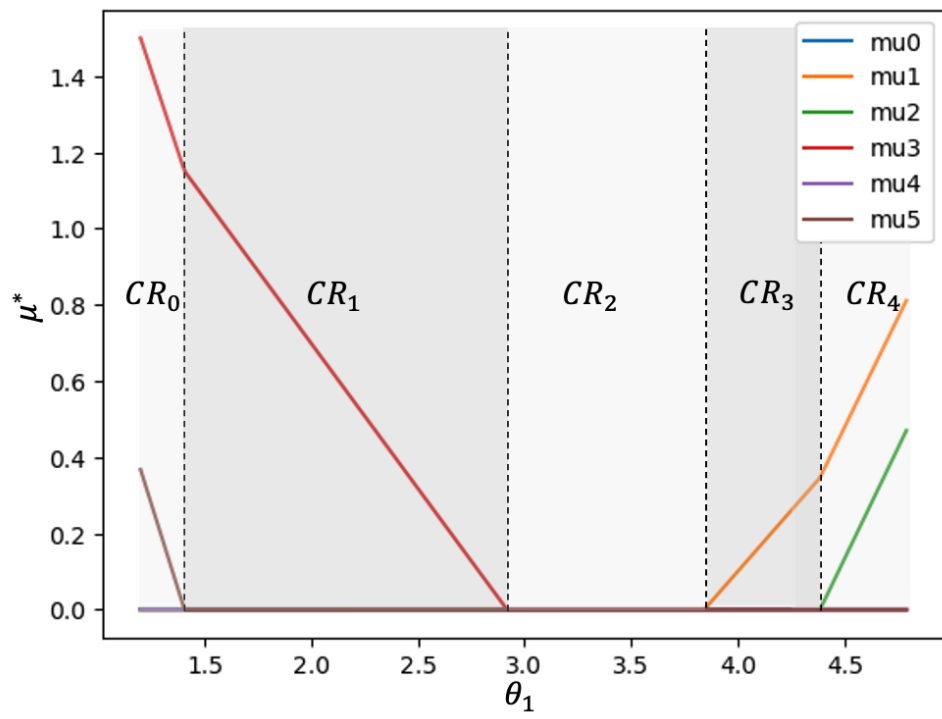


Figure 2.1: Sensitivity of μ against θ_1

important property of mp-QP that is leveraged by the method that is proposed in future chapters. Theorem 3 shows that either one of the two cases occurs when transitioning between critical regions.

Theorem 3. *Consider an active set $\mathcal{B}_i = \{k_1, k_2, \dots, k_m\}$ and its corresponding critical region CR_i in minimal representation, i.e. with all redundant constraints removed. Also, let CR_j be a full-dimensional adjacent critical region to CR_i and independent constraint qualification holds for facet $H = CR_i \cap CR_j$, and let $\mu(\boldsymbol{\theta})$ be the dual variables of the inequality constraints. Moreover, assume that there are no constraints, which are weakly active at the optimal solution $\mathbf{x}^* = x(\boldsymbol{\theta})$ at $\forall \boldsymbol{\theta} \in CR_i$. Then one of the following holds:*

- *Expanding: H is given by $\mathbf{A}_{m+1}x(\boldsymbol{\theta}) = \mathbf{b}_{k_{m+1}} + \boldsymbol{\theta}_{m+1}$, where $\mathbf{A}_{m+1}, \mathbf{b}_{m+1}$ are coefficients of the k_{m+1} th inequality constraint. Then the optimal active set in CR_j is $\{k_1, k_2, \dots, k_{m+1}\}$*
- *Diminishing: If H is given by the shadow price of the k_m th constraint, $\mu_{k_m}(\boldsymbol{\theta}) = 0$, then the optimal active set in CR_j is $\mathcal{B}_j = \{k_1, k_2, \dots, k_{m-1}\}$.*

Proof. See [94, ch 3] for the proof. □

The following section presents the mathematical formulation of the NN model and discusses its training. Also, a derivation of the PWL nature of the NN model with ReLU activation function is presented to provide mathematical background of our thesis.

2.3 Neural Networks

Neural Networks are universal function approximators that are defined as composite functions that can be expressed as an indirect mapping between inputs $\mathbf{x}$ and outputs $\mathbf{y}$, $f : \mathbf{x} \rightarrow \mathbf{y}$. The basic type of NN, known as shallow feed forward neural network, consists of an input and output layer, connected by a hidden layer, h . Loosely influenced by neurons and synapses of the brain, the value of each neuron is determined by the weighted sum of the previous neurons' values, which is called *input current* denoted by z_i for node i . However, the neuron's value is not always defined as its input current but usually transformed by an activation function. For example, if the activation is a threshold function such as ReLU, it is only activated if the input current's value is positive. This specific case can be expressed mathematically as follows:

$$z_i^1 = \sum_{j=1}^{d_x} W_{ji}^0 x_j + b_i^0, \quad h_i = \sigma(z_i^1), \quad (2.15)$$

where z_i^1 is input current of node i in the hidden layer, W_{ji}^0 is the weight of the edge connecting node j of one layer to node i of the next layer (here input layer to hidden layer), b_i^0 is the bias term adjusting the calculated value for neuron i and $\sigma(z_i^1) = ReLU(z_i^1)$ is an element-wise activation function defined as

$$ReLU(z_i) = \begin{cases} z_i & \text{if } z_i \geq 0, \\ 0 & \text{otherwise.} \end{cases} \quad (2.16)$$

The output layer is defined by the previous layer, $h(\mathbf{x})$, in a similar fashion as

$$z_k^2 = \sum_{i=1}^{d_h} W_{ik}^1 h_i + b_k^1, \quad y_k = \sigma(z_k^2). \quad (2.17)$$

Here, the activation function $\sigma(\cdot)$ is not necessarily the same as the one activating the previous layer. For simplicity, a generic notation $\sigma(\cdot)$ will be used to denote all activation functions in this thesis.

Overall, the function, $\mathbf{y} = f(\mathbf{x})$, can be calculated in algebraic form as

$$\begin{aligned} \mathbf{h} &= \sigma(\mathbf{W}^{1T} \mathbf{x}^T + \mathbf{b}^1), \\ \mathbf{y} &= \sigma(\mathbf{W}^{2T} \mathbf{h}^T + \mathbf{b}^2), \end{aligned} \quad (2.18)$$

where the superscript T denotes the transpose operation and $\mathbf{W}^{iT} = (\mathbf{W}^i)^T$ is used to simplify the notation, $\mathbf{x} = [x_1, \dots, x_{d_x}]$, $\mathbf{h} = [h_1, \dots, h_{d_h}]$, $\mathbf{y} = [y_1, \dots, y_{d_y}]$, $\mathbf{x} \in \mathbb{R}^{d_x \times d_x}$, $\mathbf{h} \in \mathbb{R}^{d_d \times d_h}$, $\mathbf{y} \in \mathbb{R}^{d_d \times d_y}$. Here, $\mathbf{W}^i \in \mathbb{R}^{n \times m}$ and $\mathbf{b}^i \in \mathbb{R}^{n \times 1}$ are parameters known as weight and bias terms that are optimized during NN training, and generically defined as,

$$\mathbf{W}^i = \begin{bmatrix} W_{11}^i & \dots & W_{1m}^i \\ \vdots & \ddots & \vdots \\ W_{n1}^i & \dots & W_{nm}^i \end{bmatrix}, \quad \mathbf{b}^i = \begin{bmatrix} b_1^i \\ \vdots \\ b_n^i \end{bmatrix}. \quad (2.19)$$

For the model in (2.18), $\mathbf{W}^1 \in \mathbb{R}^{d_x \times d_h}$, $\mathbf{b}^1 \in \mathbb{R}^{d_h}$, $\mathbf{W}^2 \in \mathbb{R}^{d_h \times d_y}$ and $\mathbf{b}^2 \in \mathbb{R}^{d_y}$.

The above example expresses a *shallow network*, i.e., a neural network with one hidden layer (illustrated in Figure 2.2a). In case of a network with two or more hidden layers,

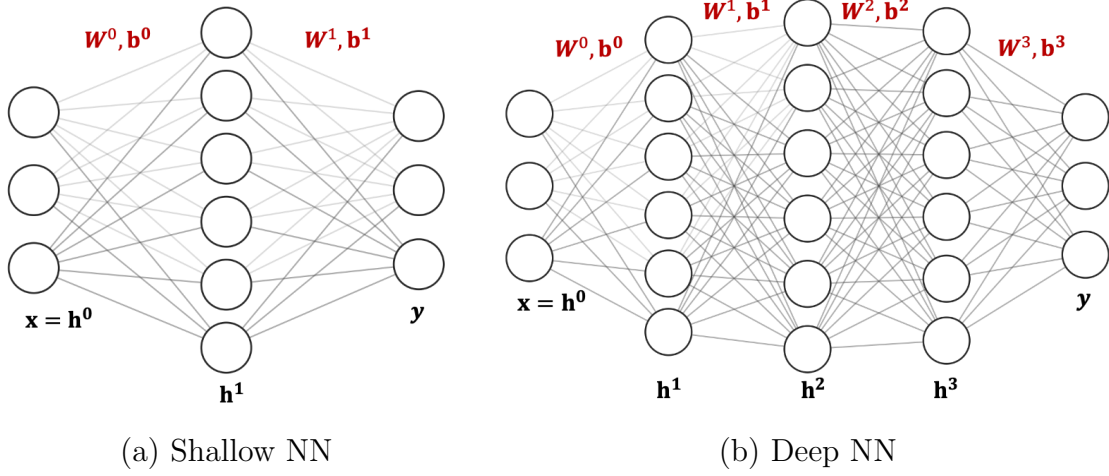


Figure 2.2: Illustration of Shallow and Deep NNs

i.e., *deep neural network* (illustrated in Figure 2.2b), the output value can be calculated iteratively as follows:

$$\begin{aligned}
 \mathbf{h}^0 &= \mathbf{x}^T, \\
 \mathbf{h}^i &= \sigma(\mathbf{W}^{i-1T} \mathbf{h}^{i-1T} + \mathbf{b}^{i-1}), \quad \text{for } i \in [1, \dots, n_h], \\
 \mathbf{y} &= \sigma(\mathbf{h}^{n_hT}),
 \end{aligned} \tag{2.20}$$

where n_h is the number of hidden layers.

2.3.1 Neural Networks as Piecewise Linear Functions

The proposed solution methodology in this thesis relies on the observation that shallow NNs with ReLU activation function fall under the class of PWL functions whose slope and intersection parameters are estimated from data. Therefore, as PWL functions, shallow

NNs can approximate any continuous function by fitting $d_h + 1$ linear facets. Thus, as d_h increases, the estimation of the function improves given that the model is trained with a large dataset. To support the discussion in this paper, this section formulates NN as PWL functions.

A shallow NN with d_h neurons and no activation function in output layer can be written as

$$f(\mathbf{x}; \varphi) = \mathbf{W}^{1T} \sigma(\mathbf{W}^{0T} \mathbf{x}^T + \mathbf{b}^0) + \mathbf{b}^1, \quad (2.21)$$

where $\varphi = [\mathbf{W}^0, \mathbf{W}^1, \mathbf{b}^0, \mathbf{b}^1]$ is the parameter set, $\mathbf{W}^0 \in \mathbb{R}^{d_h \times d_x}$, $\mathbf{W}^1 \in \mathbb{R}^{d_y \times d_h}$, $\mathbf{b}^0 \in \mathbb{R}^{d_h}$, $\mathbf{b}^1 \in \mathbb{R}^{d_y}$, $\sigma(\mathbf{z}) = \text{ReLU}(\mathbf{z})$, $\mathbf{x} \in \mathbb{R}^{d_d \times d_x}$, d_x, d_y are input and output dimensions of the NN, respectively, and d_d is the number of observations.

For the case when $d_y = 1$, for a given observation, $\mathbf{x}_i \in \mathbb{R}^{d_x}$, hidden layer weights $\mathbf{W}_j^0 \in \mathbb{R}^{d_x}$, $\mathbf{W}_j^1 \in \mathbb{R}$ and scalar b_j for $j \in [1, \dots, d_h]$, the model can be rewritten as

$$f(\mathbf{x}_i; \varphi) = \sum_{j=1}^{d_h} \mathbf{W}_j^1 (\mathbf{W}_j^0 \mathbf{x}_i + b_j^0)^+ + b^1, \quad (2.22)$$

where $(z)^+ = z$ if $z > 0$ and 0 otherwise.

It can be seen that the above function is a weighted summation of hidden layer output, which generates a different linear function within a region of $\mathbf{x}_i$ depending on whether one or more nodes are active, i.e., $(\mathbf{W}_j^0 \mathbf{x}_i + b_j^0) > 0$ in that region. Expressed mathematically, let $\mathcal{B}_k$ be the set of indices of nodes that are active for a given $\mathbf{x}$, i.e., $\mathcal{B}_k = \{j \mid \mathbf{W}_j^0 \mathbf{x} + b_j^0 > 0\}$. Also let $\mathcal{R}_k \subseteq \mathcal{D}$ be a subdomain of all x in which the nodes with indices in $\mathcal{B}_k$ are active, i.e., $\mathcal{R}_k = \{\mathbf{x} \mid \mathbf{W}_j^0 \mathbf{x} + b_j^0 > 0, j \in \mathcal{B}_k\}$.

Notice that for a compact domain of $\mathbf{x} \in \mathcal{D}$, there is a finite number K such that $\bigcup_{k=1}^K \mathcal{R}_k = \mathcal{D}$ and $\mathcal{R}_k \cap \mathcal{R}_{k'} = \emptyset$ for $k \neq k'$. Then we can write,

$$f(\mathbf{x}; \theta) = b^1 + \begin{cases} 0 & \text{for } \mathbf{x} \in \mathcal{R}_0 \\ \sum_{j \in \mathcal{B}_1} \mathbf{W}_j^1 \mathbf{W}_j^0 x_i + \mathbf{W}_j^1 b_j^0 & \text{for } \mathbf{x} \in \mathcal{R}_1 \\ \vdots & \vdots \\ \sum_{j \in \mathcal{B}_{K-1}} \mathbf{W}_j^1 \mathbf{W}_j^0 x_i + \mathbf{W}_j^1 b_j^0 & \text{for } \mathbf{x} \in \mathcal{R}_{K-1} \\ \sum_{j \in \mathcal{B}_K} \mathbf{W}_j^1 \mathbf{W}_j^0 x_i + \mathbf{W}_j^1 b_j^0 & \text{for } \mathbf{x} \in \mathcal{R}_K \end{cases}. \quad (2.23)$$

Notice that, for each $\mathcal{R}_k$, NN can generate a different linear function. Also, the maximum number of linear functions that a model can generate is limited by the hidden layer size, $K = d_h + 1$.

By substituting $\mathbf{M}_i = \sum_{j \in \mathcal{B}_i} \mathbf{W}_j^1 \mathbf{W}_j^0 \mathbf{x}$, $\mathbf{N}_i = \sum_{j \in \mathcal{B}_i} \mathbf{W}_j^1 b_j^0$, one can obtain the general

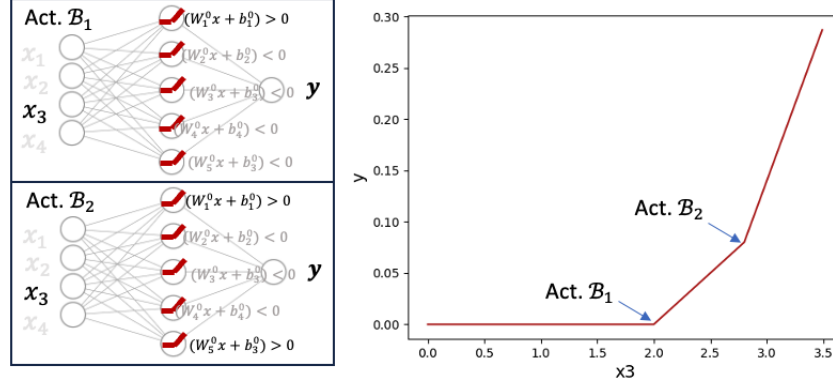


Figure 2.3: Illustration of NN as a Piecewise Linear Function

form a PWL function,

$$f(x_i, \theta) = \begin{cases} \mathbf{N}_0 & \text{for } \mathbf{x} \in \mathcal{R}_0 \\ \mathbf{M}_1 \mathbf{x} + \mathbf{N}_1 & \text{for } \mathbf{x} \in \mathcal{R}_1 \\ \vdots & \vdots \\ \mathbf{M}_{K-1} \mathbf{x} + \mathbf{N}_{K-1} & \text{for } \mathbf{x} \in \mathcal{R}_{K-1} \\ \mathbf{M}_K \mathbf{x} + \mathbf{N}_K & \text{for } \mathbf{x} \in \mathcal{R}_K \end{cases} \quad (2.24)$$

Figure 2.3 illustrates the NN as a piecewise linear function. As illustrated, for $x_3 < 2$, none of the hidden neurons have a positive value as the $\mathbf{W}_j^0 x + b_j < 0$ for all j . In the interval $2 \leq x_3 < 4$, the set of active neurons is $\mathcal{B}_1 = \{1\}$ as only $\mathbf{W}_1^0 x + b_1 > 0$ and the output increases with a certain slope as x_3 increases. After $x_3 \geq 4$, another slope is also activated and the set of active slopes are $\mathcal{B}_2 = \{1, 5\}$, which updates the change of the output value with respect to x_3 .

If the target function is not PWL, but an arbitrary nonlinear continuous function defined in a bounded interval, $x_i \in [a, b]$, then the NN can be used to approximate the function, and the prediction accuracy increases as $d_h \rightarrow \infty$, in accordance with the Universal Approximation Theorem (UAT) [51].

2.4 Operational Models for Power Systems

This section reviews the formulations of the application field of the proposed models in 3, 4 and 5. We will illustrate AC Optimal Power Flow (AC-OPF) problem followed by the linearization of the problem, DC-OPF.

2.4.1 Optimal Power Flow Problem

OPF is a central problem in Power System Management, which was first proposed by Carpentier in 1962 [18]. The goal of the problem is to find the optimal dispatch per generator to minimize an objective function such as cost of operation while satisfying the energy demand and the grid system constraints at a given point in time.

Unlike earlier methods such as the economic load dispatch problem, OPF also considers power flow constraints that consist of complex-valued electrical quantities, which is defined

mathematically as below:

$$P_i^g - P_i^d = \sum_j |V_i||V_j|(g_{ij} \cos \delta_{ij} + b_{ij} \sin \delta_{ij}) \quad \forall i \in \mathcal{N}, \quad (2.25a)$$

$$Q_i^g - Q_i^d = \sum_j |V_i||V_j|(g_{ij} \sin \delta_{ij} - b_{ij} \cos \delta_{ij}) \quad \forall i \in \mathcal{N}. \quad (2.25b)$$

where $\mathcal{N}$ is the set of buses, P_i^g, Q_i^g are active and reactive power generator dispatch, P_i^d, Q_i^d are active and reactive power demand, V_i, δ_i are voltage magnitudes at each bus and angles at each bus, and $\delta_{ij} = \delta_i - \delta_j$ is the voltage angle difference between bus i and j . The above formulae are known as AC power flow (AC-PF) equations. As presented above, the power injected into the system is formulated with trigonometric functions to avoid complex representations. This introduces additional nonlinearity that complicates finding the global minimum during optimization.

Using the above formulation of power flow equations, the AC-OPF problem can be

written as below:

$$\min \sum_{i=1}^{\mathcal{N}_g} C_i(P_i^g), \quad (2.26a)$$

s.t.

$$P_i^g - P_i^d = \sum_j |V_i||V_j|(g_{ij} \cos \delta_{ij} + b_{ij} \sin \delta_{ij}) \quad \forall i \in \mathcal{N}, \quad (2.26b)$$

$$Q_i^g - Q_i^d = \sum_j |V_i||V_j|(g_{ij} \sin \delta_{ij} - b_{ij} \cos \delta_{ij}) \quad \forall i \in \mathcal{N}, \quad (2.26c)$$

$$P_i^{Min} \leq P_i \leq P_i^{Max} \quad \forall i \in \mathcal{N}_g, \quad (2.26d)$$

$$Q_i^{Min} \leq Q_i \leq Q_i^{Max} \quad \forall i \in \mathcal{N}_g, \quad (2.26e)$$

$$V_i^{Min} \leq V_i \leq V_i^{Max} \quad \forall i \in \mathcal{N}, \quad (2.26f)$$

$$\delta_{ij}^{Min} \leq \delta_{ij} \leq \delta_{ij}^{Max} \quad \forall i \in \mathcal{N}, \quad (2.26g)$$

where $\mathcal{N}_G$ is the indices of generator buses and $C_i(P_i^g)$ is the cost of production. Notice that equations 2.26b and 2.26c are power flow equations as defined earlier, which must be satisfied to enforce physics of the system.

2.4.2 DC OPF Problem

The challenges in finding the optimal and feasible solution led researchers to different approaches. One of the most common approaches is to solve a simplified problem, DC-OPF, which is a linearized version of the original problem under some assumptions. The motivation behind the linear relaxation is that the difference between voltage angles at transmission level is very small, i.e. $\delta_i - \delta_j \approx 0$. Hence, it can be assumed that $\sin(\delta_i - \delta_j) \approx \delta_i - \delta_j$

and $\cos(\delta_i - \delta_j) \approx 1$. This also results in small variations of voltage magnitudes around unity and it can be assumed that $V_i \approx 1$. Thus, the problem is freed from trigonometric representation along with reactive power and voltage magnitude terms and can be rewritten as,

$$\min \sum_{i=1}^{\mathcal{N}_g} C_i(P_i^g) \quad (2.27a)$$

s.t.

$$- \sum_{j=1}^{\mathcal{N}} B_{ij} \delta_j = P_i^g - P_i^d, \quad (2.27b)$$

$$P_i^{Min} \leq P_i \leq P_i^{Max} \quad \forall i \in \mathcal{N}_g, \quad (2.27c)$$

$$- F_{ij}^+ \leq (\delta_i - \delta_j) B_{ij} \leq F_{ij}^+ \quad \forall i \neq j \in \mathcal{N}, \quad (2.27d)$$

where B_{ij} is the susceptance between buses i and j . In addition to generator limits, (2.27d) was added to force line limits to maintain reliability of the system. This thesis will use the DC-OPF problem in matrix form in next Chapters 4 and 5, which can be written as follows:

$$\min \mathbf{P}^{gT} \mathbf{Q} \mathbf{P}^g, \quad (2.28a)$$

s.t.

$$- \mathbf{B} \boldsymbol{\delta} = \mathbf{P}^g - \mathbf{P}^d, \quad (2.28b)$$

$$\mathbf{P}^+ \leq \mathbf{P}^g \leq \mathbf{P}^+, \quad (2.28c)$$

$$\mathbf{F}^+ \leq \mathcal{H} \mathbf{B} \boldsymbol{\delta} \leq \mathbf{F}^+, \quad (2.28d)$$

where $\boldsymbol{\theta}$ is the vector of voltage angles. $\mathbf{F}^{max} \in \mathbb{R}^{n_l}$, where n_l is the number of all lines in the system and $\mathcal{H} \in \mathbb{R}^{n_l \times n}$ is the incidence matrix that constructs line flows, defined as:

$$\mathcal{H} = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 & 0 \\ 1 & 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & -1 \end{bmatrix} \quad (2.29)$$

The method is commonly used by system operators to determine optimal dispatch due to the simplicity of the calculations. However, research shows that solutions to the DC-OPF problem are not guaranteed to be feasible for the AC-OPF problem [4]. Moreover, as stated above, the model is more appropriate for transmission level and the assumptions do not hold at distributional networks where the voltage constraints are more likely to bind and reactive power is too large to be neglected [90].

The difficulties led researchers to alternative methods including NN based approaches, which is one of the motivations behind this thesis.

2.5 Summary of the Chapter

This chapter provided a comprehensive overview of the related background to this thesis. The chapter began with multiparametric optimization, neural networks and an overview of power flow problems, all of which are fundamental to understanding the subsequent chapters.

We reviewed the theoretical background related to MP programming, highlighted the KKT optimality conditions, basic sensitivity theorem and global properties of the mp-QP solutions. Furthermore, solution function to mp-QP problem that assumes the shadow prices of inequality constraints are known, which is later relaxed with a NN-based shadow price predictor to calculate error-free the solutions using a NN.

This chapter also provided a mathematical definition of NN model with ReLU activation function and illustrated its PWL nature. In Chapter 3, this property of NN model is used to understand the relationship between model approximation errors and dataset size. In Chapter 4 and 5, the piecewise affine property of the NN model is leveraged to calculate the error-free mp-QP solutions whose functional property is also piecewise affine.

Finally, the necessary background of our application cases are detailed in Section 2.4. The PF problem addressed in Chapter 3 is defined using (2.25) and the Chapters 4 and 5 provide solutions to the DC-OPF problem defined in 2.28.

Chapter 3

Solver-Free Data Generation

Approach for Training Models for Nonlinear Problems

NN models have gained popularity as surrogate optimization solvers in diverse domains, from chemistry [58] to model predictive control [56], thanks to its solution speed advantages compared to traditional Quasi-Newton-Raphson approaches. The Newton-Raphson method involves calculating the inverse Hessian matrix for every iteration, with computational complexity $\mathcal{O}(n^3)$. Quasi-Newton solvers such as Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm reduces the number of operations to $\mathcal{O}(n^2)$ by applying n corrections to the inverse Hessian at every iteration, while limited memory BFGS (L-BFGS) reduces complexity by choosing m number of correction vectors where $m \ll n$ [83]. On the other

hand, a fully connected NN can compute predictions in a single iteration for a large number of data points, with computational complexity $\mathcal{O}(nd_h + (n_h - 1)d_h^2 + nd_h)$, where n is the input and output of the model, n_h the number of layers, and d_h the number of neurons per layer. Additionally, due to its non-iterative nature, NN predictions can easily be parallelized, resulting in substantial speed advantages compared to the Quasi-Newton solution methods. However, there is no guarantee of the optimality of the solutions as NNs are black-box models defined for all real numbers, restricted only by the activation functions.

Most applications treat the NN model as a black-box and employ standard training methods using datasets of input-output pairs obtained from previously solved optimization problems. However, using this approach to train a NN-based solver necessitates a large training dataset that is computationally expensive to obtain, yet fails to guarantee feasible and optimal results. Additionally, the model performance relies on the performance of the optimization solver that is used to construct the dataset. Despite the theoretical guarantees given by UAT, the theorem does not specify how such a network can be trained in practice and depending on the application at hand, the majority of the proposed NN-based approaches for optimization are not able to guarantee predictions at a reasonable error threshold for a given domain of inputs.[38, 71, 81].

Motivated by the UAT, this chapter examines the accuracy of NN prediction in relation to dataset size, problem size, and the construction of training data. As shown in this chapter, a sufficiently large NN can produce accurate predictions when trained with very large datasets, but even so its performance is limited by the quality of the training dataset, which is traditionally obtained by the solution of optimization problems for any given input. To overcome limitations on dataset quality and computational inefficiency, a novel data

generation approach is proposed that bypasses the need for solvers to construct datasets by directly sampling data from the underlying problem functions.

Further, we will apply the proposed method in the power system domain for a number of reasons. First, models that cannot guarantee predictions with less than an acceptable accuracy level are impractical for use by system operators. Second, many proposed approaches assume a certain input demand distribution and train the model to produce predictions for likely events. However, these models are unguarded against unexpected load characteristics outside the training distribution due to extreme events such as pandemics or blackouts. Therefore, providing insights into the training of NNs and improving prediction accuracy for unlikely events are essential for applying NN-based optimization models to power systems.

This chapter addresses these questions first by examining the error produced by assuming an ideal NN model trained under certain assumptions to solve PF problems, explaining the relationship between the model error, the problem and dataset size. Then, we proceed to conduct a series of experiments on arbitrary nonlinear functions to highlight the relationship between dataset size, model complexity and problem size. In light of these discussions, this chapter proposes a solver-free data generation method that bypasses traditional solvers and enables training on arbitrarily large high quality training sets without dataset construction. While there is no direct state of the art to compare our approach, our goal is to outperform Quasi-Newton methods. Therefore, models trained with our approach are compared on IEEE test cases against Newton Raphson (NR) based solvers, and DNNs trained with solver generated datasets. The chapter contributions to the ML for optimization literature by:

1. proposing a data generation approach utilizing physical attributes of the power system to generate large training datasets without using nonlinear solvers, which results in significantly less error compared to solver-based models due to the quality and size of the data,
2. drawing similarities between the proposed solver-free approach and traditional NR based solvers for creating training data, considering speed and accuracy.

The chapter is organized as follows. Section 3.1 details the power flow (PF) equations and traditional ways to solve the PF problem. In Section 3.1.2, our data generation approach is proposed and compared against the traditional NR based solvers. Section 3.1.5 derives an analytic upper-bound to model approximation error to examine the relationship between the error and dataset size under certain assumptions on training and the problem. In Section 3.2, results on IEEE test cases are presented in Section 3.3 discusses limitations, implications and future work.

3.1 Methodology

This section details our dataset-generating approach that bypasses solvers and the PF problem that is foundational for checking the stability of power systems detailed in Section 2.4. Therefore, first the power flow problem is defined and then shown how it could be organized to generate arbitrarily large datasets with minimal computational cost.

3.1.1 Power Flow Problem

The PF problem used in power-system applications to find reactive power injections, voltage magnitude and angles under steady state condition parameters. The problem is formally defined as follows:

$$\min_{V_i, \delta_i, Q_i^d} \sum_{i=1}^{N_{bus}} (\epsilon_{pi}^2 + \epsilon_{qi}^2) \quad (3.1a)$$

$$\text{s.t.} \sum_{k=1}^{N_{bus}} |V_i| |V_k| (G_{ik} \cos \delta_{ik} + B_{ik} \sin \delta_{ik}) - P_i = \epsilon_{pi} \quad \forall i, \quad (3.1b)$$

$$\sum_{k=1}^{N_{bus}} |V_i| |V_k| (G_{ik} \sin \delta_{ik} - B_{ik} \cos \delta_{ik}) - Q_i = \epsilon_{qi} \quad \forall i, \quad (3.1c)$$

where $\epsilon_{pi}, \epsilon_{qi}$ are mismatch variables to be minimized, $|V_i|$ voltage magnitude at bus i and δ_{ik} is voltage angle differences between buses i and k , $\delta_{ik} = \delta_i - \delta_k$ and G_{ik}, B_{ik} are conductance and susceptance between bus i to bus k , $P_i = P_i^g - P_i^d, Q_i = Q_i^g - Q_i^d$ are active and reactive power injections, P_i^g, Q_i^g active and reactive power dispatches, and P_i^d, Q_i^d active and reactive demand respectively. The objective is to satisfy the power flow equations given in (3.1b) and (3.1c) by minimizing the mismatch variables, ideally to zero. For the rest of the paper, the decision variables are referred to in vector form using bold characters, e.g. $\mathbf{v}^g, \mathbf{v}^d$ to denote voltage magnitudes of generator and demand buses, and $\boldsymbol{\delta}$ to denote voltage angles at all buses.

Commonly in the literature, NN-based solvers used to solve power system optimization problems are trained on datasets constructed by randomly generating the known values of generators and demand buses: $\mathbf{P}^g, \mathbf{v}^g, \mathbf{P}^d$, and $\mathbf{Q}^d$, and running a solver to find correspond-

ing optimal solutions for unknown values in the PF equations: $\mathbf{Q}^g, \mathbf{v}^d$ and $\boldsymbol{\delta}$. However, training a generic ReLU NN for the mapping $m : [\mathbf{P}^g, \mathbf{v}^g, \mathbf{P}^d, \mathbf{Q}^d] \rightarrow [\mathbf{Q}^g, \mathbf{v}^d, \boldsymbol{\delta}]$ with less than the desired error is not possible in larger power systems. Specifically, UAT implies that (i) the model should be complex enough to approximate the target function, (ii) the training set should be dense within the domain, and (iii) the training set must contain all the boundary points. If $m(\mathbf{P}^g, \mathbf{v}^g, \mathbf{P}^d, \mathbf{Q}^d)$ is a $2N_{bus}$ dimensional multivariate function, the third condition only suggests that the training set must contain $2^{2N_{bus}}$ boundary points, which is impractical for large systems. For example, for $N_{bus} = 100$, 1.61E+60 datapoints are required.

Thus, constructing dense training sets requires generating random system parameters and running a solver in a loop for each input. This approach is not favourable for two reasons: First is the computational challenge of running an optimization algorithm repeatedly, as discussed in the introduction. Secondly, approximation by training on such a dataset inherently assumes that the data accurately reflect the target function, which is determined by the termination error tolerance of the minimization algorithm used to generate the data. Therefore, the performance of a NN trained on such a dataset is bounded by this error even with unlimited dataset size.

3.1.2 Generator-Predictor Approach for Data Construction

A hypothetical closed-form solution to the AC-PF problem would be a mapping $m : [\mathbf{P}^g, \mathbf{v}^g, \mathbf{P}^d, \mathbf{Q}^d] \rightarrow [\mathbf{Q}^g, \mathbf{v}^d, \boldsymbol{\delta}]$ that could generate any optimal solutions with global guarantees and remove the necessity to run an iterative solver for a given input parameter.

While this ideal mapping is unavailable, its inverse, $m^{-1} : [\mathbf{Q}^g, \mathbf{v}^d, \boldsymbol{\delta}] \rightarrow [\mathbf{P}^g, \mathbf{v}^g, \mathbf{P}^d, \mathbf{Q}^d]$, is fairly accessible through the PF equations (3.1b)-(3.1c). More clearly, if $\mathbf{v}, \boldsymbol{\delta}$ is given, then $\mathbf{P}, \mathbf{Q}^g, \mathbf{Q}^d$ can be obtained directly from the equations, and the inputs and outputs can be reorganized to form m^{-1} by isolating $\mathbf{v}^g$ from $\mathbf{v}^d$. Motivated by this observation, our approach generates candidate optimal solutions, $\boldsymbol{\delta}, \mathbf{v}^d$ as well as $\mathbf{v}^g$, calculates the corresponding $\mathbf{P}^g, \mathbf{P}^d, \mathbf{Q}^g, \mathbf{Q}^d$ via the PF equations and separates $\mathbf{v}^d$ from $\mathbf{v}^g$. Once a dense enough dataset is obtained, the generated input variables and corresponding outputs are swapped and reorganized to construct an error-free dataset that forms the mapping, m .

Algorithm 1 Solver-Free Data Generator Approach

```

1: Input: Number of samples  $d_d$ , bus data, line data, power flow equations
2: Output: Trained neural network model
3: procedure CONSTRUCTDATASET
4:   Initialize empty dataset  $\mathcal{D}$ 
5:   for  $j = 1$  to  $d_d$  do
6:     Generate  $|V_j| = \text{Uniform}(V^-, V^+)$ ,  $\delta_j = \text{Uniform}(\delta^-, \delta^+)$ ,
7:     Substitute  $|V_j|$  and  $\delta_j$  equations (2) and (3) and calculate  $\mathbf{P}_j, \mathbf{Q}_j$ ,
8:     Generate  $Q_j^d = Q_{j, \text{base}}^d \cdot \text{Uniform}(0.6, 1.4)$ ,  $Q_j^g = Q_j - Q_j^d$ ,
9:     Split  $\mathbf{v}_j$  into  $\mathbf{v}_j^g, \mathbf{v}_j^d$ ,
10:    Append  $([\mathbf{P}_j, \mathbf{v}_j^g, \mathbf{Q}_j^d], [\boldsymbol{\delta}_j, \mathbf{v}_j^d, \mathbf{Q}_j^g])$  pairs to  $\mathcal{D}$ ,
11:   end for
12:   return  $\mathcal{D}$ 
13: end procedure

```

Algorithm 1 details the data generator. The algorithm first samples V_i, δ_i and Q_i^d from a reasonable input range and substitutes in the power flow equations in 3.1b-3.1c. The input $\mathbf{v}$ is split into $\mathbf{v}^g, \mathbf{v}^d$ and the variables are arranged as input/output pairs for training.

While a NN trained on this dataset is yet another model approximating the mapping, m , it does not inherit the optimization solver's error in contrast to conventional data

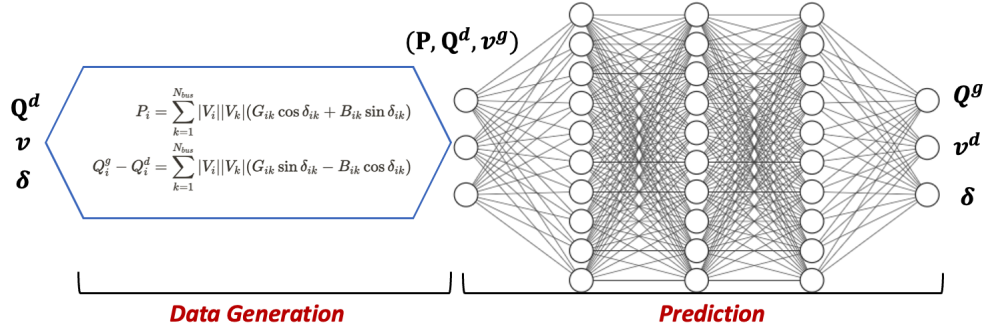


Figure 3.1: Generator-Predictor model to train solver

generation methods. Therefore, according to the UAT, NN can potentially approximate the ideal mapping, m , in a desired accuracy level. As shown in Figure 3.1, the procedure produces a trained network that approximates the hypothetical AC-PF solver function, i.e. $f \approx m$ that generates an optimal solution for any given input. This chapter argues that the pairs of $([P^g, v^g, P^d, Q^d], [Q^g, v^d, \delta])$ generated by solving the PF problem using a solver forms the same mapping, $m(\cdot)$, with a degree of error and a NN that is trained on solver-generated dataset is an approximator to this function whose performance is bounded by the solver accuracy.

3.1.3 Comparison of Newton Raphson-based Training and Solver-Free Approach

Figure 3.2 compares the predictions of trained NN models in a dataset of size d_d , constructed using both the proposed solver-free method and by solving NR repeatedly. To this end, an arbitrary problem to find the optimal solution x^* that satisfies $g(x^*) = b$ was chosen using a cubic function of $g(x) = x^3 + x$. For the solver-free method, $x \in [-2, 2]$ was

sampled d_d times, the corresponding values of $b = g(x)$ were obtained, and a single layer ReLU network with $d_h = 100$ was trained on the (b, x) pairs. For dataset construction using the NR method, $b \in [-10, 10]$ was randomly sampled d_d times and $g(x) = b$ was solved with the NR algorithm; then a NN with the same architecture as the solver-free method was trained on the collected pairs of (b, x) . Note that for both methods, boundary points (i.e. $\{-2, 2\}$ and $\{-10, 10\}$, respectively) were added to the training set manually to ensure the models were trained within a fixed domain. Both models are plotted against the true function in the figures, by generating x from a larger interval, $x \in \{-2.5, -2.49, \dots, 2.49, 2.5\}$ and finding the corresponding $b = g(x)$. The predictions were generated by $\hat{x}^* = f(b)$ for models trained with both approaches.

The figure presents the NN trained with the proposed approach in the first row and the data generated by NR method in the second row, and the sampled data as orange and gray points, respectively. As shown in the figure, by training for different b , both models approximate the same function, $g^{-1}(b)$, for the domain $b \in [-10, 10]$ with differences due to where the data were located. As explained earlier, a NN model with ReLU activation function essentially connects the given data points to approximate the solution. Hence, the difference between predictions and the actual function increases in regions where the dataset is sparse or unavailable and accuracy improves as d_d increases. Moreover, since data was sampled from a bounded region, the predictions are poor outside the training domain, $b > 10, b < -10$. This observation is in line with the UAT which guarantees performance for a bounded input set.

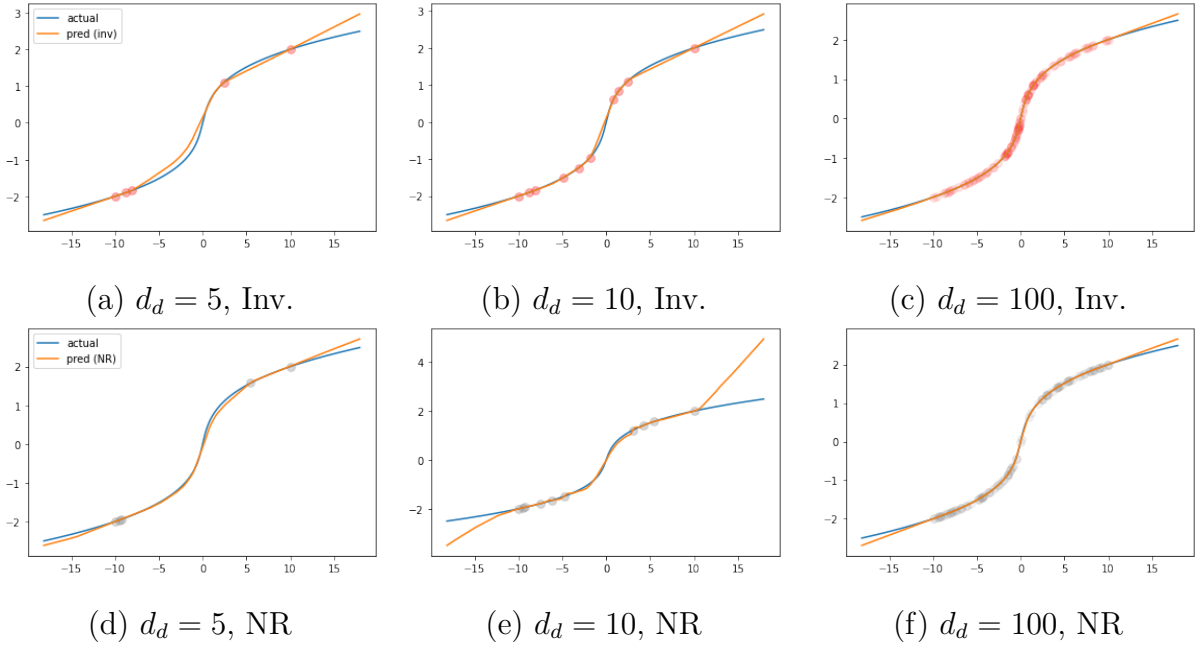


Figure 3.2: Comparison of two approaches to train a $NN(n_h = 1, d_h = 100)$ with varying d_d

3.1.4 Dataset size and Model Performance

While more data improves a model’s prediction capabilities, the improvement is limited by model complexity. ReLU NN models are piecewise linear (PWL) functions with the number of breaks determined by the number of hidden neurons and layers. Therefore, even if connecting all available data points would yield a good approximating function, the model may not have enough capacity to do so. Figure 3.3 illustrates test performances of ReLU networks with $d_h = 100, n_h = 1$ trained to model a multivariate quadratic function of the form $m(\mathbf{x}) = \mathbf{A}\mathbf{x}^2$. For each problem size, $d_p \in \{1, 2, 5, 10\}$, a NN was trained with $d_d = \{5, 100, 250, 500, 1000, 10000\}$ data points for 1000 epochs with batch size of 1024, validated on 1000 points and tested on 5000 points. Note that both axes are in log 10 scale. As shown, when $d_p = 1$ the improvement is very large until the $d_d = d_h = 100$ but slows thereafter. This is related to the fact that a NN with $d_h = 100$ neurons can represent a PWL function with 100 linear facets [79] and hence connect 101 data points. As problem size, d_p increases, the increasing dimensionality necessitates more training data points to achieve a better fit. That is, while a training set of 10000 points is enough to reduce the error to 0.001 for the problem with $d_p = 1$, more data is needed for larger problems.

It can be seen that model performance improves with the dataset size but reduces with the problem size. While this is an expected result, it is not clear how many data points is enough to reduce the error to an arbitrary ϵ threshold. Next section derives an upper bound to model error, assuming there is enough computational power to train large enough NN and generate arbitrarily large datasets.

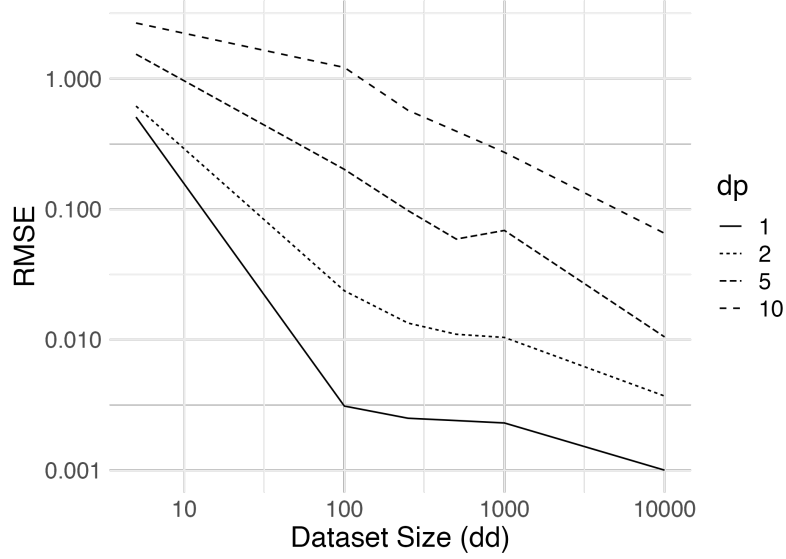


Figure 3.3: Average test RMSE of $NN(ReLU, n_h = 1, d_h = 100)$ on d_p dimensional quadratic function

3.1.5 Maximum Model Prediction Error on Nonlinear Functions

In this section, we present a framework to find a bound of NN prediction error on PF equations in (2.25) and deriving a relationship between the prediction error, problem size and dataset size. To estimate the worst-case scenario for the model error, we impose some assumptions on the model training, dataset construction, and the target function.

Our first assumption is on the target function, for which we choose the power flow equations in (2.25) assuming $V_i \approx 1$, which can be written as $g(\mathbf{x}) = [\mathbf{P}^{inj}, \mathbf{Q}^{inj}]^T$ where,

$$P_i^{inj} = \sum_{j=1}^n G_{ik} \cos(x_{ik}) + B_{ik} \sin(x_{ik}), \forall i \in [1, \dots, n], \quad (3.2)$$

$$Q_i^{inj} = \sum_{j=1}^n G_{ik} \sin(x_{ik}) - B_{ik} \cos(x_{ik}), \forall i \in [1, \dots, n]. \quad (3.3)$$

Secondly, we assume certain assumptions on the NN model. Since NNs are PWL approximators to continuous functions and PF equation is a nonlinear mapping, we first consider the case of a PWL approximation to PF equations that is obtained by connecting all training data with linear surfaces. This also means, the approximation error is closer to 0 around the training points but can increase on intervals that are further from those training points. As NNs are general forms of PWL functions, there exists at least one NN model, f_{pf} , such as in Figure 3.4a, that can exactly represent this function.

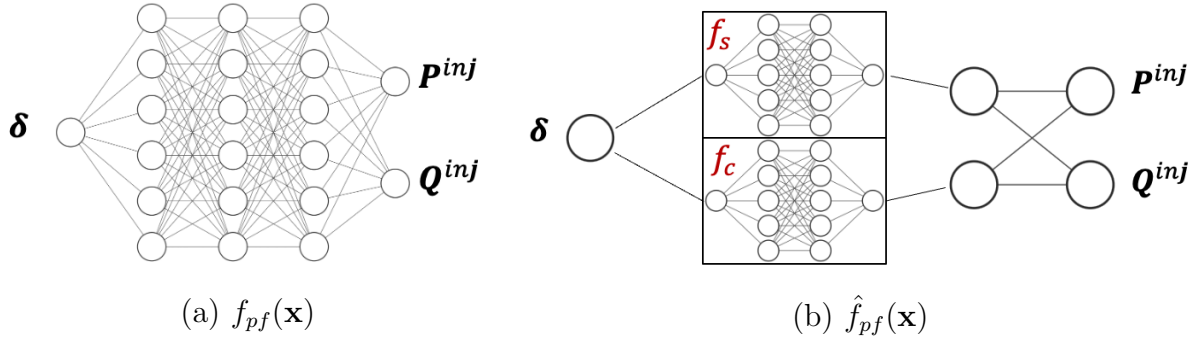


Figure 3.4: Two NN approximators to simplified PF equations in 3.3

By definition, $f_{pf}(\mathbf{x}) = g(\mathbf{x})$, $\forall \mathbf{x} \in \mathcal{D}$ where $\mathcal{D}$ is the training dataset. However, this model is hard to use for the purposes in this section. Therefore, we introduce a proxy to this NN, $\hat{f}_{pf}$ that has the same shape, i.e. connects all the training data with linear surfaces, which is illustrated in Figure 3.4b. Since $f_{pf}(\mathbf{x})$ is harder to estimating an error bound with, we define a proxy of $f_{pf}(\mathbf{x})$ that is also $\hat{f}_{pf}(\mathbf{x}) = g(\mathbf{x}) \forall \mathbf{x} \in \mathcal{D}$, which is defined

as

$$\hat{f}(\mathbf{x}) = \mathbf{W}^1 \mathbf{h}, \quad (3.4)$$

$$\mathbf{h} = [f_{\cos}(\mathbf{x}), f_{\sin}(\mathbf{x})]^T, \quad (3.5)$$

where

$$\mathbf{W}^1 = \begin{bmatrix} \mathbf{G} & \mathbf{B} \\ -\mathbf{B} & \mathbf{G} \end{bmatrix}, \quad (3.6)$$

and $f_{\sin} \approx \sin(\cdot)$ and $f_{\cos} \approx \cos(\cdot)$ are NN models in form of approximating $\sin(x)$ and $\cos(x)$, respectively under the same assumptions above, i.e. $f_{\sin}(x) = \sin(x)$ and $f_{\cos}(x) = \cos(x)$ for all x in the training set. Overall, $\hat{f}_{pf}(x)$ is a composite NN model whose first layer concatenates the outputs of $f_{\sin}(x)$ and $f_{\cos}(x)$. Under this formulation, $\hat{f}_{pf}(x)$ can also be expressed as $\hat{f}_{pf}(x_{ik}) = [\hat{P}_i^{inj}, \hat{Q}_i^{inj}]^T$ where

$$\hat{P}_i^{inj} = \sum_{j=1}^n G_{ik} f_{\cos}(x_{ik}) + B_{ik} f_{\sin}(x_{ik}), \forall i \in [1, \dots, n], \quad (3.7)$$

$$\hat{Q}_i^{inj} = \sum_{j=1}^n G_{ik} f_{\sin}(x_{ik}) - B_{ik} f_{\cos}(x_{ik}), \forall i \in [1, \dots, n]. \quad (3.8)$$

Finally, we assume that the dataset is sampled equidistantly within a fixed interval, $[a_i, b_i]$ for $i \in [1, \dots, n]$, where n is the input/output dimension of the function. If only boundary points are sampled for each dimension, then the dataset size is 2^n . If the middle points are also sampled for each dimension, then dataset size increases to 3^n and so forth.

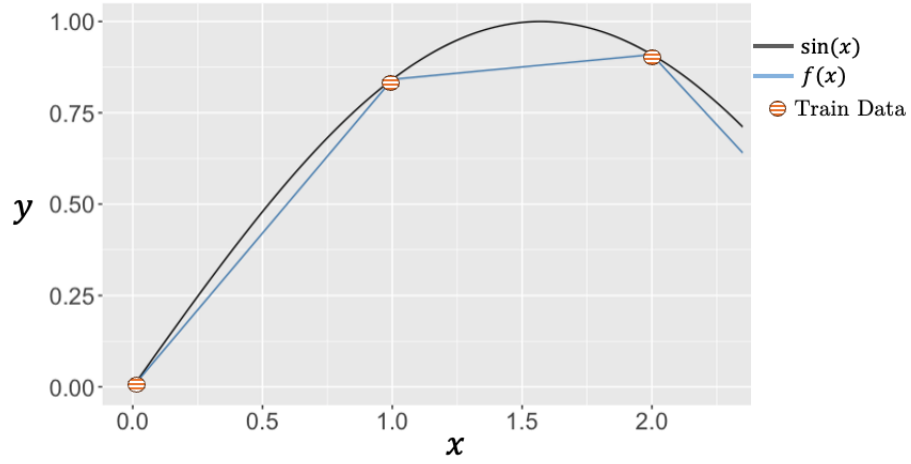


Figure 3.5: Illustration of $f_{\sin}(x)$ that minimize test $\text{MSE}=0$ against $\sin(x)$

Figure 3.5 illustrates how $f_{\sin}$ approximates $\sin(x)$, when trained on a dataset equidistantly sampled for $x = [0, 1, 2]$ and $y = [\sin(0), \sin(1), \sin(2)]$. As seen in the figure, $f_{\sin}(x)$ is a PWL function whose breaks are where the training examples are located and the model consists of linear facets between pairs of consecutive training examples.

Based on these assumptions, we introduce the following remark on the error bound.

Remark

Let $f_{pf}(\mathbf{x})$ be a minimal complexity NN model that approximates the simplified AC-PF equations in (3.3) by connecting all training points, $x_i \in \mathcal{D}$ with linear surfaces. Also, assume that $f_{pf}(x_i) = g(x_i) \forall x_i \in \mathcal{D}$ and the proxy model $\hat{f}_{pf} \sim f_{pf}$ such that $\hat{f}_{pf}(x_i) = g(x_i) \forall x_i \in \mathcal{D}$. Then the model prediction error is bounded by

$$\max |\hat{P}_i^{inj} - P_i^{inj}| = n \left[\epsilon_{\cos}^+ \sum_{j=1}^n G_{ik} + \epsilon_{\sin}^+ \sum_{j=1}^n B_{ik} \right], \quad (3.9)$$

$$\max |\hat{Q}_i^{inj} - Q_i^{inj}| = n \left[\epsilon_{\sin}^+ \sum_{j=1}^n G_{ik} - \epsilon_{\cos}^+ \sum_{j=1}^n B_{ik} \right], \quad (3.10)$$

where $\epsilon_{\sin}^+$ and $\epsilon_{\cos}^+$ as given in (3.19) and (3.25) are maximum approximation error of $f_{\cos}(x) \approx \cos(x)$ and $f_{\sin}(x) \approx \sin(x)$ over every interval of $\in [x_{i-1}, x_i]$.

To derive the error bounds in the above remark, the error between the true function, $g(x) = \sin(x)$ and the NN model predictions can be defined as

$$\epsilon_{\sin} = |f_{\sin}(x) - \sin(x)|. \quad (3.11)$$

Under the above assumptions, we can now define $f_{\sin}(x)$ as a PWL NN function, $f(x)$, as

$$f(x) = \begin{cases} W_1(x) + b_1 & \text{if } x_0 \leq x \leq x_1, \\ W_2(x) + b_2 & \text{if } x_1 \leq x \leq x_2, \\ \dots & \\ W_m(x) + b_m & \text{if } x_{m-1} \leq x \leq x_m, \end{cases} \quad (3.12)$$

where $x_i = x_{i-1} + \Delta$ as the data is sampled equidistantly. As the model can be decomposed into linear facets between every pair of consecutive points, the error for any interval $[x_i, x_i + \Delta]$ can be written as:

$$\epsilon_j = |W_j x + b_j - \sin(x)| \text{ for } x \in [x_i, x_i + \Delta]. \quad (3.13)$$

As the slope is constant in each interval, we can obtain

$$W_j = \frac{\sin(x_j + \Delta) - \sin(x_j)}{\Delta}, \quad (3.14)$$

Since bias is where function $f_j(x) = W_j x + b_j$ crosses y axis, i.e. $b_j = f_j(0)$, and the model value is equal to the target function value at the training points, i.e. $f_j(x_j) = \sin(x_j)$, we can write,

$$W_j = \frac{f_j(x_j) - f_j(0)}{x_j - 0} \Rightarrow b_j = f_j(0) = \sin(x_j) - W_j x_j. \quad (3.15)$$

Now, within any interval of $[x_{j-1}, x_j]$, the point that maximizes the error can be obtained

by calculating the derivative of the error with respect to x within the interval j and solving for x .

$$0 = \frac{d\epsilon_j}{dx} = \begin{cases} W_j - \cos(x) & \text{if } W_j x - \sin(x) \geq 0, \\ -W_j + \cos(x) & \text{if } W_j x - \sin(x) < 0. \end{cases} \quad (3.16)$$

Then, the point that maximizes the error in each interval and the maximal error can be calculated as

$$x_j^* = \cos^{-1}(W_j) = \cos^{-1} \left(\frac{\sin(x_j + \Delta) - \sin(x_j)}{\Delta} \right), \quad (3.17)$$

$$\epsilon_{j \sin}^* = |W_j x_j^* + b_j - \sin(x_j^*)|. \quad (3.18)$$

Then, maximum error is the maximum value calculated for all the $[x_i, x_i + \Delta]$ intervals, i.e.,

$$\epsilon_{\sin}^+ = \max(\epsilon_{1 \sin}^*, \epsilon_{2 \sin}^*, \dots, \epsilon_{m \sin}^*) \quad (3.19)$$

The maximum error for $f_{\cos} \approx \cos(x)$ can be calculated by following the same steps as above.

$$W_j = \frac{\cos(x_j + \Delta) - \cos(x_j)}{\Delta}, \quad (3.20)$$

$$b_j = \cos(x_j) - W_j x_j. \quad (3.21)$$

Within any interval of $[x_{j-1}, x_j]$, the point that maximizes the error can be obtained by

solving

$$0 = \frac{\partial \epsilon_j}{\partial x} = \begin{cases} W_j + \sin(x) & \text{if } W_j x - \cos(x) \geq 0 \\ -W_j - \sin(x) & \text{if } W_j x - \cos(x) < 0 \end{cases} \quad (3.22)$$

Then,

$$x_i^* = \sin^{-1}(-W_j) = \sin^{-1}\left(\frac{\cos(x_j) - \cos(x_j + \Delta)}{\Delta}\right), \quad (3.23)$$

$$\epsilon_{j \cos}^* = |W_j x_j^* + b_j - \cos(x_j^*)|, \quad (3.24)$$

$$\epsilon_{\cos}^+ = \max(\epsilon_{1 \cos}^*, \epsilon_{2 \cos}^*, \dots, \epsilon_{m \cos}^*). \quad (3.25)$$

Using the NN estimator to $\sin(x)$ and $\cos(x)$ defined above, the maximum error for P and Q injections in (3.10) can be calculated as:

$$\max |\hat{P}_i^{inj} - P_i^{inj}| = n \left[\epsilon_{\cos}^+ \sum_{j=1}^n G_{ik} + \epsilon_{\sin}^+ \sum_{j=1}^n B_{ik} \right], \quad (3.26)$$

$$\max |\hat{Q}_i^{inj} - Q_i^{inj}| = n \left[\epsilon_{\sin}^+ \sum_{j=1}^n G_{ik} - \epsilon_{\cos}^+ \sum_{j=1}^n B_{ik} \right], \quad (3.27)$$

$\forall i \in [1, \dots, n]$. Note that these estimates are the same for generator dispatches. For example, if the reactive power demand Q_i^d is known, then,

$$|\hat{Q}_i^{inj} - Q_i^{inj}| = |\hat{Q}_i^g - Q^d - (Q_i^g - Q_i^d)| = |\hat{Q}_i^g - Q_i^g|. \quad (3.28)$$

Based on these calculations, the maximum error estimates to P^{inj} and Q^{inj} are pre-

m	6-bus	14-bus	30-bus	57-bus	118-bus
1	4.71e-02	4.54e-02	4.65e-02	1.26e-01	1.72e-01
2	1.14e-02	1.10e-02	1.13e-02	3.07e-02	4.20e-02
3	5.07e-03	4.91e-03	5.05e-03	1.37e-02	1.87e-02
4	2.86e-03	2.77e-03	2.84e-03	7.73e-03	1.06e-02
5	1.83e-03	1.77e-03	1.82e-03	4.95e-03	6.76e-03
6	1.27e-03	1.23e-03	1.27e-03	3.44e-03	4.70e-03
7	9.37e-04	9.06e-04	9.31e-04	2.53e-03	3.45e-03
8	7.17e-04	6.94e-04	7.13e-04	1.94e-03	2.64e-03
9	5.67e-04	5.49e-04	5.63e-04	1.53e-03	2.09e-03
10	4.59e-04	4.45e-04	4.57e-04	1.24e-03	1.69e-03

Table 3.1: Maximum error estimates for $Q_i^g - \hat{Q}_i^g$, averaged for all i

sented in Figure 3.6 and Q^g in Table 3.1. The error estimates were calculated for $x \in [-0.1, 0.1]$ and $\Delta = 0.01$, and for each bus separately and averaged across the generator buses. As seen in the figure, the error reduces as the number of splits for each dimension, m , increases from 1 to 10, but increases as the system size increases. For example, for 30-bus system, the error estimate is 0.0209 when the number of split, $m = 1$ and is 0.0051 when $m = 2$. For same number of splits, errors increase to 0.0567 and 0.0139, for 57-bus system and to 0.0770, 0.0190 for 118-bus, respectively.

These results give insight into the relationship between the error, dataset and the system size. For example, if one can generate 50000 data points, for a 30-bus system with 6 generators, solving $m^6 = 50000$ for m yields $m \approx 6$, whereas for 57-bus system with 7 generators, solving $m^7 = 50000$ yields $m \approx 5$, and for 118-bus system with 10 generators, it yields $m \approx 3$. Therefore, training the models with 50000 data points can roughly translate to having 6, 5, and 3 splits per dimension for 30-, 57-, and 118-bus systems, respectively. Based on these calculations, one can expect maximum errors of approximately 5.71 E-04,

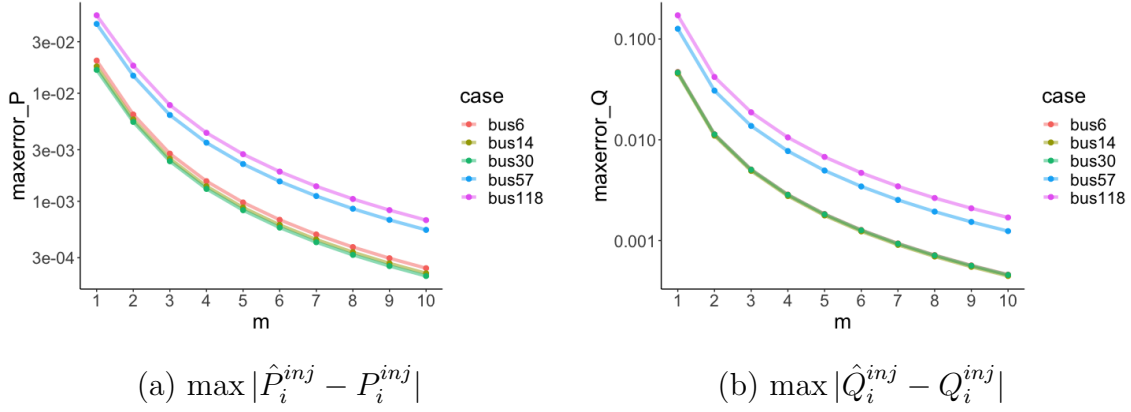


Figure 3.6: Maximum error estimates for P^{inj} and Q^{inj}

2.23 E-03, 8.44 E-03, respectively.

3.2 Numerical Results

Based on the above observations, this section presents results of experiments on IEEE 6, 14, 30, 57 and 118 bus systems. All of the experiments were conducted by training a NN with $d_h = 256$ trained with Adam optimizer, with $lr = .001$ and batch size of 1024. All training was carried out in Python 3.9 using PyTorch (v2.0.1) and Scipy (v1.11.2) was used for the alternative optimization algorithms¹.

3.2.1 Comparison of NN and Solver-Based Solutions

This section compares the performance of our inverse training approach (NN-inv) against solver based NN (NN-NR), and gradient based methods, Nelder-Maed, BFGS, and L-

¹NN models were trained on Digital Research Alliance of Canada remote servers. NN predictions and optimization were executed locally using Mac Mini M2 16GB.

BFGS-B. Training NN with the generator-predictor approach was carried out with 51200 training data point points and validated on 1000 data points and the model performing best on the validation set was chosen for the final model. For training with the solver generated dataset, a training set of $d_d = 5120$ and validation set of 1000 examples were generated using a L-BFGS-B solver. All models, including the optimization algorithms, were compared on the same test set of 4000 data points that were generated by randomly sampling $\mathbf{v} \in [0.95, 1.05]$ and $\delta \in [-0.15, 0.15]$, finding the corresponding $\mathbf{P}^g, \mathbf{P}^d, \mathbf{Q}^g, \mathbf{Q}^d$ and separating the variables into $\mathbf{P}^g, \mathbf{P}^d, \mathbf{Q}^d, \mathbf{v}^g$ as inputs and $\mathbf{Q}^g, \mathbf{v}^d, \delta$ as outputs. The optimization algorithms were run with default parameters implemented in the Python scikit library. Table 3.2 presents the MSE performance that was calculated between model predictions and true values combined (MSE_{test}) and separately for Q^g, v^d , and δ .

It can be seen that for all test systems with the exception of 6-bus, our NN (Inv) model outperforms the alternative approaches in all measurements. For the 6-bus system, NN (Inv) had the best overall MSE performance of 2.32e-4, followed by NR based methods with around 2.70E-4; BFGS and L-BFGS-B predicted Q^g with MSE of 1.542E-12 and 3.03E-9, compared to a MSE of 1.02E-4 for the proposed NN (Inv) approach. For the 14-, 30-, 57- and 118-bus systems, NN (Inv) produced predictions with 8.27E-5, 5.92E-5, 1.43E-4, 4.83E-3, respectively, while closest performance was recorded for L-BFGS-B and NN trained on L-BFGS-B generated dataset with MSE of 1E-2 to 1E-1. Similar performance differences can be observed for Q^g, v^d and δ .

Table 3.3 shows that the time to produce predictions is around 1E-2 seconds per 4000 test examples for NN models, which is not affected by the size of the system. On the other hand, L-BFGS-B produces output for 6-bus problem in 5.84E-02 seconds, which increases

to 3.48E+03 seconds for 57-bus system. Due to its computational cost, 118-bus results were calculated on a remote server and are not reported in the table.

Table 3.2 contains some unexpected results as well. As discussed earlier, NNs trained on solver generated datasets inherit the performance of the solver. The reported performance of NN (NR) closely follows L-BFGS-B and in some cases outperforms the solver with a small margin. For example, for 30-bus system overall MSE performance of NN (NR) is 1.42E-1 compared to L-BFGS-B that predicted with 1.40E-1. When the error is broken down to each variable, NN (NR) outperforms the solver on v_d and δ . For 14- and 30-bus systems NN (NR) outperforms L-BFGS-B on all measures with small differences. This can be attributed to NNs linearization behaviour that estimates a function by connecting all available training data, which in turn can have small variations and improvements over the solver that produced the training data.

Overall, the results show that NN (inv) outperforms the alternative approaches. Our model performance reduces as the system size increases; however this is also the case for optimization algorithms which makes it significantly harder to calculate with a desired accuracy. In some cases, solvers could predict largely better than our model (i.e. Q^g predictions on 6-bus). This seems to be related with nature of solvers that can improve the solutions iteratively until maximum iteration is reached, while accuracy of NN (inv) depends on the data density in a given region. Our training approach allows for scaling up the data density arbitrarily and opens up the potential to such levels. On the other hand, when NNs are trained on solver generated datasets, the model performance is limited by how the data is generated.

The main advantage of inverse training is to bypass the solvers, which eliminates any solver approximation error embedded in the dataset. Error-free dataset generation uncovers the true potential of NN to achieve the performance proposed by the UAT, whereas NNs trained on solver generated datasets are bounded by the optimization error.

Bus	Model	MSE_{test}	Q_g (p.u.)	v_d (p.u.)	θ (rad.)
6	NN (Inv)	2.32e-04	1.02e-04	4.94e-06	4.11e-04
	NN (NR)	6.46e-04	1.47e-03	2.03e-05	5.47e-04
	Nelder Mead	2.95e-01	1.17e+00	1.84e-03	3.30e-03
	BFGS	2.70e-04	1.42e-12	5.36e-15	5.41e-04
	L-BFGS-B	2.70e-04	3.03e-09	2.93e-12	5.41e-04
14	NN (Inv)	8.27e-05	1.06e-04	5.68e-06	1.24e-04
	NN (NR)	1.37e-01	6.50e-01	4.01e-02	1.63e-02
	Nelder Mead	2.91e-01	1.62e+00	9.35e-04	3.34e-03
	BFGS	2.81e-03	1.04e-02	7.84e-04	1.38e-03
	L-BFGS-B	1.41e-01	6.54e-01	4.68e-02	1.78e-02
30	NN (Inv)	5.92e-05	1.29e-04	1.09e-05	7.97e-05
	NN (NR)	1.05e-01	1.87e-01	5.25e-02	1.10e-01
	Nelder Mead	1.87e-01	1.59e+00	6.45e-04	3.34e-03
	BFGS	9.35e-03	2.76e-02	8.99e-03	5.37e-03
	L-BFGS-B	6.70e-02	3.56e-01	6.07e-02	4.41e-03
57	NN (Inv)	1.43e-04	1.23e-03	5.66e-05	8.55e-05
	NN (NR)	1.85e-01	2.76e+00	3.06e-02	3.81e-03
	Nelder Mead	1.04e+00	1.58e+01	1.59e-01	3.36e-03
	BFGS	5.32e-02	6.07e-01	2.06e-02	1.37e-02
	L-BFGS-B	1.13e-01	1.52e+00	3.80e-02	5.85e-03
118	NN (Inv)	4.83e-03	1.95e-02	6.37e-04	3.74e-04
	NN (NR)	2.46e+00	1.07e+01	7.62e-03	1.95e-03
	L-BFGS-B	2.44e+00	1.07e+01	7.24e-03	8.11e-04

Table 3.2: Comparison of our model against alternative approaches on test systems

Table 3.3: Time (s) to calculate 4000 test set predictions for each model

Bus	L-BFGS-B	Nelder Maed	NN (SF)	NN (NR)
6-bus	5.84E+01	1.31E+01	1.40E-02	5.56E-03
30-bus	1.74E+03	2.28E+01	4.56E-03	1.34E-02
57-bus	3.48E+03	7.64E+03	1.63E-02	2.21E-02
118-bus	-	-	1.98E-02	1.69E-02

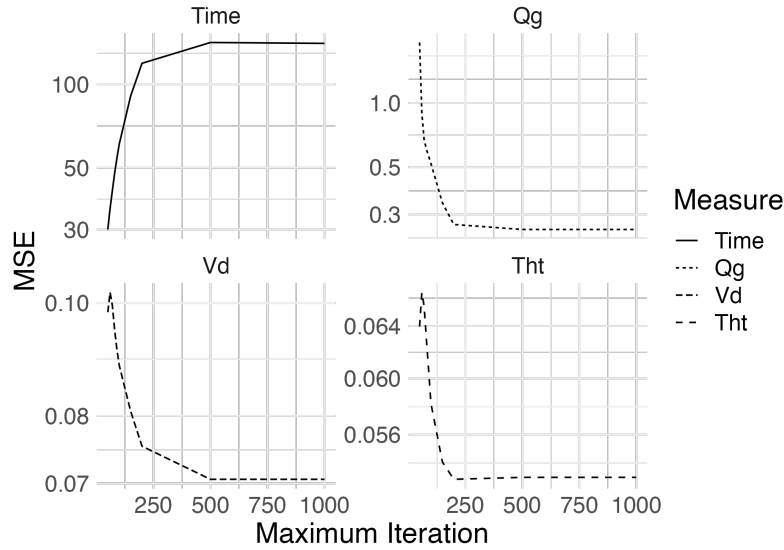


Figure 3.7: Maximum iteration on solver performance

3.2.2 Maximum Iteration on Optimization Performance

While NNs suffer from larger systems, it is also the case for solvers that require longer optimization time and more computational cost as the system size increases. This section presents a small experiment on finding solutions to 300 randomly generated 30-bus system ACPF problems using L-BFGS-B method. The optimization algorithm was implemented in scipy 1.11.2 and Python 3.9.18. The default parameters of the model terminates when tolerance = $1\text{E-}5$ or maximum iteration = 100 is reached. As the implementa-

tion does not allow for changes in tolerance, the maximum iteration was chosen from $\{40, 50, 60, \dots, 100, 150, 200, 500, 1000\}$. Initial parameters were set to $V_d = 1, Q_g = 1$ and $\delta = 0$.

The plots in Figure 3.7 shows that as maximum iteration increases, the model accuracy increases along with the solution time. It takes around 500 iterations to reduce the error on voltage magnitudes to $\text{MSE}=0.07$ and 300 iterations to reduce the error of Q_g and δ to 0.26 and 0.05 respectively. After these points the error does not reduce further and the algorithm terminates. Note that the solution time increases dramatically for larger systems for a test size of 300 examples.

3.3 Chapter Summary

The analysis in this chapter is motivated by the UAT that proves existence of NN that can achieve any error level with global guarantees. Our study experiments on arbitrary functions to understand the effect of dataset size, model complexity, problem size and nonlinearity on NN performance, and proposes a generator-predictor approach for training NNs to solve AC-PF problems that can train NNs with high quality and arbitrarily large training sets without using solvers to construct the dataset. Our model outperforms regular NN training on solver based datasets as well as quasi NR solvers on all test systems. It is shown that our model can achieve better performances by randomly generating more training inputs without the need for dataset construction, unlike regular NN training whose accuracy is limited by the performance of computationally costly solvers.

Chapter 4

Semi-Supervised Neural Network Model For Multiparametric Quadratic Programming

4.1 Introduction

This section investigates how NNs produce solutions to a common optimization problem, Quadratic Programming (QP) with linear constraints, and proposes a NN architecture and training procedure that results in NNs generating optimal and feasible solutions. The method is motivated mainly by (i) the analysis in Section [2.3.1](#) that illustrates NNs are a general form of a PWL function that can estimate any function of the same form with an optimal parameter set, and (ii) the observation that the QP solution function is also

a PWL function with respect to the RHS values of the equality constraints. While the fact that NNs are PWL functions is well-documented in the literature, the latter part of our motivation is often overlooked. Therefore, this section starts with an analysis showing that QP solutions form a PWL function and provides foundational proofs to support the argument.

Our approach largely differs from the literature based on its premise to use NN model to exactly represent the solution function. Therefore, there is no clear state-of-the-art for us to compare our method. However, most studies report larger errors than our results reported in Chapters 4 and 5, which are measured with optimality gaps of $1e-6$ to $1e-10$ and squared KKT violations less than $1e-8$. The MP literature focuses primarily on methodology and performances are not explicitly reported [56, 57]. On the other hand, PINN model trained to solve DC-OPF problem reports 0.12% to 3.11% KKT violations [80], while the NeuralDeco model reports 0.5% optimality gap in their solutions [23]. DC3 model trained to solve quadratic problems reports 6.219 (0.098% in percentage) optimality gap. Note that this model further processes the results using gradient-descent operations when the predictions do not satisfy the feasibility conditions.

This study contributes to the literature as follows:

- justifying that classical training of NN models cannot yield the NN model that represents the solution function to mp-QP problem with linear constraints without error.
 - proposing a semi-supervised NN model that can provide optimal solutions to QP with linear constraints with uncertain parameters on the RHS of equality constraints.
- The model is based on an explainable NN architecture that directly aligns with the

piecewise linearity of the underlying optimization problem, and is supported by a training approach that decomposes the learning process to subproblems to overcome the training challenges.

- The modeling approach is applied to a QP problem, DC-OPF with generator limits, and provides precise predictions that satisfy KKT optimality and feasibility conditions. The model can rapidly create a distribution of optimal and feasible solutions to a large dataset sampled from the empirical distribution of uncertain demand and renewable generator dispatches.

The outline of this chapter is as follows: The next section illustrates the challenge of training a black-box NN model to represent the solution function exactly. Section 4.2 discusses how QP solutions can be decomposed to linear and PWL parts and presents proofs to support the argument. Later, the section details our proposed NN architecture that provides exact solution to the linear and PWL functions and combines both in one NN flow. Section 4.3 details model training and compares our model performance against alternative approaches on test cases. Section 4.4 discusses some of the challenges that prevent the model from scaling up to larger problems and Section 4.5 concludes the chapter.

4.1.1 The Tradeoff Between Dataset Size and Model Complexity

As our goal in subsequent sections is to train a NN model that represents the solution function with no error, we conducted preliminary analysis to determine whether a NN can be trained to represent a target function via standard training. Specifically, we investigated whether it is possible to train a DNN on a data sampled from a target function.

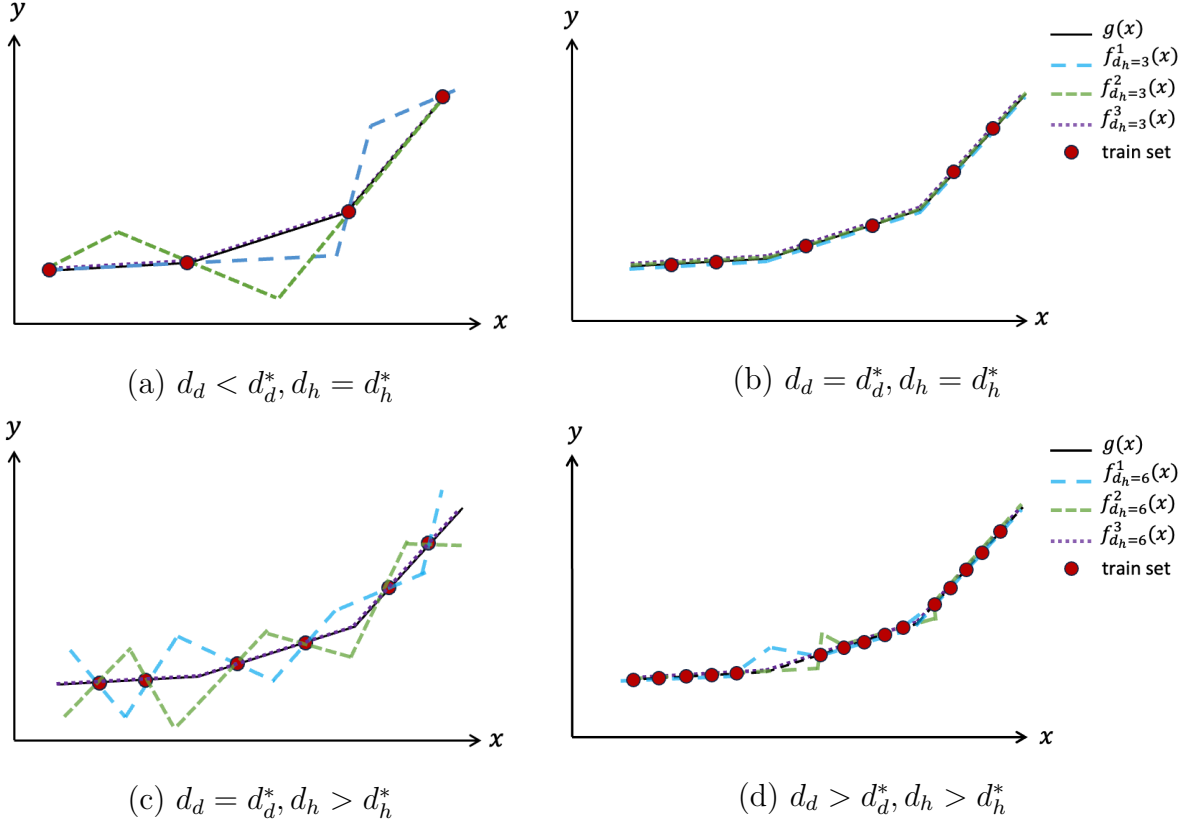


Figure 4.1: The tradeoff between dataset size and model complexity in estimating arbitrary PWL functions

In this section, we demonstrate why traditional NN training practices are not effective for finding optimal model weights and biases that can result in an exact representation of the target function, i.e. producing zero train and test error. Specifically, our analysis shows that there is a minimal model complexity and dataset size required to accurately estimate the solution function exactly. However, if a more complex model than minimal complexity is trained, it becomes practically impossible to represent the target function exactly and the model error can only be reduced with larger dataset size.

Figure 4.1 illustrates the trade-off between size of the dataset and the complexity of the model. The target function in the figures, $g(x)$, consists of three linear segments with positive slopes and is estimated by three shallow NN models. Based on the discussion in Section 2.3, a shallow NN model with $d_h = 3$, i.e. $f_{d_h=3}(x)$ is sufficient to represent a PWL function consisting of three linear segments exactly.

In Figure 4.1a, three models $f_{d_h=3}^1(x)$, $f_{d_h=3}^2(x)$ and $f_{d_h=3}^3(x)$ are trained on dataset of 4 data points consisting of boundary points of each linear segments. The figure illustrates that all three models can reduce the training error to zero by passing through the training points, but only one of them represents the true function $g(x)$. Notice that there are infinitely many different NN models that can zero the training error.

On the contrary, in Figure 4.1b, two data points are sampled for each linear segment. This is enough to limit the model parameter space to a unique set of weights and hence guarantee that the trained NN models represent $g(x)$ exactly. Notice that simply sampling 6 data points from $g(x)$ is not enough to guarantee to represent the target function exactly. This can be seen from the case where all data points are sampled from the first segment only, in which case the model would not have any information to represent the rest of the function.

While the common practice in ML is to train more complex models than necessary, this approach has some drawbacks when the target function is deterministic. In Figure 4.1c, three models with $d_h = 6$ are trained on the same data points as in Figure 4.1b. Although each model can represent a PWL function consisting of up to 6 linear segments, they are unable to zero the test error because there are infinitely many ways for the model

parameters to be adjusted to minimize the MSE, as illustrated in Figure 4.1a. In this case, one needs to increase the number of data points sampled from each segment, here to 5, to reduce the training error, however even this would not guarantee the model to represent the target function.

There are common practices in ML literature to limit the model variance with further measures, such as terminating training if the validation loss no longer reduces or using regularization terms. When the target function is deterministic, early stopping is not a viable option to represent the target function exactly because such a model would not reduce the training error to zero. It can be even better if the validation set examples were added to the training set and overfit the function as essentially the goal is to fit the target function as possible as it is. On the other hand, using regularization with L1 penalty term can potentially reduce the model complexity to the minimal model complexity with a proper choice of penalty parameter, in the later sections, in Section 4.2 we propose an approach that will set the model complexity to the minimal by design.

4.2 A General Framework to Solve QP Using NN

In general form, mp-QP with linear constraints can be written as,

$$\text{Minimize } z(\boldsymbol{\theta}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + (\mathbf{C} + \boldsymbol{\theta}_c)^T \mathbf{x} + \mathbf{C}_0, \quad (4.1a)$$

$$\text{s.t. } \mathbf{A}_e \mathbf{x} = \mathbf{b}_e + \boldsymbol{\theta}_e, \quad [\lambda] \quad (4.1b)$$

$$\mathbf{A}_c \mathbf{x} \leq \mathbf{b}_c + \boldsymbol{\theta}_c, \quad [\mu] \quad (4.1c)$$

where $\mathbf{x}, \mathbf{C}, \mathbf{C}_0 \in \mathbb{R}^n, \mathbf{b}_e \in \mathbb{R}^{m_1}, \mathbf{b}_c \in \mathbb{R}^{m_2}, \mathbf{Q} \in \mathbb{R}^{n \times n}$ is a semi-positive definite matrix, and $\mathbf{A}_e \in \mathbb{R}^{m_1 \times n}, \mathbf{A}_c \in \mathbb{R}^{m_2 \times n}$ and $\boldsymbol{\theta} \in \Theta_f$, where Θ_f is the set of all feasible parameters. The problem is parametrized by $\boldsymbol{\theta} = [\boldsymbol{\theta}_c, \boldsymbol{\theta}_e, \boldsymbol{\theta}_c]$ for $\boldsymbol{\theta}_c \in \mathbb{R}^n, \boldsymbol{\theta}_e \in \mathbb{R}^{m_1}, \boldsymbol{\theta}_c \in \mathbb{R}^{m_2}$. Here, the subscript $\mathcal{C}$ in $\mathbf{A}_c$ and $\mathbf{b}_c$ indicates the set of all inequality constraint indices, i.e. $\mathcal{C} = \{1, 2, \dots, m_2\}$.

As derived in Section 2.2.1, the optimal solutions to the (4.1) can be attained via the solution function,

$$g(\boldsymbol{\mu}^*, \boldsymbol{\theta}) = \mathbf{J}^{-1} \begin{bmatrix} -\mathbf{C} - \boldsymbol{\theta}_c + \mathbf{A}_c^T \boldsymbol{\mu}^* \\ -\mathbf{b}_e - \boldsymbol{\theta}_e \end{bmatrix}, \text{ for } \mathbf{J} = \begin{bmatrix} 2\mathbf{Q} & -\mathbf{A}_e^T \\ -\mathbf{A}_e & 0 \end{bmatrix}, \quad (4.2)$$

where $\boldsymbol{\mu}^* = \mu(\boldsymbol{\theta})$ is the optimal values of the shadow prices of inequality constraints calculated by an unobserved function, which will be modelled by a NN, $f_\mu(-\mathbf{B} - \boldsymbol{\theta})$ as described in the next section.

Assuming that the NN model $f_\mu \approx \mu$ is already trained, the solution function can be represented exactly by a nested NN model, as shown in Figure 4.2. The model takes right hand side vectors and problem parameters as input, which is written as

$$\mathbf{B} + \boldsymbol{\theta} = \begin{bmatrix} \mathbf{C} \\ \mathbf{b}_e \\ \mathbf{b}_c \end{bmatrix} + \begin{bmatrix} \boldsymbol{\theta}_c \\ \boldsymbol{\theta}_e \\ \boldsymbol{\theta}_c \end{bmatrix}. \quad (4.3)$$

The model calculates predictions in two steps. First, optimal shadow prices of inequality constraints are calculated as $\boldsymbol{\mu}^* = f_\mu(-\mathbf{B} - \boldsymbol{\theta})$, which are carried to the output as a part of

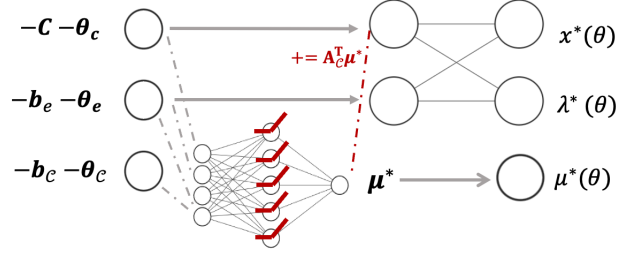


Figure 4.2: The QP Solver Model Architecture

the final predictions. Second, inputs are adjusted by $\mathbf{A}_c \boldsymbol{\mu}^*$ and the rest of the solutions are obtained via a linear mapping that is already expressed in 4.2. To represent this function as a NN, a linear model in the form of $f_g(\mathbf{z}) = \mathbf{W}\mathbf{z}$ is used where $\mathbf{W} = \mathbf{J}^{-1}$. The next section discusses how to train $f_\mu(\cdot)$ function.

4.2.1 Predicting Active Constraints and Shadow Prices

As shown in (2.14), estimating the solution of the optimization problem (4.1a)-(4.1b) requires knowing the value of $\boldsymbol{\mu}^*$. Therefore, in this section, a methodology to produce exact predictions of $\boldsymbol{\mu}^*$ and hence the active constraints is presented.

Theorem 2 and Lemma 1 in Section 2.2.1 show that while $g(\cdot)$ in (4.2) is a linear mapping, the composite function $g(\mu(\boldsymbol{\theta}), \boldsymbol{\theta})$ is a PWL function due to the same characteristics of $\mu(\boldsymbol{\theta})$. To illustrate, consider an optimization problem in the form defined in (4.1), with 5 decision variables, x_i , three equality constraints, and three upper and lower bound constraints; i.e. $x_i^- \leq x_i \leq x_i^+$ for $i \in [0, 1, 2]$. As will be shown in Numerical Results Section, this is our smallest 6-bus DC-OPF application case defined in (4.5). Here, the problem is solved using a Gurobi solver (v10.0.3) for $\theta_1 \in [0, 0.1, 0.2, \dots, 5]$ controlling RHS of equality

constraints $\mathbf{b}_e + \boldsymbol{\theta}_e = [\theta_1, .001, .001]$. In Figure 4.3, the optimal dual variables are plotted against different values of θ_1 . μ_i for $i \in [0, 1, 2]$ and μ_i for $i \in [3, 4, 5]$ are shadow prices of the upper and lower limits (i.e. the limits of power generators in the example), respectively. The slope changes in $\boldsymbol{\mu}^*$ exhibit a piecewise affine pattern, with different slopes in different critical regions ($\mathbf{CR}$), corresponding to different combinations of active constraints. In $\mathbf{CR}_0$ two lower limit constraints are active with corresponding $\mu_i > 0$. As θ_1 increases, generator lower limits stop binding and upper limits start to bind.

While $\mu(\boldsymbol{\theta})$ is piecewise linear and can be modeled with a generic NN, training a model on a dataset consisting of $(\boldsymbol{\theta}, \boldsymbol{\mu}^*)$ pairs does not guarantee optimality of the predictions for test data sampled outside the training distribution. To overcome this difficulty, our proposed method directly injects some of the information that is general to all $\boldsymbol{\theta} \in \Theta_f$, into the first layer of the model. Specifically, slopes of each linear facet of $f_\mu(\cdot)$ within a given CR_i are calculated and fixed as first layer weights of the NN model prior to the training. The rest of the model parameters (biases and second-layer weights) are estimated through standard training. Figure 4.4 illustrates the architecture of this model where $\mathbf{W}^0$ indicates the weights injected for the first layer, obtained from $\nabla \boldsymbol{\mu}_{\mathcal{B}}^*$ as follows:

Calculation of $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^*$: To obtain the slopes of $\boldsymbol{\mu}^*$ with respect to the changes in $\boldsymbol{\theta}$ for binding constraints, $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^*$, we expand the $\mathbf{J}$ matrix defined in (2.13) with the rows of the coefficient matrix of inequality constraints corresponding to the binding constraints, $\mathbf{A}_{\mathcal{B}_i}$,

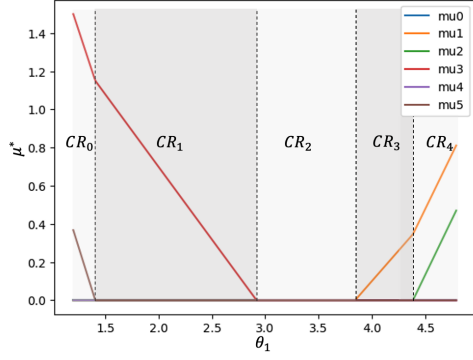


Figure 4.3: Sensitivity of μ against θ_1

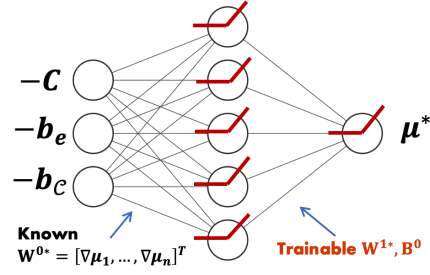


Figure 4.4: NN Model to Predict μ^*

and then invert it.

$$\begin{bmatrix} \mathbf{x}^* \\ \lambda^* \\ \mu_{\mathcal{B}_i}^* \end{bmatrix} = \mathbf{J}_{\mathcal{B}_i}^{-1}(-\mathbf{B} - \boldsymbol{\theta}) = \begin{bmatrix} \nabla \mathbf{x}^* \\ \nabla \lambda^* \\ \nabla \mu_{\mathcal{B}_i}^* \end{bmatrix} \begin{bmatrix} -\mathbf{C} - \boldsymbol{\theta}_c \\ -\mathbf{b}_e - \boldsymbol{\theta}_e \\ -\mathbf{b}_{\mathcal{B}_i} - \boldsymbol{\theta}_{\mathcal{B}_i} \end{bmatrix}, \text{ where } \mathbf{J}_{\mathcal{B}} = \begin{bmatrix} 2\mathbf{Q} & -\mathbf{A}_e^T & -\mathbf{A}_{\mathcal{B}_i}^T \\ -\mathbf{A}_e & 0 & 0 \\ -\mathbf{A}_{\mathcal{B}_i} & 0 & 0 \end{bmatrix}. \quad (4.4)$$

Notice that the last row of the above inverse Jacobian matrix, $\mathbf{J}_{\mathcal{B}_i}^{-1}$, is the gradient vector $\nabla \mu_{\mathcal{B}_i}^*$ that contains the coefficients of the linear facet of the target PWL function when a certain set of constraints, $\mathcal{B}_i$, are binding. In other words, the vector is the set of coefficients that maps $\mathbf{B}$ to $\mu_{\mathcal{B}_i}^*$.

After the first layer of the NN is fixed with all potential derivatives, the training is carried out to find when each slope is activated. However, the number of potential slopes increases exponentially with the addition of inequality constraints, which adds more burden on the pre-training step as matrix inversion is required to calculate the slopes and also increases the number of parameters to estimate in the training step. The next section

outlines a method to calculate $\nabla \mu_{\mathcal{B}_i}$ without matrix inversion and a method to filter active constraints to enhance the execution time of the solution methodology.

4.2.2 Filtering Active Constraints and Data Generation

In many practical mp-QP applications, characteristics of the problem imply that most constraint combinations can never occur, therefore a method for filtering the unused combinations is needed prior to training the model. For example, in the previous 6-bus DC-OPF example with 6 inequality constraints, the set of all constraint combinations is $\{\{0\}, \{1\}, \{2\}, \{0, 1\}, \dots, \{0, 1, 2, 3, 4, 5\}\}$ with $2^6 = 64$ elements, yet 59 of these combinations never occur. In the present study, to identify the active combinations we use a commercial solver to solve the problem for a wide range of input parameters. Specifically, the RHS vector is set to $\mathbf{b}_e = \mathbf{0}$ and an arbitrary dimension is incrementally increased from zero to the highest feasible point. Then, after solving for each input demand, the set of potential binding constraints is identified by checking whether the associated μ_i is greater than zero for any of the inputs.

Using a solver to find potential binding constraints also gives insight into the boundaries of critical regions, $[\mathbf{CR}_i^-, \mathbf{CR}_i^+]$ corresponding to $\mathcal{B}_i$ that is active at solution. This information is used to populate data without using solvers, by sampling random inputs from $[\mathbf{CR}_i^-, \mathbf{CR}_i^+]$ for all i , obtaining the matrices $\mathbf{J}_{\mathcal{B}_i}^{-1}$ and $\mathbf{b}_{\mathcal{B}_i}$ following (4.4), and calculating the corresponding optimal solutions.

The training procedure can be summarized as follows:

1. Generate RHS and solve with a solver: (i) Initiate the RHS with an initial point,

- (ii) Choose a θ_i and incrementally increase its value from 0 to sum of all generator capacities.
2. Solve the problem and find $\mathbf{CR}_i$ and $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^*$: (i) Solve mp-QP for each generated data using a solver, (ii) Find the sets of active constraints, i.e. $\mathcal{B}_i = \{ j \mid \mu_j > 0 \}$, (iii) Expand the coefficient matrix and calculate $\mathbf{J}_{\mathcal{B}_i}^{-1}$ for all binding constraints. Obtain $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^*$.
 3. Populate data: Randomly generate d_d number of RHS vectors using $Uniform(\mathbf{CR}_i^-, \mathbf{CR}_i^+)$ and find optimal solutions following (4.4). Check if KKT is satisfied for each data and regenerate data if needed.
 4. Train f_μ : (i) Initiate $f_\mu(\cdot)$ with random model parameters, fix $\mathbf{W}^0 = [\boldsymbol{\mu}_{\mathcal{B}_1}^*, \dots, \boldsymbol{\mu}_{\mathcal{B}_n}^*]^T$ and set with random weights, (ii) Train the $f_\mu(\cdot)$ with 1000 datapoints populated using the procedure described above, (iii) Define solver NN by setting first part to f_μ and second part to $\mathbf{J}^{-1}$.

4.3 Numerical Results

For proof of concept, we apply our method on the DC-OPF problem with generator limits, which can be defined as:

$$\underset{\mathbf{P}_g, \boldsymbol{\theta}}{\text{Minimize:}} \quad \mathbf{P}_g^T \mathbf{Q} \mathbf{P}_g + \mathbf{C}^T \mathbf{P}_g + \mathbf{C}_0, \quad (4.5a)$$

$$\text{s.t.} \quad \mathbf{P}_d + \boldsymbol{\theta}_e - \mathbf{B}\boldsymbol{\delta} - \mathbf{P}_g = 0, \quad [\boldsymbol{\lambda}] \quad (4.5b)$$

$$\mathbf{P}_g - \mathbf{P}_g^+ \leq 0, \quad [\boldsymbol{\mu}^+] \quad (4.5c)$$

$$\mathbf{P}_g^- - \mathbf{P}_g \leq 0, \quad [\boldsymbol{\mu}^-] \quad (4.5d)$$

where $\mathbf{C}, \mathbf{C}_0 \in \mathbb{R}^{n_g}$ are cost coefficient vectors, $\mathbf{Q} \in \mathbb{R}^{n_g \times n_g}$ is the quadratic cost coefficient matrix, $\mathbf{P}_g^-, \mathbf{P}_g^+ \in \mathbb{R}^{n_g}$ are lower and upper generator limits, and n_b, n_g are the number of buses and generator buses, respectively. The primal variables are the production output of dispatchable generators, $\mathbf{P}_g \in \mathbb{R}^{n_g}$, and voltage magnitudes of each bus, $\boldsymbol{\delta} \in \mathbb{R}^{n_b}$. Following the steps in Section 4.2, we write the Lagrangian function in (2.9) for the DC-OPF problem as below.

$$\begin{aligned} L(\mathbf{P}_g, \boldsymbol{\delta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = & \mathbf{P}_g^T \mathbf{Q} \mathbf{P}_g + \mathbf{C}^T \mathbf{P}_g + \mathbf{C}_0 + \\ & \boldsymbol{\lambda}(\mathbf{P}_d + \boldsymbol{\theta}_e - \mathbf{P}_g - \mathbf{B}\boldsymbol{\delta}) + \boldsymbol{\mu}^+(\mathbf{P}_g - \mathbf{P}_g^+) + \boldsymbol{\mu}^-(\mathbf{P}_g^- - \mathbf{P}_g). \end{aligned} \quad (4.6)$$

where $\boldsymbol{\theta}_c = 0, \boldsymbol{\theta}_C = 0$. The partial derivatives of the Lagrangian function is

$$\frac{\partial L}{\partial \mathbf{P}_g} = 2\mathbf{Q}\mathbf{P}_g + \mathbf{C} - \boldsymbol{\lambda} + \boldsymbol{\mu}^+ - \boldsymbol{\mu}^- = 0, \quad (4.7a)$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{B}^T \boldsymbol{\lambda} = 0, \quad (4.7b)$$

$$\frac{\partial L}{\partial \boldsymbol{\lambda}} = \mathbf{P}_d + \boldsymbol{\theta}_e - \mathbf{P}_g - \mathbf{B}\boldsymbol{\delta} = 0, \quad (4.7c)$$

$$\frac{\partial L}{\partial \boldsymbol{\mu}^+} = \mathbf{P}_g - \mathbf{P}_g^+ = 0, \quad (4.7d)$$

$$\frac{\partial L}{\partial \boldsymbol{\mu}^-} = \mathbf{P}_g^- - \mathbf{P}_g = 0. \quad (4.7e)$$

The right hand side vectors translate into the demand vector, $\mathbf{b}_e = \mathbf{P}_d$ for equality constraints, and the generator limits, $\mathbf{b}_C^T = \mathbf{P}^{\text{lim}} = [\mathbf{P}_g^+, -\mathbf{P}_g^-]$ for the inequality constraints, and the primal variables are $\mathbf{x}^T = [\mathbf{P}_g, \boldsymbol{\delta}]$.

As illustrated in Figure 4.4, we use a semi-supervised NN to predict the optimal solution through two sequential subnetworks, consisting of a shallow ReLU network f_μ model that calculates $\boldsymbol{\mu}^*$, and a separate NN that models $g(\boldsymbol{\mu}^*; \boldsymbol{\theta})$ in (2.13). Figure 4.5 illustrates the subnetwork that predicts the optimal dual variables of the inequality constraints, $\boldsymbol{\mu}^*$. As discussed earlier, the weights of the first layer of this subnetwork model are fixed with $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^*$ vectors for all potentially binding constraints $i \in \mathcal{C}$. Initially, each of these slope coefficient vectors have different sizes, depending on how many constraints are binding, but they are modified to have the same size by adding columns or rows containing zeros. For example, if there are 4 decision variables, 1 equality and 3 inequality constraints, only first constraint is binding, i.e. $\mathcal{B}_i = [1]$ and $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^* = \mathbf{v}$ then $\mathbf{v} \in \mathbb{R}^{6 \times 1}$ and therefore the

slope is adjusted to

$$\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^* = \begin{bmatrix} \mathbf{v} & 0 & 0 \\ \mathbf{0} & 0 & 0 \\ \mathbf{0} & 0 & 0 \end{bmatrix} \quad (4.8)$$

where $\mathbf{0} \in \mathbb{R}^{6 \times 1}$ is a vector of zeroes in the same size as $\mathbf{v}$. Here the zeroes represent the change in shadow prices with respect to changes in input parameters for nonbinding constraints.

After all $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^*$ are calculated and standardized into the same size, first layer weights are obtained simply by stacking all $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^*$ vertically. The rest of the parameters, weights of the second layer, and biases, are determined via training. Finally, the second subnetwork is a mapping $g(\boldsymbol{\mu}^*; \boldsymbol{\theta}) = [\mathbf{P}_g^*, \boldsymbol{\delta}^*, \boldsymbol{\lambda}^*]$, which can be defined as a linear model (one layered NN) without training, by fixing its weights to $\mathbf{W} = \mathbf{J}^{-1}$ defined in (2.13). Our SSNN model combines the two subnetworks in one NN flow as shown in Figure 4.6. The NN model, f_μ , takes $\mathbf{B} + \boldsymbol{\theta} = [\mathbf{C}, 0, \boldsymbol{\theta}_e, -\mathbf{P}^+, \mathbf{P}^-]^T$ as inputs and produces $\boldsymbol{\mu}^*$. For simplicity, we set $\mathbf{P}_d = 0$ and define $\boldsymbol{\theta}_e \geq 0$ as the demand of the system. The second subnetwork takes the adjusted inputs to produce the optimal solutions.

4.3.1 Test Results

Our method was evaluated using IEEE 6-, IEEE 30-, and IEEE-57-bus test systems. We used our SSNN model to predict optimal solutions to the DC-OPF problem, and compared them to solutions predicted by a NN model trained using classical methods (i.e.,

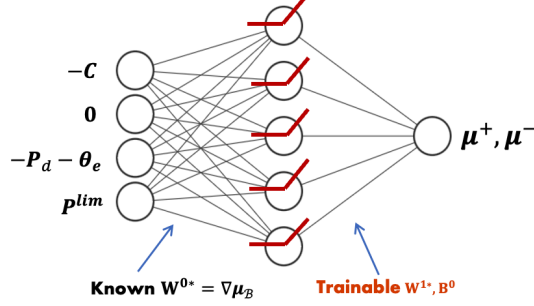


Figure 4.5: Submodel predicting μ^*

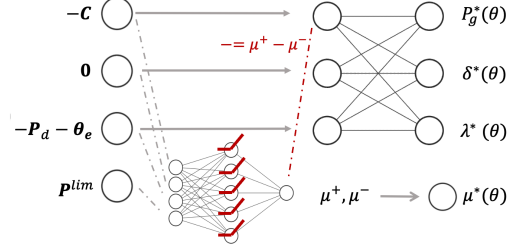


Figure 4.6: SSNN Architecture

with paired input-output data representing particular values of system parameters and corresponding optimal solutions), and also to solutions generated by the Gurobi solver (v10.0.3). The SSNN models were trained using only 1000 data points, and validated and tested on datasets of 1000 and 4000 observations, respectively. To generate training data, demand of all buses was set to 0.01, then one bus was chosen and its load incrementally increased from 0 to the maximum allowed by the system, and the procedure was repeated for one other bus. For the classically trained NN model, we generated training, validation and test sets of 5000, 1000 and 4000 observations, respectively, by setting the demand to the default load values in the Pandapower package and multiplying with a constant sampled from $\text{Uniform}(0.6, 1.4)$. Two types of test datasets were generated using both approaches, local demand range around the default load in the test case, and extreme ranges of demand. Local demand data was generated by multiplying a base demand by a constant sampled from $\text{Uniform}(0.6, 1.4)$. Extreme demand data was generated by fixing all load bus demands to 0.01, replacing one bus with a number sampled from $\text{Uniform}(0, \max)$ and repeating this step for all buses. We did not take explicit measures to avoid data leakage since the distribution of SSNN model training set and distribution of test sets largely

KKT 1 ($\mathbf{P}_g$)	KKT 1 ($\boldsymbol{\delta}$)	KKT 2	KKT 2 ($\leq$)	KKT 3	KKT 4
$\left(\frac{\partial L}{\partial \mathbf{P}_g}\right)^2$	$\left(\frac{\partial L}{\partial \boldsymbol{\delta}}\right)^2$	$\left(\frac{\partial L}{\partial \boldsymbol{\lambda}}\right)^2$	$\left[\max\left(0, \frac{\partial L}{\partial \boldsymbol{\mu}}\right)\right]^2$	$\max(0, -\boldsymbol{\mu})$	$\left(\boldsymbol{\mu}^* \frac{\partial L}{\partial \boldsymbol{\mu}}\right)^2$

Table 4.1: KKT Conditions and corresponding formulae

differ. All the experiments and training were done on a Mac Mini M2 (2023) on Python 3.9.18, using PyTorch 2.0.1.

The SSNN, classically-trained DNN, and Gurobi solver solutions were compared based on violations of the KKT optimality conditions, namely (i) Stationarity (KKT 1 $\mathbf{P}_g, \boldsymbol{\delta}$), (ii) Primal feasibility (KKT 2, KKT 2 ($\leq$)), (iii) Dual feasibility (KKT 3), and (iv) Complementary slackness (KKT 4), which are given in Table 4.1. We calculated squared distance between the true optimal conditions and the model predictions as KKT 1, 2 and 4 are 0, and KKT 3 is non-positive at optimality.

While SSNN was tested on these datasets, the training set is constructed using the critical regions $[\mathbf{CR}_i^-, \mathbf{CR}_i^+]$ detected via the discovery step. To discover sets of binding constraints, all parameters (i.e., active power demand on all buses) were set to 0.01 and an arbitrary θ'_i was incrementally increased from 0 to the maximum number the system allows. The discovery was carried out on one dimension only, so this approach would not detect if other sets of binding constraints could be found by repeating this step with other θ_i . This search pattern is sufficient for the problem studied in this section, however it should be replaced by an exhaustive search algorithm, which is discussed in detail in the next chapter.

To construct the training set for SSNN, different values of θ'_i were sampled from each $[\mathbf{CR}_i^-, \mathbf{CR}_i^+]$ and solved by expanding the coefficient matrix, $\mathbf{J}_{\mathcal{B}_i}$ with the corresponding

		KKT1- $\mathbf{P}^g$	KKT1- δ	KKT2	KKT2 ($\leq$)	KKT3	KKT4
SSNN (ours)	6bus	4.79E-13	1.46E-10	1.44E-13	1.49E-14	3.47E-14	2.92E-14
	30bus	1.31E-13	1.17E-10	8.41E-14	0.00E+00	1.46E-11	1.11E-11
	57bus	6.68E-11	8.35E-08	2.54E-12	1.90E-10	1.37E-08	6.25E-08
DNN	6bus	2.72E-06	7.04E-06	2.93E-05	9.70E-08	1.36E-08	1.01E-07
	30bus	4.08E-07	1.86E-06	5.84E-05	0.00E+00	2.17E-09	6.00E-09
	57bus	2.44E-04	5.90E-05	5.21E-03	1.19E-08	3.27E-08	2.58E-06
Gurobi	6bus	7.92E-14	3.00E-11	6.28E-16	0.00E+00	0.00E+00	2.37E-16
	30bus	6.80E-15	5.27E-12	1.23E-14	0.00E+00	0.00E+00	1.94E-19
	57bus	4.01E-13	8.63E-10	5.96E-15	0.00E+00	0.00E+00	5.13E-19

Table 4.2: Mean Squared KKT Errors on test set generated for a local demand range

binding constraints, running the calculations in (4.4) and checking if the solutions satisfy the KKT conditions. This step was repeated using one other θ_j and fixing all other parameters, assuming the same $[\mathbf{CR}_i^-, \mathbf{CR}_i^+]$ would apply to this range. If the calculated solution does not satisfy KKT, another number was sampled from the same interval. We acknowledge that this method of data generation may not generalize to every problem. This is one of the limitation of this study that is discussed in detail in Section 4.4.

Table 4.2 reports average KKT violations for the three models on the test sets with local load characteristics. Note that this dataset has the same distribution characteristics as the training set for classical DNN training but is out-of-distribution for SSNN. This is because the train set of DNN is obtained simply by train/validation/test split applied to 10,000 data points generated by multiplying a base load with a random factor. On the other hand, train set of SSNN is generated by sampling random input parameters from each critical region. Nevertheless, SSNN predictions satisfied all KKT conditions with less than 1E-10 squared error for 6- and 30-bus systems, with relatively stable performance

		KKT1- $\mathbf{P}^g$	KKT1- δ	KKT2	KKT2 ($\leq$)	KKT3	KKT4
SSNN (ours)	6bus	6.03E-13	1.61E-10	1.52E-13	1.40E-14	2.78E-14	2.64E-14
	30bus	1.64E-13	1.73E-10	2.04E-13	2.69E-12	2.38E-10	1.20E-10
	57bus	1.98E-11	4.07E-08	3.07E-12	2.76E-08	1.90E-09	2.62E-08
DNN	6bus	7.15E-03	8.38E-01	3.26E+00	1.66E-02	5.89E-02	1.48E-03
	30bus	1.66E-02	1.54E-02	3.82E-02	4.76E-05	8.06E-07	2.32E-07
	57bus	7.41E+01	8.50E+01	3.28E+01	9.75E-04	4.03E-02	1.24E+00
Gurobi	6bus	7.92E-14	3.00E-11	6.28E-16	0.00E+00	0.00E+00	2.37E-16
	30bus	6.80E-15	5.27E-12	1.23E-14	0.00E+00	0.00E+00	1.94E-19
	57bus	7.88E-13	1.36E-09	1.76E-14	0.00E+00	0.00E+00	1.30E-15

Table 4.3: Mean Squared KKT Errors on test sets generated for the extreme demand characteristics

across the three test systems and error less than 8E-08 for 57-bus system. On the other hand, KKT violations of the classically trained DNN models are much higher compared to the previous case with best overall performance is achieved for 30-bus system. For example for 57-bus system, KKT-1 violations are measured as 7.14E+01 and 8.50E+01, which are around 1E+05 and 1E+06 times higher than the previous case with local demand range. Similar performance reductions can be seen for the KKT 2 and KKT 4 results. Finally, Gurobi outperforms all models for all measures. This is expected as Gurobi solutions are obtained iteratively until an optimality gap is reduced below a given threshold.

Table 4.3 reports average KKT violations on test sets with extreme load characteristics, reflecting out-of-training distribution examples for classical DNN and mostly out-of-training distribution for SSNN¹. Performance of the SSNN model on the extreme dataset

¹Recall that this dataset was sampled by varying one bus load while fixing others to 0.01 and repeating for all buses. Therefore, it is mostly out-of-training distribution for SSNN, which was trained with a dataset that allows for variations in two load buses only

was similar to its performance on the realistic demand dataset, with 6- and 30-bus errors less than 1E-10 for all KKT measures. Performance reduced somewhat for the 57-bus system, where KKT1- δ , KKT2 ($\leq$), and KKT4 errors increased to 4.07E-08, 2.76E-08, and 2.62E-08, respectively. For the extreme dataset, KKT violations for the DNN model were well above SSNN. The best performance was recorded for the 30-bus system, which produced KKT4 violations of 2.32 E-07, at least 10^3 times larger squared error than SSNN.

4.3.2 Uncertainty

A comparison of the prediction speed of our SSNN model against Gurobi showed that our model has significant computational advantages over traditional solvers. To calculate solutions for 5000 observations, SSNN was 63 times faster than the Gurobi solver for the 6 bus system, which increased to 3543 times faster for the 57 bus system. In nominal terms, the model takes 0.01 to 0.05 seconds to calculate all solutions for all test systems, while solution times for Gurobi increased substantially with system size.

To compare the speed of our model against Gurobi, we generated random 1000 input parameters and solved the problem for all test systems. Figure 4.7 presents the results. As seen, for the 6 bus system, it takes 20 seconds to solve the problem with Gurobi, which increases to 22, 33 and 78 seconds for 14-, 30- and 57-bus systems, respectively. On the other hand, the solution time with SSNN ranges between 0.024 to 0.050 for any test system. In other words, while the solution time with Gurobi increases exponentially as the system size increases, it remains constant with a NN approach.

The low computational cost and strong KKT performance of the SSNN model implies

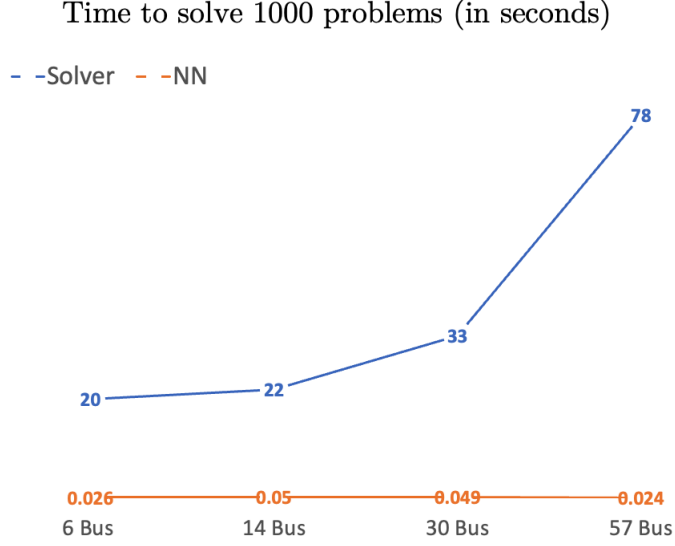


Figure 4.7: Calculation Speed Comparison of SSNN and Gurobi (Solver) over 1000 test problems

that it can be used to quickly generate large distributions of optimal and feasible solutions for the simulation and long range planning of energy systems with uncertain renewable resources, such as wind generation. To explore this, we modified eq. 4.5b of the DC-OPF problem as follows to allow variation:

$$\mathbf{B}\boldsymbol{\delta} - \mathbf{P}_g - \mathbf{P}_{ren} + \mathbf{P}_d + \boldsymbol{\theta}_e = 0, \quad (4.9)$$

where $\mathbf{P}_{ren} \sim \text{Exponential}(\lambda = 1.25)$. In the simulations, the problem was solved repeatedly for 500 random cases of $\mathbf{P}_{ren}$ for each hour. A 1.5 unit generator limit was enforced by truncating the generated values above 1.5.

For this analysis, default load values from the Pandapower package were taken as base demand. To reflect seasonality of hourly changing demand, we used historical demand

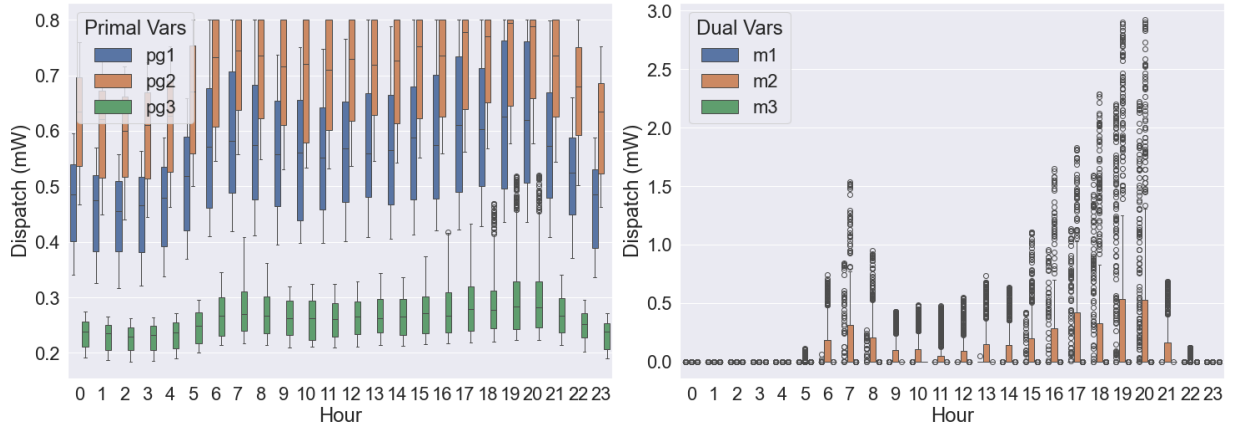


Figure 4.8: Hourly distribution of optimal primal and dual prices

data for Ontario, Canada² as a scaler. Specifically, an arbitrary day (May 11, 2024) was selected and the daily demand was divided by the maximum value throughout the day to act as scalars. Each bus value was multiplied by the constant for every hour to obtain 24 demand inputs. To calculate solutions for 1200 different inputs, our SSNN model ran for 0.03 seconds.

Figure 4.8 presents the distribution of the resulting 500 optimal generator dispatch solutions for 24 hours using boxplots. All three generators have a capacity limit of 0.8 mW. As shown, the boxplots for the lowest cost Generator 2 are higher than other generators. During the day, Generator 2 often runs at full capacity and the dual variable of the associated capacity constraint ($P_2 - 0.8 \leq 0$) is positive. During peak hours (5pm-9pm), Generator 1 also often reaches capacity with its dual variable increasing above zero, forcing the system to use Generator 3.

²<https://www.ieso.ca/en/power-data/data-directory>

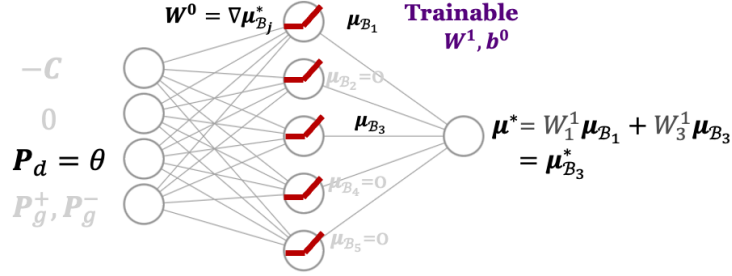


Figure 4.9: Illustration of Model Predictions

4.4 Scalability and Limitations

The SSNN methodology presented in this chapter has limitations with respect to scalability. First, while our models perform better than the alternatives, for the 57 bus system, the training takes a large number of epochs to converge. The model was trained for 300,000 epochs until the training error reduced below 1E-10. This problem is due to finding the optimal trainable model weights and biases that can choose the correct slope from the first layer of the NN. To illustrate this point mathematically, consider the model

$$f_{\mu}(-\mathbf{B} - \boldsymbol{\theta}) = \mathbf{W}^1 \sigma(\mathbf{W}^0 \mathbf{x} + b^0). \quad (4.10)$$

As the slopes of $\mu(\boldsymbol{\theta})$ for all critical regions are contained in the first layer weights, the output of the first layer is all candidate slopes. After the ReLU activation function is applied, the negative predictions are replaced with 0, which yields

$$\mathbf{h}^0 = \sigma(\mathbf{W}^0(-\mathbf{B} - \boldsymbol{\theta})) = \sigma([\boldsymbol{\mu}_{B_1}^+, \boldsymbol{\mu}_{B_2}^+, \dots, \boldsymbol{\mu}_{B_k}^+]^T), \quad (4.11)$$

for k number of critical regions, where $\boldsymbol{\mu}_{\mathcal{B}_j}^+ = \boldsymbol{\mu}_{\mathcal{B}_j}^*$ if the result is nonzero and 0 otherwise. Then, the rest are linearly combined with the second layer weights, as illustrated in Figure 4.9, where the optimal solution, $\boldsymbol{\mu}^* = \boldsymbol{\mu}_{\mathcal{B}_3}^*$, is obtained by linearly combining $\boldsymbol{\mu}_{\mathcal{B}_1}^*$ and $\boldsymbol{\mu}_{\mathcal{B}_3}^*$ when $\boldsymbol{\theta} \in CR_3$. Finding such model parameters that would apply all critical regions is a challenging task that is addressed in Chapter 5 in detail. In the next chapter, we will introduce a NN model that analytically chooses the correct slope in the last layers of the model.

Another limitation of our study is the need to run a solver iteratively to discover the critical regions, by generating inputs by increasing the input parameter with a small increment, e.g., $\theta^i - \theta^{i-1} = 0.01$. Ideally, this process would be repeated for every bus and for different directions, which for larger problems requires solving the problem for a large number of inputs.

Moreover, during the discovery steps, the parameters are fixed to a small number and one parameter, e.g. θ_j , is chosen randomly to be increased from 0 to the largest feasible number. This process discovers lower and upper bounds of critical regions for slopes $\nabla \boldsymbol{\mu}_{\mathcal{B}_i}^*$ such that when $\theta_i \in [CR_j^-, CR_j^+]$, $\nabla \boldsymbol{\mu}_{\mathcal{B}_j}^*$ produces the optimal solutions. To populate data, we assumed all directions are bounded by the same intervals and sampled $\theta_{i'} \in [CR_j^-, CR_j^+]$. This unrealistic assumption sometimes calculated suboptimal results, in which case the problem is solved for another value until an optimal solution is found.

This problem could be addressed as follows. Assuming $\mathbf{W}_{\mathcal{B}_i} \forall i$ are discovered in the discovery step, for every randomly generated $\boldsymbol{\theta}$, there exists one $\mathbf{W}_{\mathcal{B}_i}$ such that $\mathbf{W}_{\mathcal{B}_i}(\mathbf{B}\boldsymbol{\theta})$ satisfies the KKT conditions. Therefore, one could find the optimal solutions for all ran-

domly generated inputs and populate data.

In the next chapter, we will address these limitations with a model that discovers the critical regions without dataset construction.

4.5 Chapter Summary

This chapter introduced an explainable semi-supervised NN modeling approach for multiparametric QP optimization problems with linear constraints that provides precise solutions with generalizability power. The model aligns the piecewise linear nature of NN architecture with the underlying mathematical structure of the optimization problem by deriving the model weights directly from the linear sensitivity function of solutions in each critical region defined by the sets of binding constraints. To train the model, a cost efficient data generation approach was proposed that bypasses using solvers in a loop. As a proof of concept, the SSNN modeling approach was applied to DC-OPF problems with upper and lower bound generator limits. SSNN models trained on only 1000 data points achieved higher KKT optimality and feasibility results than generic DNN models trained classically on five times as much data. The SSNN models generated optimal solutions to large input sets much faster than the Gurobi solver, with little sacrifice in KKT performance. As such, the models can rapidly create a distribution of optimal and feasible solutions to inputs sampled from empirical distributions of uncertain demand and renewable generator dispatches.

This work has focused on the SSNN model architecture, its training, and providing

proof-of-concept support. We acknowledge that further work is needed to develop an efficient discovery algorithm for identifying active constraint combinations that is applicable to a general class of empirical applications. Next chapter addresses these drawbacks with a model that discovers the solution function without the need of solvers or dataset construction.

Chapter 5

A Neural Network Framework for Discovering Closed-Form Solutions to Quadratic Programs with Linear Constraints

The previous chapter showed that it is possible to create a NN model of the solution function to quadratic problems with linear constraints, which can produce optimal solutions with global guarantees. Although the SSNN outperforms traditional NN approaches, there are still practical limitations to training the model to achieve zero error for problems with a large number of variables and inequality constraints. As discussed in Section 4.4, there are a couple of challenges to address. First, although the SSNN derives some of the model

parameters, training is still required for the rest of the weights, which takes a large number of epochs to converge. Second, the discovery of critical regions requires using solvers repeatedly on a large number of input parameters, which increases with the system size.

In this chapter, we address these limitations with a NN model and a learning-via-discovery (LvD) algorithm that calculates all of the model parameters directly from the problem coefficients without training. The proposed approach calculates the model parameters by manipulating the problem coefficients via linear algebraic calculations, rather than training by minimizing the MSE loss function. Therefore, model predictions are independent from dataset sampling and generalize to every feasible input parameter, given that all critical regions of the feasible domain are discovered by the algorithm. In other words, our proposed NN model is a representation of the closed-form solution function to mp-QP with linear constraints given that all critical regions are visited during the discovery step.

The methodology introduced in this chapter builds on the SSNN method introduced in the previous chapter. Similar to SSNN, the first layer weights of the model are fixed to $\mathbf{W}^0 = [\nabla \boldsymbol{\mu}_{\mathcal{B}_1}^*, \dots, \nabla \boldsymbol{\mu}_{\mathcal{B}_k}^*]$, reflecting the slopes of the linear segments of the PWL solution function, each of which corresponds to a critical region, $CR_1, \dots, CR_k$. The second and third layers decide which slope coefficients will be active at any critical region, CR_j , by adding or subtracting the slope difference between adjacent critical regions, CR_{j-1} , i.e. $\nabla \boldsymbol{\mu}_{\mathcal{B}_j}^* - \nabla \boldsymbol{\mu}_{\mathcal{B}_{j-1}}^*$.

The LvD algorithm is used to learn the parameters of the model. The algorithm starts at a feasible input parameter, $\boldsymbol{\theta}^0$, and the corresponding binding constraints at the solution, $\mathcal{B}_0$. Then, the model is initiated only with slope of the linear segment in this critical region,

$\mathbf{W}^0 = [\nabla \boldsymbol{\mu}_{\mathcal{B}_0}^*]$. This model is used to predict optimal solutions until an input parameter, $\boldsymbol{\theta}^{j+1}$, is found such that the current state of the model can no longer calculate optimal and feasible solutions, which indicates that a new critical region has been discovered. Then, the binding constraints in the new critical region, $\mathcal{B}_{j+1}$, are found without the use of solvers, and the NN model is expanded by adding the corresponding $\nabla \boldsymbol{\mu}_{\mathcal{B}_{j+1}}^*$ to $\mathbf{W}^0$. This procedure is repeated until all critical regions are identified and included in the model.

Our approach differs from the literature and contributes as follows:

1. We propose a learning algorithm and a NN model that can solve mp-QP with uncertain RHS values applied on the equality constraints. Our approach derives the weights of the NN model directly from the problem coefficients, independent from data sampling or training. Once a critical region is discovered, optimal model weights are calculated analytically, which guarantees optimality for all input parameters within the critical region. Therefore, our approach to learning is substantially different from the literature with the goal of discovery of critical regions rather than estimating the model weights by minimizing a loss function.
2. The proposed model is entirely explainable where each operation within the NN implements the calculations described in Section 2.2.1. It differs from other NN approaches in the literature (e.g. [56]) where DNNs are trained as black-box models.
3. Unlike some methods that recover optimal solutions if constraints are violated [32], the proposed model guarantees optimality and feasibility without further steps as the closed-form solution formula is represented directly as a NN.

4. Finally, to the best of our knowledge, our model is the first that represents closed-form solutions as a NN to the extent that the discovery step visits all critical regions.

This chapter is organized as follows. Section 5.1 restates the problem and presents a summary of the solution function derived in Section 2.2.1. Section 5.2 describes the proposed model architecture. Section 5.3 introduces the LvD algorithm and explains it using a 2D example. Section 5.4.1 compares the proposed model’s precision and speed against alternative approaches using IEEE test sets and the last section discusses limitations and concludes the chapter.

5.1 Multiparametric QP with Linear Constraints

As discussed in the introduction, the proposed method in this chapter builds on the SSNN model in Chapter 4. In this section, we put a summary of the essential equations for the reader to be able to follow up.

The mp-QP can be written as

$$\text{Minimize } z(\boldsymbol{\theta}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + (\mathbf{C} + \boldsymbol{\theta}_c)^T \mathbf{x} + \mathbf{C}_0, \quad (5.1a)$$

$$\text{s.t. } \mathbf{A}_e \mathbf{x} = \mathbf{b}_e + \boldsymbol{\theta}_e, \quad [\lambda] \quad (5.1b)$$

$$\mathbf{A}_c \mathbf{x} \leq \mathbf{b}_c + \boldsymbol{\theta}_c, \quad [\mu] \quad (5.1c)$$

As detailed in Section 2.2.1, solutions to the above problem can be attained via a com-

posite function that first predicts the optimal solutions to shadow prices of the inequality constraints and then constructs the rest of the solutions. That is,

$$\boldsymbol{\mu}^* = \mu(\boldsymbol{\theta}) \quad (5.2)$$

$$[\mathbf{x}^*, \boldsymbol{\lambda}^*] = g(\boldsymbol{\mu}^*, \boldsymbol{\theta}) \quad (5.3)$$

where

$$g(\boldsymbol{\mu}^*, \boldsymbol{\theta}) = \mathbf{J}^{-1} \begin{bmatrix} -\mathbf{C} - \boldsymbol{\theta}_c + \mathbf{A}_c^T \boldsymbol{\mu}^* \\ -\mathbf{b}_e - \boldsymbol{\theta}_e \end{bmatrix}, \text{ for } \mathbf{J} = \begin{bmatrix} 2\mathbf{Q} & -\mathbf{A}_e^T \\ -\mathbf{A}_e & 0 \end{bmatrix}. \quad (5.4)$$

In the next section, we describe how the above solution function can be represented as a NN model.

5.2 Methodology

5.2.1 The Model

Our model implements the calculations described in Section 5.1 in one NN flow by calculating shadow prices of the inequality constraints and the rest of the solutions through separate parts of the model. As illustrated in Figure 5.1, the model consists of two subnetworks. The first subnetwork, $f_\mu(z)$ is a DNN model with a custom architecture that estimates $\mu(\boldsymbol{\theta})$, whose PWL properties are discussed in Section 2.2.2. The second subnetwork, $f_g(\boldsymbol{\mu}^*, \boldsymbol{\theta})$, is a linear model in the form of $f_g(\mathbf{z}) = \mathbf{W}\mathbf{z} + \mathbf{b}$ that estimates $g(\boldsymbol{\mu}^*, \boldsymbol{\theta})$

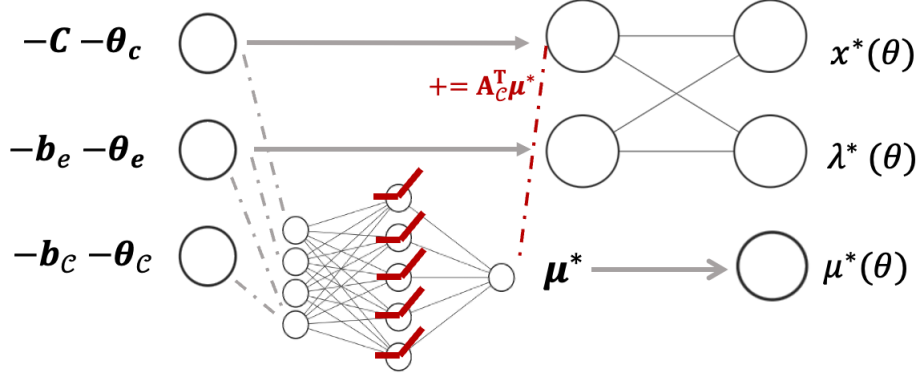


Figure 5.1: The QP Solver Model Architecture

in (5.4).

Notice that the parameters of $f_g(\boldsymbol{\mu}^*, \boldsymbol{\theta})$ are already derived in (4.2), where $\mathbf{W} = \mathbf{J}^{-1}$ and $\mathbf{b} = 0$. However, there is no straightforward method to derive the parameters of $f_\mu(z)$. One can train a surrogate NN model by solving the problem repeatedly and creating datasets. However, as discussed earlier, training this model using traditional approaches is challenging for large dimensional functions. Instead, our approach leverages the discussion in Section 2.2.2 that within each critical region, CR_i , the change in shadow prices with respect to the parameters, $\mu(\boldsymbol{\theta})$ for $\boldsymbol{\theta} \in CR_i$, is a separate linear function whose parameters can be derived from the problem parameters using (4.4). Then, we propose a NN model that chooses the correct $\boldsymbol{\mu}_i^*(\boldsymbol{\theta})$ for any $\boldsymbol{\theta}$. The next section explains the motivation behind this NN model, which is followed by the formulation of $f_\mu(\boldsymbol{\theta})$.

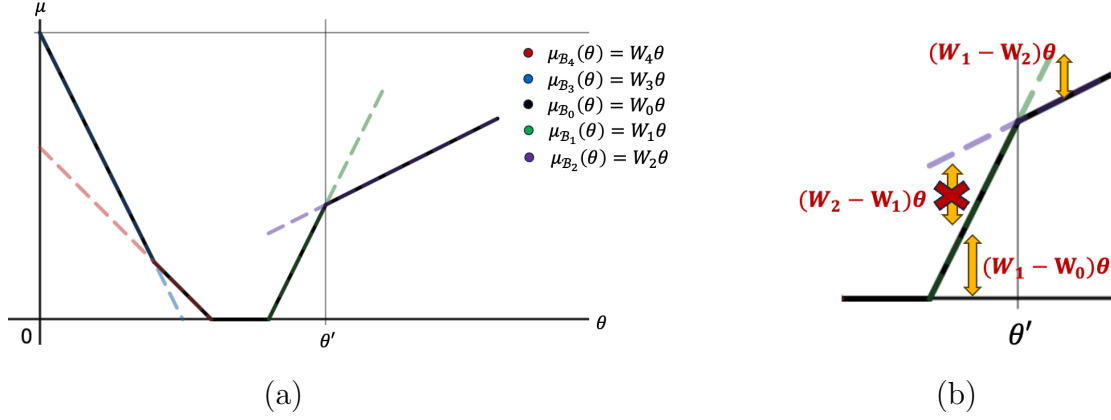


Figure 5.2: Illustration of slope update rules on an example PWL function.

5.2.2 Modeling PWL Changes

The second and third model layers are used to select the correct $\mu_{B_i}^*$ from the array of all slopes, $[\mu_{B_1}^*, \dots, \mu_{B_k}^*]$. To this end, assume the PWL function in Figure 5.2a consists of five linear segments, each of which can be expressed as a linear function, $\mu_{B_i}(\theta) = \mathbf{W}_i\theta$. Each linear facet is adjacent to one or two other facets, and two functions are equal at the intersecting points, θ'' , between adjacent facets, i and j ; i.e., $\mu_{B_i}(\theta') = \mu_{B_j}(\theta')$ and $\mu_{B_i}(\theta'') = \mu_{B_j}(\theta'')$. However, in the case of a multidimensional PWL function, a linear facet can be adjacent to more than two facets and two functions intercept at a multidimensional line.

To control the transition from one slope to another, our model starts with a base slope, i.e., $\mathbf{W}_0$ in the example, and adds the incremental change between two slopes. To illustrate this, consider the example in Figure 5.2b where the slope increases from W_0 to W_1 , and

decreases from W_1 to W_2 . This function can be obtained as

$$f(\theta) = [\mathbf{W}_0\theta]^+ + [(\mathbf{W}_1 - \mathbf{W}_0)\theta]^+ - [(\mathbf{W}_1 - \mathbf{W}_2)\theta]^+ \quad (5.5)$$

where $[z]^+$ is the ReLU activation function, which returns 0 if $z \leq 0$ and z if $z > 0$.

Therefore,

$$f(\theta) = \begin{cases} \mathbf{W}_0\theta & \text{if } \theta < \theta', \\ \mathbf{W}_0\theta + (\mathbf{W}_1 - \mathbf{W}_0)\theta & \text{if } \theta' \leq \theta < \theta'', \\ \mathbf{W}_0\theta + (\mathbf{W}_1 - \mathbf{W}_0)\theta - (\mathbf{W}_1 - \mathbf{W}_2)\theta & \text{if } \theta \geq \theta'', \end{cases} \quad (5.6)$$

$$= \begin{cases} \mathbf{W}_0\theta & \text{if } \theta < \theta', \\ \mathbf{W}_1\theta & \text{if } \theta' \leq \theta < \theta'', \\ \mathbf{W}_2\theta & \text{if } \theta \geq \theta''. \end{cases} \quad (5.7)$$

Notice in the above example, the '+' or '-' signs between terms and in front of each slope difference depend on whether the slope increases or decreases from the previous critical region to the next. The rule can be summarized as follows: When transitioning from CR_i to CR_j , the slope must update from W_i to W_j . If for θ_j sampled from the new critical region, CR_j $\mathbf{W}_j\theta_j > \mathbf{W}_i\theta_j$, then one can conclude that the slope is updated upwards and

downwards otherwise. Therefore, the incremental change in slope is

$$\Delta f(\theta) = \begin{cases} +(\mathbf{W}_j - \mathbf{W}_i)\theta & \text{if } (\mathbf{W}_j - \mathbf{W}_i)\theta_j > 0, \\ -(\mathbf{W}_i - \mathbf{W}_j)\theta & \text{if } (\mathbf{W}_j - \mathbf{W}_i)\theta_j < 0. \end{cases} \quad (5.8)$$

5.2.3 Shadow Price Model: $f_\mu(\mathbf{z})$

The shadow prices of the inequality constraints are predicted via a deep NN with three hidden layers whose calculations are entirely explainable. Briefly, the first layer weights are fixed to the slopes of shadow prices that are obtained by following the calculations in (4.4), and the second and third layer weights are designed to choose the correct slope for a given input. Mathematically, the model can be written as

$$f_\mu(\theta) = \mathbf{W}^{2T} \sigma(\mathbf{W}^{1T} \sigma(\mathbf{W}^{0T}(-\mathbf{B} - \boldsymbol{\theta}))) \quad (5.9)$$

where

$$\mathbf{B} + \boldsymbol{\theta} = \begin{bmatrix} \mathbf{C} \\ \mathbf{b}_e \\ \mathbf{b}_{\mathcal{B}_i} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\theta}_c \\ \boldsymbol{\theta}_e \\ \boldsymbol{\theta}_C \end{bmatrix} \quad (5.10)$$

To explain, we assume there are k critical regions in the global solution function and start by assuming all critical regions and corresponding $\mathcal{B}_i$ are known (We later relax this assumption later with the introduction of the learning algorithm). If $\mathcal{B}_i$ are known, then one can easily calculate $\boldsymbol{\mu}_{\mathcal{B}_i}^*$ following the operations in (4.4). The Shadow Price Model

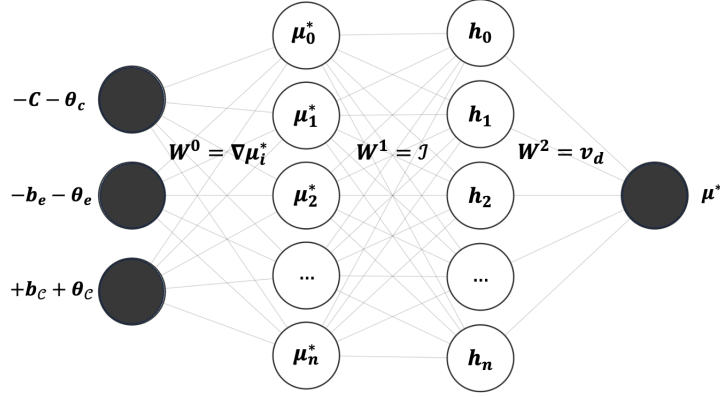


Figure 5.3: $f_\mu(\theta)$ model

defined earlier, $f_\mu(z)$, fixes the first layer weights by stacking all $\mu_{\mathcal{B}_i}^*$ vertically as

$$\mathbf{W}^0 = [\nabla \mu_{\mathcal{B}_1}^*, \nabla \mu_{\mathcal{B}_2}^*, \dots, \nabla \mu_{\mathcal{B}_k}^*]^T \quad (5.11)$$

where $\mu_{\mathcal{B}_i}^* \in \mathbb{R}^{k \times m_2}$.

As the slopes of $\mu(\theta)$ in all critical regions are known, the output of the first layer will produce all candidate optimal shadow prices as

$$\mathbf{W}^0(-\mathbf{B} - \theta) = [\mu_{\mathcal{B}_1}^*, \mu_{\mathcal{B}_2}^*, \dots, \mu_{\mathcal{B}_k}^*]^T. \quad (5.12)$$

Next, we will introduce incidence matrix and direction vector to choose the correct slope at each critical region.

Incidence Matrix and Direction Vector

To choose the correct slope in each critical region and form the PWL shadow price function, we linearly combine the first layer outputs by adding or subtracting one to another. These operations can be handled by two additional NN layers, whose weights are defined as Incidence Matrix, $\mathcal{I} \in \mathbb{R}^{n \times n}$, and Direction Vector, $v_d \in \mathbb{R}^{n \times 1}$. The former is a sparse matrix that forms the increments from the first layer outputs in (5.12).

$$(\mathcal{I}_{ij}, \mathcal{I}_{jj}) = \begin{cases} (-1, 1) & \text{if } (\mathbf{W}_j - \mathbf{W}_i)\theta_j > 0 \text{ for adjacent } (i, j), \\ (1, -1) & \text{if } (\mathbf{W}_j - \mathbf{W}_i)\theta_j < 0 \text{ for adjacent } (i, j), \\ (0, 0) & \text{otherwise.} \end{cases} \quad (5.13)$$

Similarly, the direction vector can be written as

$$v = \begin{cases} 1 & \text{if } (\mathbf{W}_j - \mathbf{W}_i)\theta_j > 0, \\ -1 & \text{if } (\mathbf{W}_j - \mathbf{W}_i)\theta_j < 0. \end{cases} \quad (5.14)$$

In summary, if slope increases when moved from CR_i to CR_j , the size of incidence matrix expands with a new row and column and direction vector expands one dimension. Values of $\mathcal{I}[i, j]$ and $\mathcal{I}[j, j]$ take $(-1, 1)$ and direction vector expands with the value of 1 to form $+(\mathbf{W}_j - \mathbf{W}_i)^+$ function. On the other hand, if the slope reduces, then values of $\mathcal{I}[i, j]$ and $\mathcal{I}[j, j]$ takes $(1, -1)$ and direction vector expands with -1 to form $-(\mathbf{W}_i - \mathbf{W}_j)^+$ function.

Next section, we introduce the learning algorithm that calculates layer weights from the problem coefficients.

5.3 Learning via Discovery Algorithm

The model described in Section 5.2 is constructed directly by using the slopes of the linear segments in each critical region, $\nabla \mu_{\mathcal{B}_i}^*$, the incidence matrix and direction vectors that can be calculated using these slopes. The learning algorithm leverages the flexibility of our model construction that does not require training and employs an iterative process to train the model. The algorithm starts with a set of input parameters, $\mathcal{D}$, that is constructed by incrementally increasing an input parameter towards a direction, and the solution at the initial point, θ_0 . Using this initial solution, a NN model, $f(\theta)$, is initiated with the slope of the solution function, $\nabla \mu_0^*$, in the critical region corresponding to the set of binding constraints, $\mathcal{B}_0$. Initially assuming that there is only one critical region, this model is already capable of predicting solutions to all parameters within this critical region, $\theta \in CR_0$, but solutions calculated by the model for input parameters in $\mathcal{D}$ violate KKT optimality conditions when $\theta \notin CR_0$. If the model violates KKT conditions for any input, we can infer that a new critical region has been discovered given that the input parameters are sampled from a feasible region. In this case, the NN model is expanded by adding the slope of the new critical region to the model and the process is repeated until all critical regions are discovered.

Note that, when a new critical region is found, the set of binding constraints in this region can be found without solving the problem with a solver.

Remark

Recall from Theorem 3 that when $\boldsymbol{\theta}$ is moved from a critical region to an adjacent one, one of the following can occur:

1. A new inequality constraint, k , starts to be violated, i.e., $\mathbf{b}_m + \boldsymbol{\theta}_m - \mathbf{A}_m x(\boldsymbol{\theta}) > 0$. Then the set of binding constraints in the new critical region, $\mathcal{B}_{j+1} = \mathcal{B}_j \cup \{k\}$.
2. One of the constraints that was binding in the previous region is no longer binding, i.e., $\mu_k^*(\boldsymbol{\theta}_{j-1}) > 0$ & $\mu_k^*(\boldsymbol{\theta}_j) = 0$.

Therefore, when it is found that the model cannot satisfy KKT conditions, the new set of binding constraints can be found by expanding the set with the violated constraint or dropping the constraint for which $\nabla \mu_{\mathcal{B}_{i-1}}^*(-\mathbf{B} - \boldsymbol{\theta}_i) < 0$.

For every calculated solution, the algorithm checks whether the solutions satisfy the KKT conditions via the *KKT* function defined as,

$$\begin{aligned} KKT(f(-\mathbf{B} - \boldsymbol{\theta}), \boldsymbol{\theta}) = & \sum \left(\frac{\partial L}{\partial \mathbf{x}}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \right)^2 + \sum \left(\frac{\partial L}{\partial \boldsymbol{\lambda}}(\mathbf{x}) \right)^2 + \sum (\max(0, -\boldsymbol{\mu}))^2 \\ & + \sum \left(\max \left(0, \frac{\partial L}{\partial \boldsymbol{\mu}}(\mathbf{x}) \right) \right)^2 + \sum \left(\boldsymbol{\mu} \frac{\partial L}{\partial \boldsymbol{\mu}_c}(\mathbf{x}) \right)^2 \end{aligned} \quad (5.15)$$

where $\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu} = f(\boldsymbol{\theta})$ and $\partial L / \partial \mathbf{x}$, $\partial L / \partial \boldsymbol{\lambda}$, $\partial L / \partial \boldsymbol{\mu}_c$ are defined in (2.9) and $\sum$ operator is element-wise summation of all outputs. Notice that the function measures violations of all four KKT conditions. Specifically, the first three outputs measure KKT violations of Stationarity, Primal Feasibility of equality constraints and Primal feasibility of inequality

constraints, and the last two outputs are measures for Dual Feasibility and Complementary Slackness conditions, respectively. The function yields 0 for optimal and feasible solutions and is nonzero otherwise. Algorithm 2 details our approach.

Algorithm 2 Learning via Discovery

Input: Search pattern $\mathcal{P} = [\mathcal{D}_1, \dots, \mathcal{D}_n]$, starting point, θ^0 , tolerance tol
Output: Learned model parameters
Solve the problem at the initial point θ^0 and find $\mathcal{B}_0$.
Initiate model $f(-\mathbf{B} - \theta)$ with $\mathbf{W}^0 = [W_{\mathcal{B}_0}^0]$, $\mathbf{W}^1 = [1]$, $\mathbf{W}^2 = [1]$ and $\mathbf{W} = \mathbf{J}^{-1}$
for $\mathcal{D}$ in $\mathcal{P}$ **do**
 while $KKT(f(-\mathbf{B} - \theta^i), \theta^i) \geq tol \quad \forall i \in [1, 2, \dots, d]$ **do**
 Calculate $f(\theta^i) \quad \forall i \in [1, 2, \dots, d]$
 Find smallest i such that $KKT(f(-\mathbf{B} - \theta_i), \theta_i) > tol$
 Find the violated constraint, update $\mathcal{B}_{i+1}$ and W_{i+1}^0
 $\mathbf{W}^0 \leftarrow \mathbf{W}^0 + [W_{i+1}^0]$
 if $(W_{i+1}^0 - W_i^0)\theta^i > 0$ **then**
 $(\mathbf{W}_{ij}^1, \mathbf{W}_{jj}^1) \leftarrow (-1, 1)$
 $\mathbf{W}_j^2 \leftarrow 1$
 else if $(W_{i+1}^0 - W_i^0)\theta^i < 0$ **then**
 $(\mathbf{W}_{ij}^1, \mathbf{W}_{jj}^1) \leftarrow (1, -1)$
 $\mathbf{W}_j^2 \leftarrow -1$
 end if
 end while
end for
 return $\mathbf{W}^0, \mathbf{W}^1, \mathbf{W}^2$

Algorithm 2 takes a search pattern as input, which consists of lists of parameters that are created incrementally from an initial point towards a direction. The outcome of the algorithm highly sensitive to the search pattern, whether it visits all critical regions and the transition from one region to another satisfies the assumptions in Theorem 3. We discuss this point in Chapter 6 in detail.

For the numerical results in Section 5.4.1, we used a simple search pattern, which is

illustrated in Figure 5.7 and detailed in Algorithm 3. The search pattern is constructed by choosing one parameter and increases its value from 0 to the largest feasible number while fixing all parameters to a small number (e.g. 0.001). The search direction is then changed by choosing another parameter to increase and setting the rest of the parameters to a small number.

Algorithm 3 Basic Search Pattern Generator

Input: Initial point θ^0 , increment size α and upper limit θ^+ .
Output: Search pattern in form of list of list of parameters.
Initiate an empty $\mathcal{P}$
for i in $1 : \dim(\theta^0)$ **do**
 Initiate an empty $\mathcal{D}$
 $\theta \leftarrow \theta^0$
 for d in 0 to θ^+ step α **do**
 $\theta_i = d$
 Append θ_i to $\mathcal{D}$
 end for
 Append $\mathcal{D}$ to $\mathcal{P}$
end for
return $\mathcal{P}$

5.3.1 Illustration of the Algorithm on a 2D Example

In this section, we illustrate the functioning of the learning algorithm over a 2D example that can be found in [94, ch. 4]. The problem can be written as below:

$$\text{Min: } \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{C} \mathbf{x} \quad (5.16)$$

$$\text{s.t.:} \quad (5.17)$$

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} 350 \\ 600 \\ -\theta_1 \\ -\theta_2 \end{bmatrix} \quad (5.18)$$

$$\mathbf{x} \geq 0, \quad (5.19)$$

where $\mathbf{Q} = \text{Diag}([156, 162, 162, 126])$ is a diagonal matrix and $\mathbf{C} = [25, 25, 25, 25]$ is a horizontal matrix. The feasible set of parameters is $\boldsymbol{\theta}_f = \{\theta_1, \theta_2 \mid \theta_1 + \theta_2 \leq 1000\}$.

Figure 5.4 presents the critical regions of the feasible parameters for this problem geometrically and the corresponding connectivity graph. As seen, there are four critical regions, CR_1, CR_2, CR_3 , and CR_4 , defined by the corresponding sets of binding constraints, $\mathcal{B}_1 = \{3, 4\}$, $\mathcal{B}_2 = \{1, 3, 4\}$, $\mathcal{B}_3 = \{1, 3, 4, 5\}$, and $\mathcal{B}_4 = \{1, 3, 4, 6\}$, respectively. The critical regions 1 and 2, 2 and 3, 2 and 4 are adjacent to each other.

Note that the proposed algorithm does not require the information about the critical regions or the connectivity between them prior to the training. Algorithm 2 starts at an arbitrary feasible point and discovers this graph iteratively as a byproduct of constructing

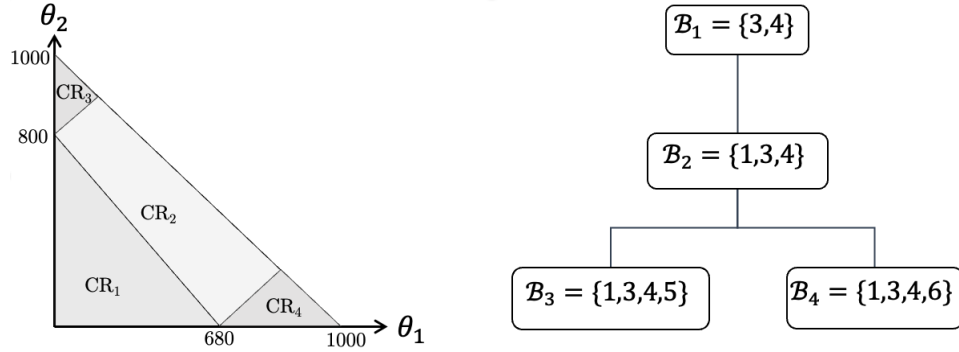


Figure 5.4: Critical regions and connectivity graph for an mp-QP with two parameters (Adapted from [94, p. 59])

the NN model. As illustrated in 5.5, the algorithm starts at an arbitrary θ^0 (iteration 0). As this point is in CR_1 , solving the problem at θ^0 yields $\mathcal{B}_1 = \{3,4\}$, which is used to calculate μ^* and $\mathbf{J}^{-1}$ following (4.4). Then, the solver model, $f(-\mathbf{B} - \theta; \mathbf{W}^0, \mathbf{W}^1, \mathbf{W}^2, \mathbf{W})$ is initiated with $\mathbf{W}^0 = [\nabla \mu_{\mathcal{B}_1}]$, $\mathbf{W}^1 = [1]$, $\mathbf{W}^2 = 1$, and $\mathbf{W} = \mathbf{J}^{-1}$.

As seen in Figure 5.5, at iteration 0, this initial model can already predict the optimal solutions $\forall \theta \in CR_1$. This initial model is used to predict solutions for all data points sampled in the chosen search direction. However, as θ is moved outside the CR_1 towards θ^{d_1} , the model starts violating KKT conditions due to the changing set of binding constraints in the new critical region. The changes in the set of binding constraint can be identified using Remark in Section 5.3. Specifically, in the example, the model violates Constraint 1, which can be identified from the positive value returned from $\mathbf{b}_m + \theta_m - \mathbf{A}x(\theta)$. Therefore, in the next iteration, the model is expanded with the slope of the new critical region, $\nabla \mu_{\mathcal{B}_2}$, and the first layer parameters are updated as $\mathbf{W}^0 = [\nabla \mu_{\mathcal{B}_1}, \nabla \mu_{\mathcal{B}_2}]$. To decide the

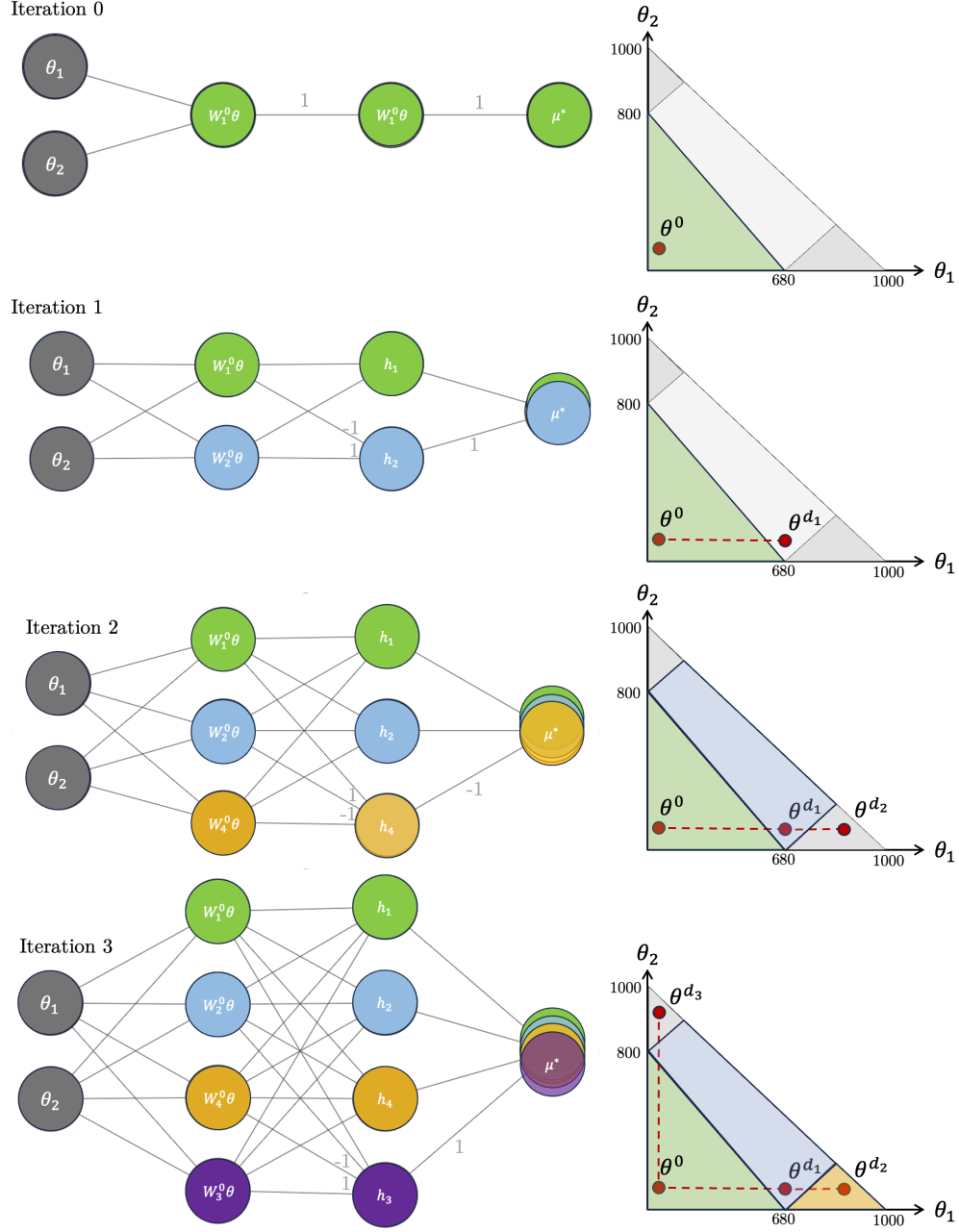


Figure 5.5: Illustration of slope update rules on an example PWL function.

direction of the increment, one can check

$$\nabla \mu_{\mathcal{B}_2}(-\mathbf{B} - \boldsymbol{\theta}^{d_1}) > \nabla \mu_{\mathcal{B}_1}(-\mathbf{B} - \boldsymbol{\theta}^{d_1}) \quad (5.20)$$

As the above statement holds in our example, we expand $\mathbf{W}^1 = \mathcal{I}$ and $\mathbf{W}^2 = v$ with zeros and set $(\mathcal{I}_{12}, \mathcal{I}_{22}) = (-1, 1), v_2 = 1$.

As illustrated in Figure 5.5 - Iteration 1, the updated model can now predict optimal solutions to $\forall \boldsymbol{\theta} \in CR_1 \cup CR_2$ and is used to predict solutions for all data points sampled in the search direction. This procedure is repeated until the KKT conditions are satisfied for all $\boldsymbol{\theta}$ in the search direction.

In the next step, another direction is chosen and the process is repeated. When $\boldsymbol{\theta}$ is increased from $\boldsymbol{\theta}^0$ to $\boldsymbol{\theta}^{d_3}$, the model predicts optimal solutions to all points except $\boldsymbol{\theta} \in CR_3$, where the new slope is added and the model now predicts the optimal solution for all points. After this step, the final model becomes,

$$\begin{aligned} f_\mu(-\mathbf{B} - \boldsymbol{\theta}) &= \mathbf{W}^2 \sigma(\mathbf{W}^1 \sigma(\mathbf{W}^0(-\mathbf{B} - \boldsymbol{\theta}))) \\ &= [\mathbf{W}_1^0 \mathbf{B}_\theta]^+ + [(\mathbf{W}_2^0 - \mathbf{W}_1^0) \mathbf{B}_\theta]^+ + [(\mathbf{W}_3^0 - \mathbf{W}_2^0) \mathbf{B}_\theta]^+ - [(\mathbf{W}_2^0 - \mathbf{W}_4^0) \mathbf{B}_\theta]^+ \end{aligned}$$

where $\mathbf{B}_\theta = -\mathbf{B} - \boldsymbol{\theta}$. We acknowledge that this algorithm is not an exhaustive search and may not discover all critical regions. See Limitations for further discussion.

5.4 Numerical Results

To obtain proof of concept support for our proposed modeling approach, we applied the approach to the DC optimal power flow problem. The problem is defined as a quadratic program with linear constraints where the objective is to find the generator dispatch that minimizes the cost of generating energy while satisfying the power flow constraints and generator dispatch limits, which is defined as follows:

$$\min_{\mathbf{P}_g, \boldsymbol{\delta}} \mathbf{P}^T \mathbf{Q} \mathbf{P} + \mathbf{C}^T \mathbf{P} + \mathbf{C}_0, \quad (5.21a)$$

$$\text{s.t.: } \mathbf{P}_d + \boldsymbol{\theta}_e - \mathbf{P} - \mathbf{B} \boldsymbol{\delta} = 0, \quad (5.21b)$$

$$\mathbf{P}^- \leq \mathbf{P} \leq \mathbf{P}^+, \quad (5.21c)$$

where $\mathbf{Q}, \mathbf{C}, \mathbf{C}_0$ cost coefficients and $\mathbf{B}$ is the susceptance matrix, all of which are constant system parameters. $\mathbf{P} \in \mathbb{R}^N$ is the vector of generator dispatches with $P_i = 0 \ \forall i \notin \mathcal{N}_g$, $\boldsymbol{\delta}_i \in \mathbb{R}^N$ is voltage angles at every bus with $\delta_i = 0$ for the slack bus, and $\mathbf{P}^d \in \mathbb{R}^N$ is the demand at every bus.

We used the proposed closed-form NN approach to discover the solution function to DC-OPF problems for IEEE-6, IEEE-30, IEEE-57 bus test systems, and compared the results against solutions predicted by a DNN model trained using classical methods (i.e., with paired input-output data representing particular values of system parameters and corresponding optimal solutions), and also against solutions generated by the Gurobi solver (v10.0.3). For all test systems, Algorithm 2 was used to derive closed-form NN model. The generic DNN model was trained and validated on datasets of 5000 and 1000 obser-

vations, respectively, and all models (CF, DNN and Gurobi) were compared on 4000 test observations.

We followed two alternative procedures to generate datasets, local perturbations and extreme characteristics, to compare inside and outside training distribution set performances of models. The first dataset is generated by multiplying base load values in the Pandapower package with a random uniform number, i.e. $P_{di} = P_{di}^{base} \times Uniform(0.6, 1.4) \forall i \in \mathcal{N}$. For the dataset with extreme characteristics, one bus demand is increased from 0 to the sum of all generator upper limits while the rest of the loads was set to 0.01, and then this procedure is applied repeatedly for every bus. Solutions for input parameters generated by following both procedures were solved using the Gurobi solver and any infeasible values were labeled. All experiments and training were conducted on a Mac Mini M1 (2020) on Python 3.9.16, using PyTorch 2.0.0.post3. The performance of proposed CF approach, classically-trained DNN, and Gurobi solver solutions were compared based on the KKT violations in 4.1.

5.4.1 Test Results

Table 5.1 reports the average KKT violations by the three models on the dataset prepared by applying perturbations to the base load. For Closed-form NN model, we produce two results; CF and CF64. The former is a model constructed using single precision (represented by 32-bits of memory) allowing for 7 decimal points to represent each number, which is the default setting in PyTorch. The latter uses a double-precision (represented by 64-bits of memory), which allows for 15-17 decimal points.

The results show that CF produces competitive results to Gurobi and calculates solutions with very small error margins. For 6- and 30-bus, KKT violations are less than $4.45\text{E-}10$ for all measures. For 57-bus system, the errors increase around 2 digits compared to 30-bus system but worst violation is calculated as $8.41\text{E-}08$. On the other hand, CF model outperforms DNN model for all cases and all measures, and worst case predictions calculated for CF approach performs better than DNN for almost all cases. These results are remarkable for CF approach considering the test dataset has the same characteristics as the training set of DNN model and is out of training distribution for CF by default.

On the other hand, when precision is increased to double with 64-bit representation, the model accuracy increases dramatically. For all test cases, CF64 outperforms all other models including Gurobi, by a large margin. For example, KKT violations calculated for 57-bus system ranges around $6\text{E-}10$ to $5\text{E-}19$ for Gurobi whereas the worst-case performances calculated for CF64 ranges between $1\text{E-}22$ to $3\text{E-}27$.

Table 5.2 reports the average squared KKT violations for all three modeling approaches as well as the worst case violations seen in the dataset by our approach on the dataset prepared for extreme characteristics. The results for CF approach does not change overall from the previous case and outperform DNN in all cases. However, DNN models performance reduces significantly, generating $1\text{E+}5$ to $1\text{E+}6$ times more violation for all measures overall. For example, $\text{KKT1-}\mathbf{P}_g$ violations calculated for 6-bus reduces from $2.72\text{E-}06$ to $7.15\text{E-}03$ for 6-bus system and from $2.44\text{E-}04$ to $7.41\text{E+}01$ for 57-bus system. Moreover, DNN model feasibility violations are Moreover, the DNN performance reduces with the increasing system size, whereas the CF model performance is relatively stable.

		KKT1- $\mathbf{P}_g$	KKT1- δ	KKT2	KKT2 ($\leq$)	KKT3	KKT4
CF	6bus	1.12E-12	4.45E-10	1.46E-13	5.64E-11	0.00E+00	5.16E-15
	30bus	1.78E-13	1.13E-10	8.14E-14	0.00E+00	0.00E+00	0.00E+00
	57bus	6.48E-11	8.41E-08	2.93E-12	1.91E-11	0.00E+00	2.11E-08
CF (worst)	6bus	5.09E-12	3.44E-09	2.47E-12	1.20E-03	0.00E+00	5.62E-13
	30bus	2.79E-12	1.03E-08	9.88E-12	0.00E+00	0.00E+00	0.00E+00
	57bus	2.90E-09	1.46E-05	1.83E-10	4.55E-08	0.00E+00	2.42E-05
CF64	6bus	1.69E-30	1.17E-27	5.11E-30	5.78E-36	0.00E+00	3.80E-31
	30bus	5.36E-31	5.11E-28	1.03E-30	0.00E+00	0.00E+00	0.00E+00
	57bus	2.27E-28	3.15E-25	2.26E-29	4.54E-29	0.00E+00	7.63E-26
CF64 (worst)	6bus	1.26E-29	2.05E-26	3.08E-29	1.11E-31	0.00E+00	1.34E-29
	30bus	9.66E-30	4.79E-26	3.18E-29	0.00E+00	0.00E+00	0.00E+00
	57bus	8.53E-27	4.14E-23	2.79E-27	5.98E-26	0.00E+00	1.17E-22
DNN	6bus	2.72E-06	7.04E-06	2.93E-05	9.70E-08	1.36E-08	1.01E-07
	30bus	4.08E-07	1.86E-06	5.84E-05	0.00E+00	2.17E-09	6.00E-09
	57bus	2.44E-04	5.90E-05	5.21E-03	1.19E-08	3.27E-08	2.58E-06
Gurobi	6bus	1.52E-15	9.78E-28	2.64E-32	0.00E+00	0.00E+00	2.37E-16
	30bus	1.82E-16	3.32E-21	6.24E-19	0.00E+00	0.00E+00	1.08E-19
	57bus	4.01E-13	8.63E-10	5.96E-15	0.00E+00	0.00E+00	5.13E-19

Table 5.1: Mean Squared KKT Errors on the test sets with Local Perturbations

		KKT1- $\mathbf{P}_g$	KKT1- δ	KKT2	KKT2 ($\leq$)	KKT3	KKT4
CF	6bus	1.12E-12	4.45E-10	1.46E-13	5.64E-11	0.00E+00	5.16E-15
	30bus	1.77E-13	1.62E-10	1.96E-13	3.08E-16	0.00E+00	1.67E-16
	57bus	1.94E-11	4.30E-08	3.11E-12	8.77E-09	0.00E+00	1.11E-08
CF (worst)	6bus	5.50E-12	3.77E-09	2.80E-12	1.29E-06	0.00E+00	6.96E-13
	30bus	3.64E-12	2.35E-08	2.19E-11	8.88E-14	0.00E+00	7.06E-14
	57bus	1.89E-09	3.16E-06	9.96E-10	4.86E-04	0.00E+00	1.71E-05
CF64	6bus	1.83E-30	1.59E-27	5.96E-30	0.00E+00	0.00E+00	2.43E-31
	30bus	8.81E-31	1.04E-27	1.36E-30	6.83E-32	0.00E+00	4.83E-32
	57bus	2.01E-28	2.38E-25	2.68E-29	5.01E-29	0.00E+00	3.32E-26
CF64 (worst)	6bus	1.26E-29	1.19E-26	5.05E-29	1.36E-30	0.00E+00	2.27E-29
	30bus	1.26E-29	6.06E-26	1.28E-28	4.55E-30	0.00E+00	6.65E-30
	57bus	5.57E-27	3.00E-23	5.81E-27	1.69E-26	0.00E+00	3.92E-23
DNN	6bus	7.15E-03	8.38E-01	3.26E+00	1.66E-02	5.89E-02	1.48E-03
	30bus	1.66E-02	1.54E-02	3.82E-02	4.76E-05	8.06E-07	2.32E-07
	57bus	7.41E+01	8.50E+01	3.28E+01	9.75E-04	4.03E-02	1.24E+00
Gurobi	6bus	9.13E-15	1.03E-27	1.83E-31	0.00E+00	0.00E+00	3.62E-16
	30bus	4.17E-15	3.36E-21	6.06E-17	0.00E+00	0.00E+00	9.67E-17
	57bus	5.78E-15	1.17E-14	2.25E-17	0.00E+00	0.00E+00	1.78E-14

Table 5.2: Mean Squared KKT Errors on the test set with Extreme Characteristics

Local	Case	Min	25%	50%	75%	99%	Max
CF64	6bus	8.28E-13	5.46E-09	1.02E-08	3.09E-08	1.66E-07	1.97E-03
	30bus	-1.43E-06	-7.92E-09	1.60E-10	1.20E-08	6.05E-07	1.38E-06
	57bus	-3.34E-05	-1.51E-08	1.05E-07	7.31E-07	5.26E-06	7.65E-05
CF	6bus	-5.77E-06	2.97E-07	2.13E-06	3.92E-06	7.86E-06	1.98E-03
	30bus	-8.88E-07	-3.91E-08	1.69E-07	3.59E-07	8.81E-07	1.19E-06
	57bus	-5.30E-04	-4.70E-05	-1.11E-05	2.42E-05	1.67E-04	5.66E-04
SSNN	6bus	-7.24E-06	-1.14E-06	7.42E-07	2.79E-06	6.77E-06	9.38E-06
	30bus	-1.15E-06	-3.02E-07	-9.72E-08	1.02E-07	5.98E-07	1.01E-06
	57bus	-1.15E-06	-3.02E-07	-9.72E-08	1.02E-07	5.98E-07	1.01E-06
Extreme							
CF64	6bus	-5.68E-14	-1.42E-14	7.32E-10	8.74E-09	1.33E-07	1.58E-06
	30bus	-1.43E-06	-7.92E-09	1.60E-10	1.20E-08	6.05E-07	1.38E-06
	57bus	-1.25E-05	-5.67E-09	1.01E-08	1.69E-07	4.56E-04	3.08E+01
CF	6bus	-5.93E-06	1.38E-06	2.62E-06	3.98E-06	7.70E-06	9.78E-06
	30bus	-9.79E-03	-8.08E-03	-7.12E-03	-6.01E-03	-3.24E-03	-2.50E-03
	57bus	-3.10E-04	-3.05E-05	-1.15E-05	6.78E-06	5.62E-04	3.08E+01
SSNN	6bus	-8.12E-06	-7.27E-07	1.15E-06	2.81E-06	7.01E-06	1.02E-05
	30bus	2.56E-03	6.02E-03	7.12E-03	8.08E-03	9.67E-03	9.80E-03
	57bus	-1.50E-03	-5.06E-05	-2.12E-05	2.74E-07	3.73E-03	3.08E+01

Table 5.3: $C(\mathbf{P}^g) - \hat{C}(\mathbf{P}^g)$ calculated for datasets constructed via local perturbations and reflecting extreme characteristics

Table 5.3 reports the optimality gap between Gurobi solutions and our approaches. Here, we assume Gurobi as the reference point and calculated $C(\mathbf{P}^g) - \hat{C}(\mathbf{P}^g)$. For the test data, we report minimum, median and maximum optimality gaps along with 25th, 75th and 99th percentiles. The results show that the cost difference between our approaches and Gurobi is the minimal for vast majority of the cases with median values around $\pm 1E - 05$. For the dataset consists of local perturbations, it can be seen that the median values calculated for CF64 are positive with 1.02E-08, 1.60E-10, 1.02E-07, indicating that our solutions produce less cost than Gurobi solutions. For other two models, median gaps increase with the problem size. Considering the KKT violations calculated in previous section, it can be said that CF64 model produced less cost than Gurobi, while CF and SSNN models closely follow Gurobi performance.

Results on dataset reflecting extreme characteristics present similar performance to the case with local perturbations. The only exception is for 57-bus system where the maximum gap is calculated as 3.08E+01 for all models. This indicates that for a few examples, Gurobi solutions largely underperform compared to our models. Although these anomalies require further analysis, it does not change overall results that the optimality gap calculated for our approaches are minimal.

Finally, in Figure 5.6 we present a comparison of squared KKT2 equality violations between CF64 model and Gurobi. Here, the violations indicate whether the results satisfy the power balance equations at each bus. It can be seen that both models produce small violations where CF64 performs more consistent among the buses and outperforms Gurobi for most of the power buses. CF64 violations are measured between 1E-29 and 1E-30 whereas Gurobi solutions show larger differences between 1E-17 and 1E-30.

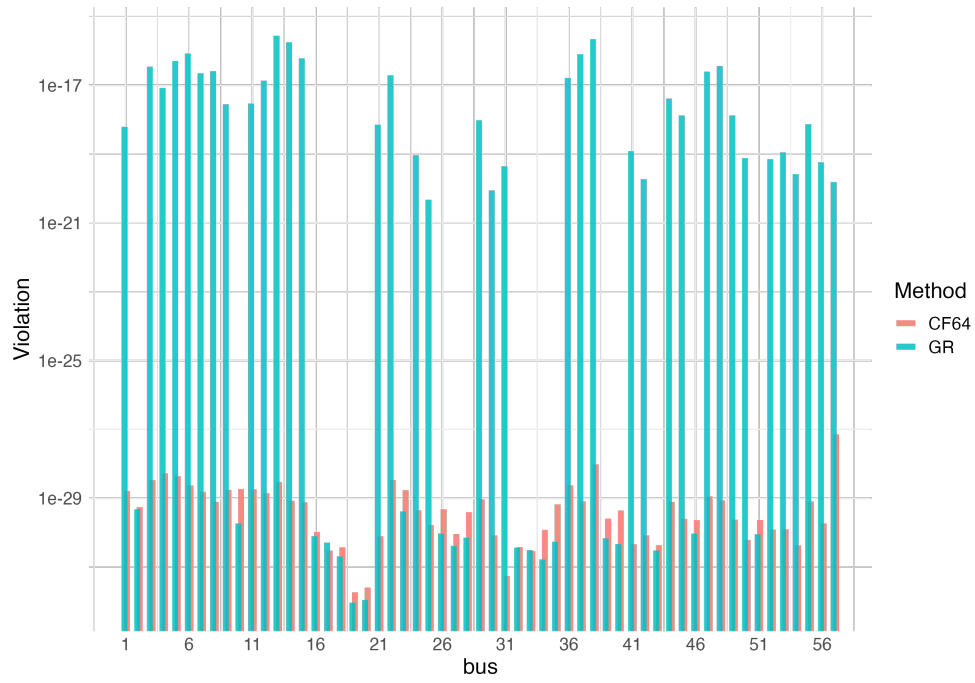


Figure 5.6: Comparison of power injection violations of two methods in 57-bus system

5.4.2 Speed Comparison

In this section we compare the computational speed of the CF-NN model to traditional solver-based optimization methods. Specifically, we use an analysis similar to one in Section 4.3.2, to plan energy dispatches on the 57-bus system over 1-day and 1-year planning horizons, given uncertainty in demand and supply due to renewable resources. To generate input parameters, we used default load values from the Pandapower package as base demand; historical hourly demand data for Ontario were then scaled down within the range $\{0, 1\}$. Each bus value was multiplied by the constant of the hour to obtain hourly demand inputs. As in Section 4.3.2, we selected May 11, 2024 for 1-day planning and the year 2024 (January 1st to December 31st) for 1-year planning.

Additionally, we sampled 500 random renewable generator dispatch values per hour, $\mathbf{P}_{ren}$, from an exponential distribution with $\lambda = 1.25$. A 1.5 unit generator limit was enforced by truncating the generated values above 1.5. The generator dispatch was injected to the test system by modifying the DC-OPF power balance equation (5.21b) with a nondispatchable renewable generator as follows:

$$\mathbf{B}\boldsymbol{\delta} - \mathbf{P}_g - \mathbf{P}_{ren} + \mathbf{P}_d = 0. \quad (5.22)$$

The results are shown in Table 5.4. Our CF model was able to predict the optimal solutions with very small errors, for the case of one-day horizon (12000 input parameters) in 16 milliseconds, and for one-year horizon (4392000 input parameters) in 6.23 seconds. Comparable solver-based computation times for Gurobi are estimated to be 15.6 minutes

	1 day	1 year
KKT 1 ($\mathbf{P}_g$)	1.08E-10	5.98E-11
KKT 1 (δ)	1.44E-07	8.06E-08
KKT 2 ($=$)	3.83E-12	2.63E-12
KKT 2 ($\leq$)	4.66E-11	1.82E-12
KKT 3	0	0
KKT 4	8.87E-08	2.28E-09
Time (NN-Closed-form)	0.016 s	6.23 s
Time (Gurobi)	15.6 min (est)	95.16 hr (est)
# Observations	12000	4392000

Table 5.4: Performance Comparison of Closed-form and Gurobi on 57-bus system to calculate solutions to long-term planning problem

for daily data and 95.16 hours for annual data, respectively. Note that these estimates were based on the time needed for Gurobi to solve 1000 57-bus problems, as reported in Figure 4.7.

5.4.3 Line Limits

The results in Section 5.4.1 and 5.4.2 show that our CF model and the proposed learning-via-discovery algorithm can discover the solution function for the problem in (5.21). However, the problem in (5.21) ignores line limits, which can affect the topology of (adjacency between) critical regions.

The problem with line limits can be written as

$$\min \mathbf{P}^T \mathbf{Q} \mathbf{P} + \mathbf{C}^T \mathbf{P} + \mathbf{C}_0, \quad (5.23a)$$

$$\text{s.t.: } \mathbf{P}_d + \boldsymbol{\theta}_e - \mathbf{P} - \mathbf{B} \boldsymbol{\delta} = 0, \quad (5.23b)$$

$$\mathbf{P}^- \leq \mathbf{P} \leq \mathbf{P}^+, \quad (5.23c)$$

$$-\mathbf{F}^+ \leq \mathcal{H} \mathbf{B} \boldsymbol{\delta} \leq \mathbf{F}^+, \quad (5.23d)$$

where (5.23d) are the line limits.

When the line limits are ignored, the connectivity graph between adjacent critical regions is a simple graph that can be discovered with a simple search algorithm. However, when the line limits are included, we need a better search algorithm. Figure 5.7a illustrates critical regions similar to the test cases in Section 5.4.1, where each region with a different shade is a distinct critical region. It can be seen that, when the discovery algorithm starts at $\boldsymbol{\theta}^0$ and $\boldsymbol{\theta}$ is increased along the θ_1 axis until the last feasible point, the algorithm visits all critical regions. When moving from $\boldsymbol{\theta}^0$ along the θ_2 -axis, the same critical regions are discovered.

When line limits are added as constraints, increasing each parameter, θ_i , can trigger a different set of binding constraints, as illustrated in Figure 5.7b. By increasing $\boldsymbol{\theta}$ from $\boldsymbol{\theta}^0$ along the θ_1 axis, the discovery algorithm would discover CR_1, CR_2, CR_3 , and CR_6 , respectively, whereas increasing $\boldsymbol{\theta}$ along θ_2 axis would lead to the discovery of CR_1, CR_2 , and CR_5 respectively. Notice that in both of these cases, the search algorithm would miss CR_4 . Moreover, changing the starting point from $\boldsymbol{\theta}^0$ to another point such as $\boldsymbol{\theta}'$ and

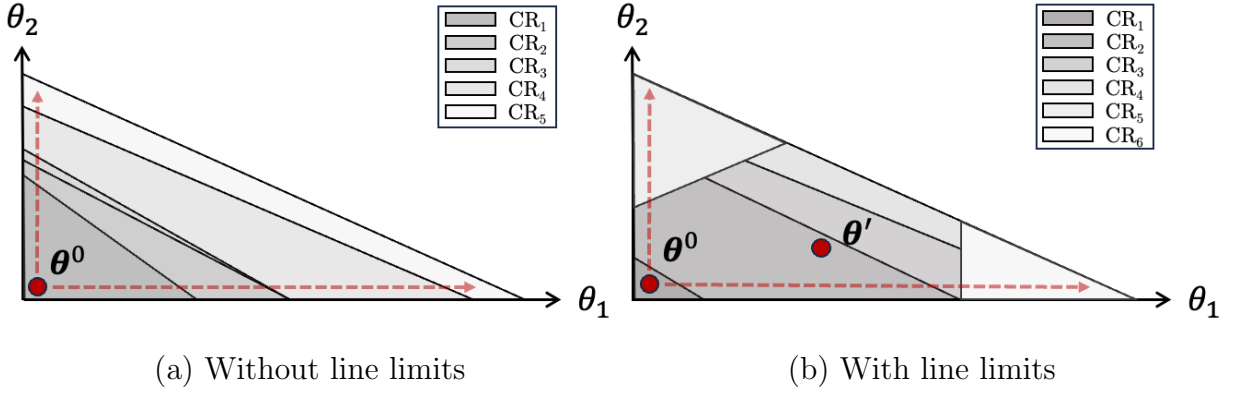


Figure 5.7: Connectivity of critical regions for problems without and with line limits

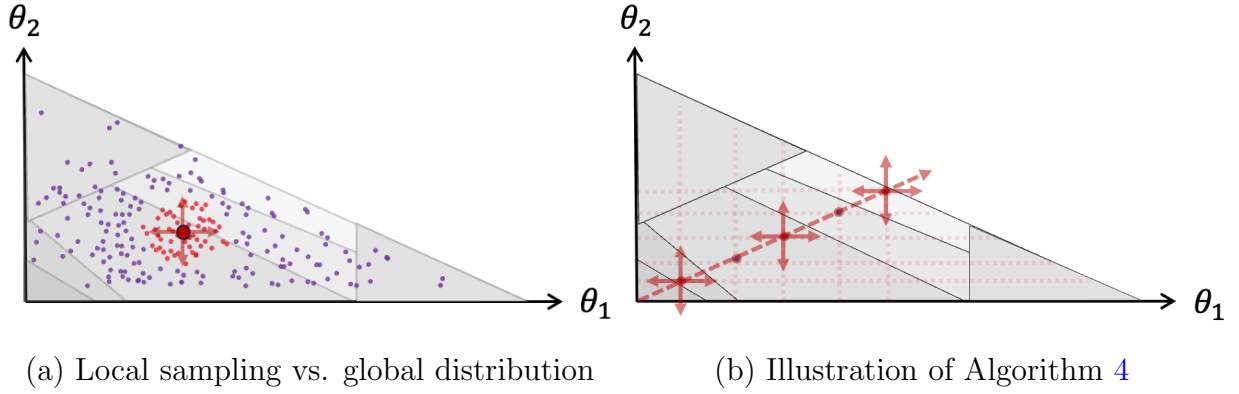


Figure 5.8: Illustration of dataset distribution and the search pattern in Algorithm 4

moving along both axes does not guarantee discovering all critical regions.

In our preliminary experiments, we sampled a dataset by locally perturbing a base parameter, P_{base}^d with a multiplier sampled from a distribution, which is the method we used to generate the dataset in earlier sections (i.e. for Table 4.2). When the discovery algorithm was run by setting $\theta_e = \mathbf{P}_{base}^d$ and moving each θ_i from 0 to the maximum feasible number, our model could predict the optimal solutions for every test point. However, when $\mathbf{P}_{base}$ was changed, our model failed to calculate optimal solutions due to new critical

regions that were not discovered by the algorithm.

Figure 5.8a illustrates the distribution of input parameters obtained by applying local perturbations and the global distribution of the parameters, shown as red dots and purple dots, respectively. As seen in the figure, sampling data by applying local perturbations to a base load generates datapoints from two critical regions only. Therefore, this approach does not result in a model that can guarantee solutions outside these two regions.

For this section, we modified our search pattern to generate input parameters moving along axes starting from a base load that is changed iteratively. Algorithm 4 and Figure 5.8b detail this approach. As shown in the figure, the search algorithm starts with a small value of k that is used to increase or decrease the base load, and parameters are generated along each axis starting from the base point until the end of feasible region. The learning-via-discovery algorithm is run to discover critical regions, and to build the model for these regions. Then, k is incrementally increased and the search and discovery steps are repeated.

To test our model, we also update the test data generation algorithm in a similar fashion to the search algorithm, which is given in Algorithm 5. The algorithm starts with a base parameter set, θ , and the set of scaling factors, $[k_1, \dots, k_n]$. With each value of k , θ_{base} is updated and a new parameter set is generated by multiplying the updated base parameter set with a random multiplier, r_j , sampled from a Uniform distribution.

Algorithm 4 Search Pattern for DC-OPF with Line Limits

```
1: Input: List of scaling factors,  $[k_1, k_2, \dots, k_n]$ , base parameter,  $\theta^{base}$ , step size,  $\alpha$ 
2: Output: Dataset  $\mathcal{D}$ 
3: procedure GENERATE PARAMETERS
4:   Initialize empty dataset  $\mathcal{D}$ 
5:   Initialize  $maxCap = \text{sum}(\mathbf{P}^+)$ 
6:   for  $k$  in  $[k_1, k_2, \dots, k_n]$  do
7:     for  $load$  in  $[0, 0.01, 0.02, \dots, maxCap]$  do
8:       Initiate  $\theta = \theta_{base} \cdot k$ 
9:       for  $j$  in  $1 : \text{dim}(\theta)$  do
10:         $\theta_j = load$ 
11:        Append  $\theta$  to  $\mathcal{D}$ ,
12:      end for
13:    end for
14:  end for
15:  return  $\mathcal{D}$ 
16: end procedure
```

5.4.4 Test Results with Line Limits

Our proposed algorithm was run on the 6-bus system to solve the problem in 5.23 as proof of concept support for our modified search pattern and LvD algorithm. The line limits for each line were set to 200 MW and the base load was set to 140 MW for all load buses. The set of scaling factors was set to $\mathcal{K} = [1, 1.125, \dots, 2]$. To generate the test set, we used Algorithm 5 to sample 10,000 input parameters for each value of k and solved each input case using the Gurobi solver. After removing infeasible inputs, the resulting number of data points are presented in Table 5.5 under the # column.

Table 5.5 presents the results. As seen in the table, for all values of $k \leq 1.625$, the model satisfies KKT optimality conditions with violations less than 5E-10 MSE for all measures. For $k = [1.75, 1.875, 2]$, violation errors begin to increase for KKT2 ($\leq$) and

Algorithm 5 Test Dataset Generation for DC-OPF with Line Limits

```
1: Input: List of scaling factors,  $\mathcal{K} = [k_1, k_2, \dots, k_n]$ , base parameter,  $\theta^{base}$ , sample size,  $N$ 
2: Output: Dataset  $\mathcal{D}$ 
3: procedure GENERATE PARAMETERS
4:   Initialize empty dataset  $\mathcal{D}$ 
5:   for  $k$  in  $\mathcal{K}$  do
6:     Initiate  $\theta' = \theta^{base} \cdot k$ 
7:     for  $i$  in  $1 : N$  do
8:       Initiate  $\theta = [0, 0, \dots, 0]$ 
9:       for  $j$  in  $1 : \dim(\theta)$  do
10:        Generate  $r_j \sim Unif(0.6, 1.4)$ 
11:         $\theta_j = \theta'_j \cdot r_j$ 
12:        Append  $\theta$  to  $\mathcal{D}$ ,
13:      end for
14:    end for
15:  end for
16:  return  $\mathcal{D}$ 
17: end procedure
```

k	#	KKT1 ($\mathbf{P}^g$)	KKT1 (δ)	KKT2	KKT2 ($\leq$)	KKT3	KKT4
1.0	9924	5.40E-13	1.63E-10	1.98E-13	4.91E-15	0	6.04E-15
1.125	10000	5.78E-13	1.71E-10	1.82E-13	4.24E-15	0	3.10E-15
1.25	10000	6.51E-13	1.86E-10	1.51E-13	2.85E-15	0	1.27E-15
1.375	10000	7.12E-13	2.01E-10	1.23E-13	1.70E-15	0	4.84E-16
1.5	9998	7.56E-13	2.16E-10	1.02E-13	7.80E-16	0	2.17E-16
1.625	9755	8.06E-13	2.29E-10	9.46E-14	3.92E-16	0	6.59E-16
1.75	8797	8.16E-13	2.33E-10	9.37E-14	9.77E-11	0	1.24E-10
1.875	7196	8.28E-13	2.40E-10	9.61E-14	1.63E-08	0	5.46E-09
2.0	5372	8.31E-13	2.42E-10	1.03E-13	2.85E-07	0	1.06E-07
1.75*	8792	8.16E-13	2.33E-10	9.36E-14	1.81E-16	0	1.67E-15
1.875*	7170	8.28E-13	2.40E-10	9.53E-14	6.81E-14	0	4.13E-13
2.0*	5331	8.31E-13	2.41E-10	1.02E-13	5.06E-14	0	1.35E-12

Table 5.5: Mean Squared KKT Errors on the test set with Local Perturbations

KKT4. An analysis of the cases where our model under-performs indicated that these input parameters were located in undiscovered critical regions close to the boundaries of the feasible region. When these input parameters are removed from the test set, the KKT violations reduced to levels similar to those observed for $k \leq 1.625$.

Figure 5.9 presents the connectivity graph summarizing the critical regions discovered by our algorithm. Each node in the graph represents a critical region where certain constraints are binding. For example, the top node represents a critical region where constraints #3 and #4 are binding. Each path from the top node ($\mathcal{B} = [3, 4]$) to the bottom (i.e., $[0, 14]$, $[1, 8, 13]$, $[8, 13]$ $[4, 10]$, $[0, 10]$, $[12, 14]$) is discovered by the search algorithm as it traverses critical regions in different directions.

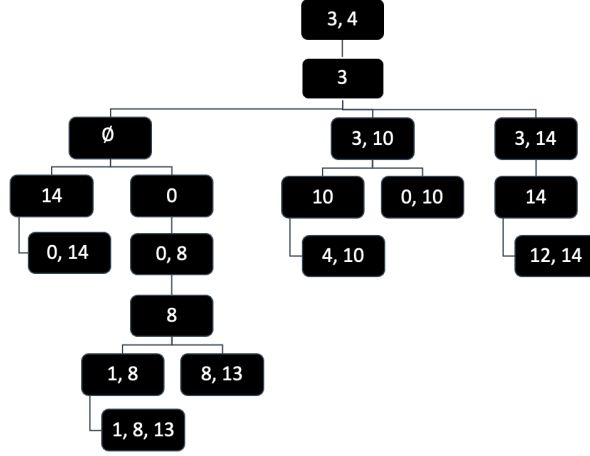


Figure 5.9: Connectivity graph between critical regions

5.5 Chapter Summary

This chapter proposed a NN based approach to discover the closed-form solution to QP with linear constraints. The methodology builds up on shortcomings of the SSNN model in the previous chapter, namely difficulty in reducing the training error even with large numbers of epochs and the requirement of using a solver in a loop to discover the critical regions. Unlike SSNN that requires training, the closed-form approach derives all the model parameters from the problem coefficients based on the adjacency of critical regions. More clearly, the model forms the PWL solution function using the slopes of each linear segments and adding breaks where the two adjacent segment intersects.

To discover the adjacency of the critical regions, we proposed an algorithm that starts with a NN consisting of one linear segment only and generate data along one direction. While this model is able to calculate optimal solutions for parameters within the initial critical region, the predictions violate the KKT conditions in other critical regions, where

new set of binding constraints and the slope of the new linear segment are derived without referring a solver. The model is updated with the slope of this new region and is able to calculate solutions in all discovered critical regions. The procedure is repeated until all critical regions are visited.

The proposed model has significant implications. First, the model imitates the QP solution procedure in an entirely explainable way; by handling different solution steps via two subsequent networks that calculates the shadow prices of inequality constraints and constructing the rest of the solutions. Moreover, the model parameters are obtained directly from the problem parameters via algebraic operations and not by training on a dataset. Hence, our approach generalizes seamlessly to outside the training dataset as long as all critical regions are discovered. In other words, because linear segments in the solution function are derived from the problem, the model guarantees optimality independent of how data is sampled.

There are some limitations to our approach, mainly about the search pattern employed by the discovery algorithm, its assumptions that does not generalize to all cases and floating point precision limit of the package, which are discussed in the Conclusion chapter in detail.

Chapter 6

Conclusion

This thesis investigated how NNs solve nonlinear optimization problems, examined the sources of prediction error, and proposed a NN modeling approach that discovers the closed-form solution to QP with linear constraints. We were motivated by the observation that surrogate NN models perform well at predicting solutions to certain types of optimization problems. Conventionally, mathematical optimization has been viewed as problems that are solved iteratively by computer programs until an acceptable error level is achieved. However, the performance of surrogate NN models indicates that optimization could also be studied as a process of finding continuous solution functions that map feasible input parameters to optimal solutions.

In Chapter 3, we studied the AC-PF problem, whose solutions with respect to feasible input parameters form a continuous nonlinear function. We first estimated an error bound between the true solutions and a hypothetical NN model that is just complex enough to

connect all training data with linear interpolation, and examined the relation between the estimated error bound and the dataset size. The chapter shows that a sufficiently large NN can produce accurate predictions when trained with very large datasets, but even so its performance is limited by the quality of the training dataset, which is traditionally obtained by the solution of optimization problems for any given input. To overcome the challenges, we proposed a solver-free data generation approach for training DNNs to solve the AC-PF problem. The method bypasses the need for solvers to construct datasets by directly utilizing the PF equations. By applying the proposed approach to IEEE test systems we showed that it outperforms solver-based NNs and different optimization algorithms by a wide margin.

Instead of approximating solution functions of continuous optimization problems, previous research in multiparametric programming hypothesized that NNs could be used to represent the solution function of QP with linear constraints exactly, thanks to its piecewise affine functional form. In Chapter 4, we illustrated that it would be practically impossible to obtain a NN model to represent this function using standard training approaches and highlighted strategies to train this model. Later, we proposed a semi-supervised NN architecture that derives some of the optimal model weights, which are typically learned from the dataset, directly from the problem coefficients. Using IEEE DC-OPF test sets, we showed that SSNN can easily produce optimal and feasible solutions to thousands of random input parameters within a second.

Although the SSNN model can predict solutions with minimal error, construction of the training set still requires using a solver in a loop to discover the critical regions within the feasible input range. In Chapter 5, we addressed this issue by modifying the NN archi-

ture and proposing a learning procedure to derive all of the model parameters without training. The proposed algorithm starts with one solution within a critical region and initiates the NN model with the parameters of the linear segment of the solution function within this critical region. This basic NN model is already able to satisfy KKT optimality conditions for all points in this critical region, but as parameters are incrementally adjusted, eventually solutions are predicted that violate the optimality or feasibility conditions, indicating that a new critical region has been discovered. Based on which KKT conditions are violated, the learning via discovery algorithm infers the set of binding constraints in the newfound critical region and expands the NN model with the parameters calculated for this region. The procedure is repeated until all critical regions are discovered. On DC-OPF problem with box constraints, our model can produce solutions with less than $1e-30$ squared violations, outperforming major commercial optimizers. It is shown that when line limits are considered, our model can maintain its performance if a more comprehensive search pattern is employed.

6.1 Discussion and Future Work

There are limitations to our approach in this current form to be generalized to larger optimization problems. First, the success of the learning algorithm is highly sensitive to the search pattern, i.e. whether all the critical regions are visited in the discovery step. While the basic search pattern illustrated in Figure 5.7 is enough for the problem without the line limits, we had to change the pattern when the limits are included. Nevertheless, these examples also show that this issue is not a fundamental problem for the proposed

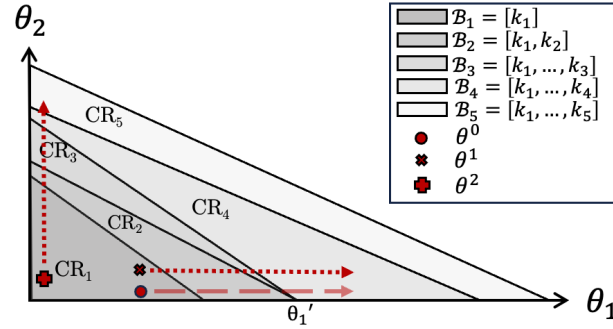


Figure 6.1: The issue arises in case of degeneracy

NN model and learning algorithm since after switching the search pattern, the model successfully produced optimal and feasible solutions for the DC-OPF problem with line limits.

Second, the results for the problem with line limits in Section 5.4.3 shows that our proposed LvD algorithm is also sensitive to the assumptions of the Theorem 3. Specifically, it is assumed that the parameters are moved from region to another, one constraint is added or deduced from the list of binding constraints. This assumption does not hold in case of degeneracy at the solution, i.e., when the number of binding constraints at the solution are more than the dimensionality of the space. One such case is illustrated in Figure 6.1. When the search pattern starts from θ^0 and moves along θ_1 -axis, the model discovers CR_1, CR_2 and CR_4 and skips CR_3 entirely. This is a violation of the assumption because when moved from CR_2 to CR_4 , constraints k_3, k_4 are added to the set. In this case, the model would fail to produce optimal solutions for regions CR_3, CR_4 and CR_5 since important parameters are not included in the model.

One solution can be adding small perturbations to the θ^0 , such as moving the point above (to θ^1 in the figure). However, depending on the size of the increments, the algorithm

may skip CR_3 in this case as well. Therefore, the issue could be solved by reducing the increment size or changing the direction of search, e.g., from θ^2 along second axis.

The closed-form NN model and the learning algorithm proposed in Chapter 5 have remarkable features that offer significant advantages. Usually, training models require large training sets, tedious validation procedures and computationally expensive training steps to guarantee good performance on test sets, and yet often fail to generalize outside the training set. The closed-form discovery approach takes less than a second to learn the optimal model using matrix operations applied on problem coefficients. Moreover, it derives all model parameters without training and results in near-zero test set error.

The low computation cost feature of the proposed closed-form NN approach offers an alternative use of that would not be possible with solver-based training approaches in the literature. Specifically, for each direction on which the model is expected to discover new regions, one can initiate a new model and add the discovered regions to the connectivity graph. Once all regions are discovered, another model can be initiated from the graph. Therefore, separating the discovery and learning steps can be a promising direction to be addressed in future work.

The proposed approaches in Chapter 4 and 5 were applied to problems with uncertain parameters added to right hand side of equality constraints and ignored the changes in inequality constraints, θ_C . In our preliminary experiments that are not reported in this thesis, we noted that our model performance is unaffected unless θ_C is within a certain interval. However, larger values of θ_C changes the topology of the critical regions and hence the connectivity graph that the model is constructed with is no longer valid. Moreover, if

such changes are applied to constraints that appear in central nodes in the connectivity graph, then it can sequentially affect a large part of the graph. In future studies, we will address the effect of parameters on the inequality constraints more detailed.

This research has notable practical implications on accuracy and the speed of the model calculations. In the experiments, we showed that our models satisfy the KKT optimality and feasibility conditions and maintain better performances as system size increases compared to DNN models trained on datasets. Also, significant computational speedup in the proposed model predictions is demonstrated in the methodology chapters. Applying the proposed closed-form NN model to a 57-bus power system with 500 data points per hour everyday for 1-year interval yields fast computation times of approximately 6 seconds compared to 95 hours with commercial solvers like Gurobi. Additional performance benchmarks in Chapter 3 demonstrate that solution times remain virtually unchanged as problem sizes are increased.

Moreover, compared to the other models in the literature, the proposed models in Chapter 4 and 5 have strong generalizability power. Our analysis in Section 4.1.1 indicated that there is a strict relationship between model complexity and minimal dataset size that must be satisfied in order for the trained model to represent the solution function. Therefore, we strictly controlled NN model architectures and the proposed learning procedures to derive properties of the solution function from the coefficients of the underlying problem. To train the semi-supervised model, the dataset was constructed by sampling data points from each critical region separately, to satisfy the requirements of minimal dataset size.

There are promising directions to generalize our approach to more common problems.

The proposed NN approach constructs solution functions to continuous problems, yet it is promising for solving mixed integer problems such as MIQP. As discussed in [94, p. 114], solution function to MIQP problems are discontinuous and the critical regions can also be divided by nonlinear segments. In future work, we will examine these scenarios and modify the proposed NN approach to model these functions.

One difficult case for NN based approaches, including ones in the literature, is the time-coupling constraints such as battery state of charge and ramp-up constraints. Including these constraints dramatically increases the number of decision variables. For example, having a battery in the system requires the problem to be solved for 24 hours separately, with each hour's decision variables depend on the previous hour's solution. One solution to this problem is using time decomposition techniques with Bender's cuts where the NN runs as a subproblem solver. As the solutions to all subproblems can be predicted in 50 milliseconds by the NN model, it would gain enormous time advantages. A similar approach can be applied to solve MIQP problem, which could be more practical than developing special NN models to solve mixed integer problems.

This study addresses QP with linear constraints, which is commonly thought to be the general case of LP. For our methodology, however, solving LP is quite different than QP as the problem cannot be decomposed naturally into linear and PWL parts. Specifically, our approach is based on predicting shadow prices and subtracting from the RHS and obtaining the rest of the solutions via a linear mapping. This solution approach is not applicable to LP because the solutions cannot be isolated from the inequality constraints. More clearly, LP does not have a solution when the inequality constraints are not binding. Moreover, the shadow prices of the inequality constraints are stepwise functions, which

cannot be naturally represented by NNs with ReLU activation functions whose functional form is always continuous. However, CF-NN model in Chapter 5 that adds the increments between slopes of different segments, i.e. $\mathbf{W}_{j-1} - \mathbf{W}_j$, can be a promising direction towards NN based mp-LP solvers. Future study will address this point in more detail.

Finally, our methods proposed in this thesis showed that optimization problems can be viewed as finding continuous functions that are defined from a domain to a range, i.e. from input parameters to optimal solutions. In Chapter 3, it was shown that while this function cannot be written analytically, it can be approximated with a NN model. For the QP problems in Chapter 5, we successfully derived this unobserved solution function without model training by aligning the NN model with the properties of the function. This was achieved thanks to the ReLU activation functions that give the NN model the PWL shape.

While it is out of scope of this study, aligning NN models with other nonlinear solution functions might be possible by affine transformation using customized activation functions. For example, the simplified target function 3.3 in Chapter 3 can be represented by a NN using custom cos and sin functions applied in the first layer. Similarly, a truncated Fourier Series, $g(x) = \frac{1}{2}a_0 + \sum_{n=1}^{k-1}(a_n \cos nx + b_n \sin nx)$, can be represented by a NN model with an architecture similar to that used in (3.4) - (3.8). We believe exact NN representations of widely used engineering applications has potentially impactful benefits that deserves future work.

Our thesis is a step towards analytically formulating solution functions of optimization problems with the aid of NN models. This thesis demonstrates that solution functions

of optimization problems can indeed be formulated analytically for QP problems with linear constraints. Our findings are demonstrated on problems with box constraints or more complex optimization problems involving a smaller number of variables. Significant further research is required to address the limitations and update the learning algorithm to be used in a general class of problems. Considering decades of research in optimization theory that developed solutions to challenges gradually, we believe that a solid theoretical foundation for applying NNs to solve a wide range of optimization problems will likely be developed in the near future.

References

- [1] Joaquin Acevedo and Efstratios N Pistikopoulos. An algorithm for multiparametric mixed-integer linear programming problems. *Operations research letters*, 24(3):139–148, 1999.
- [2] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pages 136–145. PMLR, 2017.
- [3] Kyri Baker. A learning-boosted quasi-newton method for ac optimal power flow. *arXiv preprint arXiv:2007.06074*, 2020.
- [4] Kyri Baker. Solutions of dc opf are never ac feasible. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, pages 264–268, 2021.
- [5] Kyri Baker and Andrey Bernstein. Joint chance constraints reduction through learning in active distribution networks. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 922–926. IEEE, 2018.
- [6] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning

- to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018.
- [7] Mato Baotić, Frank J Christophersen, and Manfred Morari. A new algorithm for constrained finite time optimal control of hybrid systems with a linear performance index. In *2003 European Control Conference (ECC)*, pages 3323–3328. IEEE, 2003.
 - [8] Joshua Bassey, Lijun Qian, and Xianfang Li. A survey of complex-valued neural networks. *arXiv preprint arXiv:2101.12249*, 2021.
 - [9] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit solution of model predictive control via multiparametric quadratic programming. In *Proceedings of the 2000 American control conference. ACC (IEEE Cat. No. 00CH36334)*, volume 2, pages 872–876. IEEE, 2000.
 - [10] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.
 - [11] Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable perturbed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.
 - [12] DP Bertsekas. *Nonlinear Programming*, volume 1015. Athena Scientific Belmont, MA, third edition, 2016.
 - [13] David Biagioni, Peter Graf, Xiangyu Zhang, Ahmed S Zamzam, Kyri Baker, and

- Jennifer King. Learning-accelerated admm for distributed dc optimal power flow. *IEEE Control Systems Letters*, 2020.
- [14] Jan Blechschmidt and Oliver G Ernst. Three ways to solve partial differential equations with neural networks—a review. *GAMM-Mitteilungen*, 44(2):e202100006, 2021.
- [15] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. Geometric algorithm for multiparametric linear programming. *Journal of optimization theory and applications*, 118:515–540, 2003.
- [16] C Caetano, JL Reis Jr, J Amorim, M Ruv Lemes, and A Dal Pino Jr. Using neural networks to solve nonlinear differential equations in atomic and molecular physics. *International Journal of Quantum Chemistry*, 111(12):2732–2740, 2011.
- [17] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6), 2021.
- [18] Mary B Cain, Richard P O’neill, Anya Castillo, et al. History of optimal power flow and formulations. *Federal Energy Regulatory Commission*, 1:1–36, 2012.
- [19] Pablo Felipe Oteíza Canales. *Machine Learning Aided Column Generation for Solving Transmission Network Expansion Planning*. PhD thesis, Universidad Técnica Federico Santa María, 2024.
- [20] Jacques Carpentier. Optimal power flows. *International Journal of Electrical Power & Energy Systems*, 1(1):3–15, 1979.

- [21] Minas Chatzos, Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. High-fidelity machine learning approximations of large-scale optimal power flow. *arXiv preprint arXiv:2006.16356*, 2020.
- [22] Wenbo Chen, Mathieu Tanneau, and Pascal Van Hentenryck. End-to-end feasible optimization proxies for large-scale economic dispatch. *IEEE Transactions on Power Systems*, 2023.
- [23] Yize Chen, Ling Zhang, and Baosen Zhang. Learning to solve dcopf: A duality approach. *Electric Power Systems Research*, 213:108595, 2022.
- [24] Yu Christine Chen, Jianhui Wang, Alejandro D Domínguez-García, and Peter W Sauer. Measurement-based estimation of the power flow jacobian matrix. *IEEE Transactions on Smart Grid*, 7:2507–2515, 2015.
- [25] Cheng Chi, Amine Aboussalah, Elias Khalil, Juyoung Wang, and Zoha Sherkat-Masoumi. A deep reinforcement learning framework for column generation. *Advances in Neural Information Processing Systems*, 35:9633–9644, 2022.
- [26] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [27] Tim De Ryck, Ameya D Jagtap, and Siddhartha Mishra. Error estimates for physics informed neural networks approximating the navier-stokes equations. *arXiv preprint arXiv:2203.09346*, 2022.
- [28] Deepjyoti Deka and Sidhant Misra. Learning for dc-opf: Classifying active sets using neural nets. In *2019 IEEE Milan PowerTech*, pages 1–6. IEEE, 2019.

- [29] Fabrizio Detassis, Michele Lombardi, and Michela Milano. Teaching the old dog new tricks: Supervised learning with constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3742–3749, 2021.
- [30] Frederik Diehl. Warm-starting ac optimal power flow with graph neural networks. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 1–6, 2019.
- [31] Priya Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. *Advances in neural information processing systems*, 30, 2017.
- [32] Priya Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021.
- [33] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.
- [34] Christian Feller and Tor Arne Johansen. Explicit mpc of higher-order linear processes via combinatorial multi-parametric quadratic programming. In *2013 European Control Conference (ECC)*, pages 536–541. IEEE, 2013.
- [35] Christian Feller, Tor Arne Johansen, and Sorin Olaru. Combinatorial multi-parametric quadratic programming with saturation matrix based pruning. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 4562–4567. IEEE, 2012.

- [36] Anthony V Fiacco. Sensitivity analysis for nonlinear programming using penalty methods. *Mathematical programming*, 10(1):287–311, 1976.
- [37] Anthony V Fiacco. *Introduction to sensitivity and stability analysis in non linear programming*. New York: Academic Press, 1983.
- [38] Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 630–637, 2020.
- [39] Ferdinando Fioretto, Pascal Van Hentenryck, Terrence WK Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. Lagrangian duality for constrained deep learning. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part V*, pages 118–135. Springer, 2021.
- [40] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.
- [41] Stéphane Fliscounakis, Patrick Panciatici, Florin Capitanescu, and Louis Wehenkel. Contingency ranking with respect to overloads in very large power systems taking into account uncertainty, preventive, and corrective actions. *IEEE Transactions on Power Systems*, 28(4):4909–4917, 2013.
- [42] Tomas Gal and Josef Nedoma. Multiparametric linear programming. *Management Science*, 18(7):406–422, 1972.

- [43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [44] Arun Gupta, Sharad Bhartiya, and Paluri SV Nataraj. A novel approach to multi-parametric quadratic programming. *Automatica*, 47(9):2112–2117, 2011.
- [45] Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33:18087–18097, 2020.
- [46] Boris Hanin and Mark Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.
- [47] Fouad Hasan, Amin Kargarian, and Javad Mohammadi. Hybrid learning aided in-active constraints filtering algorithm to enhance ac opf solution time. *IEEE Transactions on Industry Applications*, 57(2):1325–1334, 2021.
- [48] Xing He, Lei Chu, Robert Qiu, Qian Ai, and Wentao Huang. Data-driven estimation of the power flow jacobian matrix in high dimensional space. *arXiv preprint arXiv:1902.06211*, 2019.
- [49] Akira Hirose. Continuous complex-valued back-propagation learning. *Electronics Letters*, 20(28):1854–1855, 1992.
- [50] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

- [51] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [52] W Huang and M Chen. Deepopf-ngt: A fast unsupervised learning approach for solving ac-opf problems without ground truth. In *ICML 2021 Workshop on Tackling Climate Change with Machine Learning*, 2021.
- [53] Yuchong Huo, François Bouffard, and Géza Joós. Integrating learning and explicit model predictive control for unit commitment in microgrids. *Applied Energy*, 306:118026, 2022.
- [54] Xiaowei Jia, Jared Willard, Anuj Karpatne, Jordan S Read, Jacob A Zwart, Michael Steinbach, and Vipin Kumar. Physics-guided machine learning for scientific discovery: An application in simulating lake temperature profiles. *ACM/IMS Transactions on Data Science*, 2(3):1–26, 2021.
- [55] Zhenya Jia and Marianthi G Ierapetritou. Uncertainty analysis on the righthand side for milp problems. *AIChE journal*, 52(7):2486–2495, 2006.
- [56] Benjamin Karg and Sergio Lucia. Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics*, 50(9):3866–3878, 2020.
- [57] Justin Katz, Iosif Pappas, Styliani Avraamidou, and Efstratios N Pistikopoulos. Integrating deep learning models and multiparametric programming. *Computers & Chemical Engineering*, 136:106801, 2020.

- [58] Dustin Kenefake, Rahul Kakodkar, Sahithi S Akundi, Moustafa Ali, and Efstratios N Pistikopoulos. A multiparametric approach to accelerating relu neural network based model predictive control. *Control Engineering Practice*, 151:106041, 2024.
- [59] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.
- [60] Rajab Khalilpour and IA Karimi. Parametric optimization with uncertainty on the left hand side of linear programs. *Computers & chemical engineering*, 60:31–40, 2014.
- [61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [62] James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. Learning hard optimization problems: A data generation perspective. *Advances in Neural Information Processing Systems*, 34:24981–24992, 2021.
- [63] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. *arXiv preprint arXiv:2103.16378*, 2021.
- [64] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.
- [65] Isaac E Lagaris, Aristidis C Likas, and Dimitris G Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

- [66] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.
- [67] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *Proc. ICLR*, 2018.
- [68] Wenting Li and Deepjyoti Deka. Physics-informed learning for high impedance faults detection. In *2021 IEEE Madrid PowerTech*, pages 1–6. IEEE, 2021.
- [69] Yuxuan Li, Chaoyue Zhao, and Chenang Liu. Model-informed generative adversarial network (MI-GAN) for learning optimal power flow. *arXiv preprint arXiv:2206.01864*, 2022.
- [70] Zukui Li and Marianthi G Ierapetritou. A new methodology for the general multi-parametric mixed-integer linear programming (MILP) problems. *Industrial & engineering chemistry research*, 46(15):5141–5151, 2007.
- [71] Amir Lotfi and Mehrdad Pirnia. Constraint-guided deep neural network for solving optimal power flow. *Electric Power Systems Research*, 211:108353, 2022.
- [72] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017.
- [73] Alan Sussmann Manne. *Notes on parametric linear programming*. Rand Corporation, 1953.

- [74] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.
- [75] Andrew J Meade Jr and Alvaro A Fernandez. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 19(12):1–25, 1994.
- [76] Sidhant Misra, Line Roald, and Yeesian Ng. Learning for constrained optimization: Identifying optimal active constraint sets. *INFORMS Journal on Computing*, 2021.
- [77] George S Misyris, Andreas Venzke, and Spyros Chatzivasileiadis. Physics-informed neural networks for power systems. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2020.
- [78] Georgios S Misyris, Jochen Stiasny, and Spyros Chatzivasileiadis. Capturing power system dynamics by physics-informed neural networks and optimization. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 4418–4423. IEEE, 2021.
- [79] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.
- [80] Rahul Nellikkath and Spyros Chatzivasileiadis. Physics-informed neural networks for minimising worst-case violations in dc optimal power flow, 2021.

- [81] Rahul Nellikkath and Spyros Chatzivasileiadis. Physics-informed neural networks for AC optimal power flow. *Electric Power Systems Research*, 212:108412, 2022.
- [82] Yeesian Ng, Sidhant Misra, Line A Roald, and Scott Backhaus. Statistical learning for dc optimal power flow. In *2018 Power Systems Computation Conference (PSCC)*, pages 1–7. IEEE, 2018.
- [83] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 2006.
- [84] Richard Oberdieck, Nikolaos A Diangelakis, Ioana Nascu, Maria M Papathanasiou, Muxin Sun, Styliani Avraamidou, and Efstratios N Pistikopoulos. On multiparametric programming and its applications in process systems engineering. *Chemical engineering research and design*, 116:61–82, 2016.
- [85] Richard Oberdieck and Efstratios N Pistikopoulos. Explicit hybrid model-predictive control: The exact solution. *Automatica*, 58:152–159, 2015.
- [86] Xiang Pan, Wanjun Huang, Minghua Chen, and Steven H Low. Deepopf-al: Augmented learning for solving ac-opf problems with multiple load-solution mappings. *arXiv preprint arXiv:2206.03365*, 2022.
- [87] Iosif Pappas, Nikolaos A Diangelakis, and Efstratios N Pistikopoulos. The exact solution of multiparametric quadratically constrained quadratic programming problems. *Journal of Global Optimization*, 79(1):59–85, 2021.
- [88] Daniel R Parisi, Maria C Mariani, and Miguel A Laborde. Solving differential equations with unsupervised neural networks. *Chemical Engineering and Processing: Process Intensification*, 42(8-9):715–721, 2003.

- [89] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.
- [90] Byungkwon Park, Yang Chen, Mohammed Olama, Teja Kuruganti, Jin Dong, Xiaofei Wang, and Fangxing Li. Optimal demand response incorporating distribution imp with pv generation uncertainty. *IEEE Transactions on Power Systems*, 37(2):982–995, 2021.
- [91] Junyoung Park and Jinkyoo Park. Physics-induced graph neural network: An application to wind-farm power estimation. *Energy*, 187:115883, 2019.
- [92] Sejun Park, Chulhee Yun, Jaeho Lee, and Jinwoo Shin. Minimum width for universal approximation. *arXiv preprint arXiv:2006.08859*, 2020.
- [93] Maria Laura Piscopo, Michael Spannowsky, and Philip Waite. Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions. *Physical Review D*, 100(1):016002, 2019.
- [94] Efstratios N Pistikopoulos, Nikolaos A Diangelakis, and Richard Oberdieck. *Multi-parametric optimization and control*. John Wiley & Sons, 2020.
- [95] Dimitris C Psychogios and Lyle H Ungar. A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38(10):1499–1511, 1992.
- [96] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems

- involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.
- [97] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Inferring solutions of differential equations using noisy multi-fidelity data. *Journal of Computational Physics*, 335:736–746, 2017.
- [98] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.
- [99] Paul Ramond. The abel–ruffini theorem: Complex but not complicated. *The American Mathematical Monthly*, 129(3):231–245, 2022.
- [100] Francisco Sahli Costabal, Yibo Yang, Paris Perdikaris, Daniel E Hurtado, and Ellen Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8:42, 2020.
- [101] Girish Sharma, Abhishek Agarwala, and Baidurya Bhattacharya. A fast parallel gauss jordan algorithm for matrix inversion using cuda. *Computers & Structures*, 128:31–37, 2013.
- [102] Y. Shen, Y. Sun, X. Li, A. Eberhard, and A. Ernst. Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. In *The Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022.
- [103] Jochen Stiasny, George S Misyris, and Spyros Chatzivasileiadis. Physics-informed

- neural networks for non-linear system identification for power system dynamics. In *2021 IEEE Madrid PowerTech*, pages 1–6. IEEE, 2021.
- [104] Jochen Stiasny, Georgios S Misyris, and Spyros Chatzivasileiadis. Transient stability analysis with physics-informed neural networks. *arXiv preprint arXiv:2106.13638*, 2021.
- [105] M Sugawara. Numerical solution of the schrödinger equation by neural network and genetic algorithm. *Computer Physics Communications*, 140(3):366–380, 2001.
- [106] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pages 9367–9376. PMLR, 2020.
- [107] Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.
- [108] Petter Tøndel, Tor Arne Johansen, and Alberto Bemporad. An algorithm for multi-parametric quadratic programming and explicit mpc solutions. *Automatica*, 39(3):489–497, 2003.
- [109] Pascal Van Hentenryck. Machine learning for optimal power flows. *Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications*, pages 62–82, 2021.
- [110] Enrique Gabriel Vera, Claudio A Cañizares, Mehrdad Pirnia, Tatiana Pontual Guedes, and Joel David Melo Trujillo. Two-stage stochastic optimization model

- for multi-microgrid planning. *IEEE Transactions on Smart Grid*, 14(3):1723–1735, 2022.
- [111] Felix Voigtlaender. The universal approximation theorem for complex-valued neural networks. *arXiv preprint arXiv:2012.03351*, 2020.
 - [112] Y. Wang, X. Zhou, H. Zhou, L. Chen, Z. Zheng, Q. Zeng, S. Cai, and Q. Wang. Transmission network dynamic planning based on a double deep-q network with deep resnet. *IEEE Access*, 9:76921–76937, 2021.
 - [113] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.
 - [114] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, 30, 2017.
 - [115] Martina Wittmann-Hohlbein and Efstratios N Pistikopoulos. On the global solution of multi-parametric mixed integer linear programming problems. *Journal of Global Optimization*, 57:51–73, 2013.
 - [116] D. Wu and A. Lisser. A deep learning approach for solving linear programming problems. *Neurocomputing*, 520:15–24, 2023.
 - [117] Haoyu Yuan, Fangxing Li, Yanli Wei, and Jinxiang Zhu. Novel linearized power flow and linearized opf models for active distribution networks with application in distribution lmp. *IEEE Transactions on Smart Grid*, 9(1):438–448, 2016.

- [118] Ahmed S Zamzam and Kyri Baker. Learning optimal solutions for extremely fast ac optimal power flow. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 1–6. IEEE, 2020.
- [119] Matthew D Zeiler. Adadelata: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.