Machine Learning and Quantum Computing for Optimization Problems in Power Systems

Sarthak Gupta

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

December 12, 2022

Blacksburg, Virginia

Keywords: Optimal power flow, constrained learning, deep learning, smart inverters, chance constraints, quantum computing, variational quantum eigensolver

Machine Learning and Quantum Computing for Optimization Problems in Power Systems

Sarthak Gupta

(ABSTRACT)

While optimization problems are ubiquitous in all domains of engineering, they are of critical importance to power systems engineers. A safe and economical operation of the power systems entails solving many optimization problems such as security-constrained unit commitment, economic dispatch, optimal power flow, and optimal planning. Although traditional optimization solvers and software have been successful so far in solving these problems, there is a growing need to accelerate the solution process. This need arises on account of several aspects of grid modernization, such as distributed energy resources, renewable energy, smart inverters, batteries, etc, that increase the number of decision variables involved. Moreover, the technologies entail faster dynamics and unpredictability, further demanding a solution speedup. Yet another concern is the growing communication overhead that accompanies this large-scale, high-speed, decision-making process. This thesis explores three different directions to address such concerns. The first part of the thesis explores the *learning-to-optimize* paradigm whereby instead of solving the optimization problems, machine learning (ML) models such as deep neural networks (DNNs) are trained to predict the minimizers of these problems. The second part of the thesis also employs deep learning, but in a different manner. DNNs are utilized to model the dynamics of IEEE 1547.8 standard-based local Volt/VAR control rules, and then leverage efficient deep learning libraries to solve the resulting optimization problem. The last part of the thesis dives into the evolving field of quantum computing and develops a general strategy for solving stochastic binary optimization problems using variational quantum eigensolvers (VQE).

Machine Learning and Quantum Computing for Optimization Problems in Power Systems

Sarthak Gupta

(GENERAL AUDIENCE ABSTRACT)

A reliable and economical operation of power systems entails solving large-scale decision-making mathematical problems, termed as *optimization* problems. Modern additions to power systems demand an acceleration of this decision-making process while managing the accompanying communication overheads efficiently. This thesis explores the application of two recent advancements in computer science – machine learning (ML) and quantum computing (QC), to address the above needs. The research presented in this thesis can be divided into three parts. The first part proposes replacing conventional mathematical solvers for optimization problems, with ML models that can predict the solutions to these solvers. Colloquially referred to as *learning-to-optimize*, this paradigm learns from a historical dataset of good solutions and extrapolates them to make new decisions in a fast manner, while requiring potentially limited data. The second part of the thesis also uses ML models, but differently. ML models are used to represent the underlying physical dynamics, and convert an originally challenging optimization problem into a simpler one. The new problem can be solved efficiently using popular ML toolkits. The third and final part of the thesis aims at accelerating the process of finding optimal binary decisions under constraints, using QC.

Dedication

To my parents,

Jyoti and Raman Gupta,

for the infinite love.

Acknowledgments

First and foremost, I would like to acknowledge my PhD advisor and mentor, Dr. Vassilis Kekatos. For accepting me back in his group, for showing confidence in my skills, for teaching me everything I know about being a researcher, for being patient with me when I was slow, and for pushing me when I needed to be pushed. This dissertation is a direct result of directions and motivations, from my discussions with him. I consider myself utterly fortunate that I had him as my PhD advisor, for I know this journey would have been many times more difficult otherwise. I cherish our time together, as it allowed me to fill my little bucket from his ocean of knowledge.

Next, I would like to thank my PhD committee for their valuable questions, inquiries, and suggestions for my PhD research. I am also very grateful to my mentors – Dr. Deepjyoti Deka, and Dr. Sidhant Misra, for my internship at Los Alamos National Laboratory. Working under their supervision, help me expand my skill-set in the direction of applied research for the industry. It also enabled me to publish some of work that forms this dissertation.

I would like to acknowledge the financial support I received through the National Science Foundation (NSF) grants 1751085 and 2034137, and through seed funding from Virginia Commonwealth Cybersecurity Initiative (CCI).

I am extremely thankful to my partner Julie Westinghouse, who provided me with immense emotional support during this PhD journey. I am grateful for her patience, compassion towards me, and love. I recognize how lucky I am to have her in my life and look forward to our future together.

I would be amiss if I did not recognize the friendship and support of my friends Tapas, Manish, Aarushi, Mana, Akshay, Mayank, Prerna, Vamshi, and Hannah. Without them,

the duration of my PhD work would have been devoid of laughter, joy, and fun.
I would like to make a special mention of my dearest friend Ashish Arora, who was always there for me in my times of need. He was one of the kindest and most gentle souls I knew, and tragically left this planet towards the final days of my PhD. I deeply miss him and pray that the next part of his journey is kinder to him.

And lastly, I am most grateful to my Guru, Gurudev Sri Sri Ravi Shankar. To him, I owe everything, outside of his grace, I have nothing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Dissertation Outline

This dissertation is a compilation of six research works published in (submitted to) peer review journals and conferences, and can be partitioned into three parts. Part 1 explores the *learning to optimize* paradigm for two power systems applications – reactive power compensation using smart inverters, and stochastic optimal power flow for transmission grids. It innovates by presenting the *optimize and learn* approach whereby DNNs learn the underlying mapping to optimal decisions without labels, in an unsupervised manner. Moreover, we handle stochastic constraints on average values, and probabilistic chance constraints, using the primal-dual updates. Part 2 focuses on the optimal design of local Volt/VAR control rules, for smart inverters, inspired by the IEEE 1547.8 standard [4]. While also leveraging DNNs, the utilization is different from the *learning to optimize* paradigm. DNNs are trained not to predict the optimal solutions but to model the underlying closed-loop, control dynamics. We show how such modeling can enable solving a challenging mixed integer non-linear program (MINLP) in a scalable manner using DNN libraries. The last part of the work focuses on the more general problem of stochastic binary optimization, which is NP-hard to solve using classical computing algorithms. Focusing on the noisy intermediate-scale quantum (NISQ) era of quantum computing, we develop a constrained VQE solver for stochastic binary optimizations. The solution strategy is useful in power system optimization problems that involve discrete variables such as unit commitment, transmission line switching, optimal transformer tap ratio, and phase shift setting. We next elucidate upon the specific chapters that make up the three parts, before diving into the individual chapters themselves.

1.2 DNNs for Solving Stochastic OPFs

Part 1 of this thesis aims to predict optimal solutions to stochastic optimization problems given by the general formulation

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbb{E}[f(\mathbf{x}, \mathbf{z})] \\ \text{s.to} \quad & \mathbb{E}[\mathbf{g}(\mathbf{x}, \mathbf{z})] \leq \mathbf{0}. \end{aligned} \tag{1.1}$$

where $\mathbf{x}$ is the vector optimization variable, $\mathbf{z}$ is the vector of random variables, and f and $\mathbf{g}$ are scalar and vector valued functions of $\mathbf{x}$ and $\mathbf{z}$, respectively. Under the *learning to optimize* paradigm, conventionally one strives to train a machine learning model $\boldsymbol{\pi}(\mathbf{w}, \mathbf{z})$, such as DNNs, with $\mathbf{w}$ as the parameters, and $\mathbf{z}$ as the input, that would predict the optimal solution to the deterministic version of the optimization problem (1.1). This would require a training dataset of the form $\{(\mathbf{x}_i^*, \mathbf{z}_i)\}_{i=1}^S$, where $\mathbf{x}_i^*$ is the optimal solution to the deterministic problem, with $\mathbf{z}_i$ as the random instance. Hence, it assumes the ability to solve a large number of optimization problems fast to generate the labels $\mathbf{x}_i^*$, or the availability of such historical datasets. We call this approach *optimize then learn*, where "*then*" emphasizes the two-step process of generating labels and training.

To bypass the above assumptions, we utilize an *optimize and learn* strategy where we train a machine learning model $\boldsymbol{\pi}(\mathbf{w}, \mathbf{z})$ in the process of solving a stochastic optimization problem, and without labels. Here, the "*and*" signifies a single-step process. The stochastic primal-dual gradient updates are employed that explicitly take into account the constraints in the optimization problem (1.1), hence enabling *constrained learning*. We test different variations of this approach across different application domains, in Chapters 2–4.

Chapter 2 was published as a conference paper [37] and addresses the problem of providing reactive power compensation for voltage regulation on distribution grids. Using the primal-dual learning approach, the goal is to minimize the Ohmic losses on the grid while keeping the average voltages within the desirable upper and lower bounds. A communication-cognizant DNN architecture was proposed that reduces the real-time communications overhead.

Chapter 3 extends the previous chapter to a journal paper [34]. Also focusing on the problem of optimal reactive power compensation by DERs, many extensions are accommodated – *1)* Underlying grid is modeled as an AC network, *3)* probabilistic chance constraints are enforced using convex relaxations, *3)* approach is compatible to proxies for the original data and partial information, and *4)* A gradient-free version of the approach is suggested.

Lastly, Chapter 4, which was published as a conference paper [33] and a journal [39], presents the application of the primal-dual learning approach on transmission grids for solving the stochastic optimal power flow (SOPF) problem. The goal is to predict the optimal setpoint for generator buses to meet the load demands while satisfying all the operational constraints such as voltage limits and flow limits. Operational constraints are imposed via chance constraints, but instead of convex relaxations, more accurate approximations involving the logistic function are considered.

1.3 DNNs as Digital Twins of Volt/VAR Dynamics

The second part of the thesis concerns itself with the optimal design of local Volt/VAR control rules for smart inverters. Local Volt/VAR control rules are implemented at the individual smart inverter level, and determine the smart inverter reactive power setpoints as a function of local voltage measurements. While the goal is again to provide reactive power compensation, as in Part 1, the setup is vastly different. In the new setup, different DERs interact with the grid using their individual Volt/VAR control rules. Since voltages are affected by reactive power setpoints, the above interaction results in close-loop dynamics, which may or not may not converge to equilibrium. Hence, the main goal under optimal rule design (ORD) is to design stable rules such that the voltages at equilibrium are closer to 1, across S given scenarios. This can be represented via the optimization problem

$$\min_{\mathbf{z}\in\mathcal{Z}} \quad \frac{1}{2S}\sum_{s=1}^{S}\|\mathbf{v}_s^*(\mathbf{z})-\mathbf{1}\|_2^2 \tag{1.2}$$

where $\mathbf{v}_s^*(\mathbf{z})$ is the equilibrium voltage corresponding to the scenarios s, $\mathbf{z}$ is the vector of curve design parameters, and $\mathcal{Z}$ represent the feasible space for $\mathbf{z}$. With the insight that the above optimization problem resembles a DNN training task, we strive to design DNNs that can model the Volt/VAR dynamics i.e., they accept the uncontrollable grid conditions as input, and produce $\mathbf{v}_s^*(\mathbf{z})$ at the output. If these DNNs are parameterized by $\mathbf{z}$ then the above problem does reduce to a DNN training task. This is more desirable than the alternatives of solving bilevel optimization problems or MINLPs, as discussed in Chapters 5 and 6, which constitute Part 2. Note that unlike Part 1, the DNNs do not predict the optimal setpoints, but model the close loop dynamics to output the equilibrium quantities.

Chapters 5 deals with designing *non-incremental* control rules as those in the IEEE 1547.8 standard [4]. Stability and convergence guarantees are presented for both single and multiphase feeders. DNNs modeling the Volt/VAR dynamics are trained using stochastic projected gradient descent (SPGD) updates. A novel MINLP solver is also used to benchmark the performance of the DNN-based approach. The work has been submitted to the IEEE Transactions on Smart Grids [38].

Non-incremental control rules suffer from a stability-performance trade-off, whereby by focusing on stability one loses on the voltage regulation capabilities of the rules, and vice versa. *Incremental* control rules can bypass such a trade-off and obtain better voltage regulation. Chapter 6, which was submitted to a conference [36], discusses the DNN-based design of incremental control rules, for both single and multiphase grids.

1.4 Quantum Approach to Stochastic Binary Constrained Optimization

The third and final part of this thesis deals with constrained binary optimization problems of the form

$$\begin{aligned} \min_{\mathbf{b}\in\{0,1\}^n} \quad & f_0(\mathbf{b}) \\ \text{s.to} \quad & f_m(\mathbf{b}) \leq 0, \quad m = 1 : M. \end{aligned} \tag{1.3}$$

Such problems appear in power systems operations, wireless communications, and machine learning. Solving a general version of the above problem is *NP*-hard. Inspired by the recent innovations in the NISQ era quantum computing, we design a quantum-classical hybrid solver to propose solutions to a stochastic version of (1.3). We present a constrained variational quantum eigensolver (VQE) that solves the above problem using stochastic dual decomposition updates. In addition to the fact that this third part also puts forth approaches to handle optimization problems in power systems operations, it possesses other subtle connections with the previous two parts. VQEs employ parameterized quantum circuits (PQCs), which are quantum circuits configured by a set of parameters. Employing VQEs to solve binary optimization entails finding the optimal values for this set of parameters, a task resembling the training of DNNs. Furthermore, the algorithm developed here to handle stochastic constraints bears similarities in conception, design, and analysis, to the constrained learning methodology employed earlier in the thesis. Lastly, the experience gained from working with Python and its ML libraries facilitated the implementation of the proposed algorithm using Python quantum computing libraries.

This final part of the thesis, as presented in Chapter 7, has been submitted as a conference paper [35].

Chapter 2

Deep Learning for Reactive Power Control of Smart Inverters under Communication Constraints

2.1 Publication Details

S. Gupta, V. Kekatos and M. Jin, ”Deep Learning for Reactive Power Control of Smart Inverters under Communication Constraints”, In *Proc. IEEE Intl. Conf. on Smart Grid Commun., Tempe, AZ, 2020.*

2.2 Abstract

Aiming for the median solution between cyber-intensive optimal power flow (OPF) solutions and subpar local control, this work advocates deciding inverter injection setpoints using deep neural networks (DNNs). Instead of fitting OPF solutions in a black-box manner, inverter DNNs are naturally integrated with the feeder model and trained to minimize a grid-wide objective subject to inverter and network constraints enforced on the average over uncertain grid conditions. Learning occurs in a quasi-stationary fashion and is posed as a stochastic OPF, handled via stochastic primal-dual updates acting on grid data scenarios. Although trained as a whole, the proposed DNN is operated in a master-slave architecture. Its master part is run at the utility to output a condensed control signal broadcast to all inverters. Its slave parts are implemented by inverters and are driven by the utility signal along with local inverter readings. This novel DNN structure uniquely addresses the small-big data conundrum where utilities collect detailed smart meter readings yet on an hourly basis, while in real time inverters should be driven by local inputs and minimal utility coordination to save on communication. Numerical tests corroborate the efficacy of this physics-aware DNN-based inverter solution over an optimal control policy.

2.3 Introduction

Distribution grids are currently challenged by voltage fluctuations due to the proliferation of distributed energy resources (DERs). The voltages experienced at buses of a feeder depend heavily on the power injected or withdrawn, while the power generated by a PV under intermittent cloud coverage may vary by 80% within one-minute intervals [94]. The inverters interfacing DERs have been suggested as a promising fast-responding mechanism and are now allowed to provide reactive power support per the amended IEEE 1547 standard. If properly orchestrated, inverters can regulate nodal voltages and/or reduce ohmic line losses. Nonetheless, coordinating hundreds of inverters in real-time is a formidable task.

The literature on inverter control can be broadly classified into optimization- and learning-based approaches. The former class includes approaches where inverter control is posed as an optimal power flow (OPF) problem. Under a centralized OPF setup [24], [97], the utility reads the values of solar generation and loads, solves an OPF, and communicates the optimal setpoints to inverters. To avoid any cyber overhead, inverter setpoints can be decided using simple Volt/VAR or Watt/VAR control rules driven by local readings [94]. Nonetheless, the equilibria of such rules do not coincide with the sought OPF solutions and can be subpar [52], [46].

Learning-based approaches shift the computational effort offline, and perform numerically less intensive tasks during real-time operation. Learning-based approaches can be further clustered into the *OPF-then-learn* and the *OPF-and-learn* philosophies. According to the former, one first solves a large number of OPF instances parameterized by their inputs (solar/load conditions). The pairs of OPF inputs or instances and OPF minimizers are subsequently used for the ML model to learn the OPF mapping in a supervised manner. In real time, the ML model approximates OPF decisions on the fly as soon as it is presented with a new OPF instance. Under this paradigm, references [19] and [50] use kernel–based regression to learn inverter control rules. DNNs have alternatively been employed to learn OPF solutions under a linearized [79]; or an exact AC grid model [101], [29], [100], [78].

Rather than fitting OPF minimizers, the *OPF-and-learn* paradigm trains an ML model directly through an OPF in a single step. Therefore, it does not require solving multiple OPFs to generate a labeled training set. Under the *OPF-and-learn* paradigm, reference [48] adopts kernel-based learning to design inverter control rules, adjusted to grid conditions in a quasi-stationary fashion. Although rules can be learned using a convex program, the kernel functions have to be specified beforehand. In [99], inverter control rules are optimized along with capacitor status decisions to minimize voltage deviations using a two-timescale reinforcement learning (RL) approach. Nonetheless, no feeder-level constraints are involved. Enforcing network constraints is challenging for learning-based OPF methods. One could heuristically project the ML prediction for the OPF solution [101], [48]. Other approaches to coping with constraints include penalizing constraint deviations [50], [79], [48], [99]; or enforcing constraints in a discounted sense [104]. Reference [104] models inverter policies as DNNs. It successively linearizes feeder constraints and updates policies continuously through communication exchanges between interconnected microgrids. A similar *safe RL* learning scheme is put forth in [98], but with a centralized implementation.

A key promise of designing policies is to alleviate the cyber burden of inverter control. This critical aspect has been largely overlooked by the existing literature. In particular, references [104] and [98], which are most closely related to this work, update policies continuously and require considerable amounts of data to be communicated in real time. To account for this aspect, the contributions of this work are in two fronts: First, inverter policies are modeled as DNNs that are jointly trained in a quasi-stationary fashion, while feeder constraints are enforced explicitly in a stochastic sense. Second, a carefully designed DNN architecture accommodates application scenarios where inverter rules are driven by local measurements as well as a low-bandwidth control signal broadcast by the utility.

Outline: Section 2.4 formulates the task of designing inverter control policies after reviewing an approximate grid model. Section 2.5 adopts a stochastic primal-dual algorithm to find the optimal inverter control policies. Section 2.6 puts forth the novel communication-cognizant DNN-based inverter control architecture. The proposed schemes are evaluated using real-world solar generation and load data on the IEEE 13-bus feeder in Section 2.7. Conclusions along with ongoing and future research directions are discussed in Section 2.8.

Notation: lower- (upper-) case boldface letters denote column vectors (matrices), and calligraphic symbols are reserved for sets. Symbol $^\top$ stands for transposition and $\|\mathbf{x}\|_2$ denotes the ℓ_2-norm of $\mathbf{x}$. Vectors $\mathbf{0}$ and $\mathbf{1}$ are respectively the vectors of all zeros and ones of appropriate dimensions.

2.4 Grid Modeling and Problem Formulation

Consider a feeder with $N+1$ buses, including the substation indexed by 0. Let $p_n + jq_n$ be the complex power injection at bus n. Its active power component can be decomposed as $p_n = p_n^g - p_n^c$, where p_n^g is the solar generation and p_n^c the inelastic load at bus n. Its reactive power component can be similarly expressed as $q_n = q_n^g - q_n^c$. If vectors $(\mathbf{p}, \mathbf{q})$ collect the power injections at all non-substation buses, they can be decomposed as $\mathbf{p} = \mathbf{p}^g - \mathbf{p}^c$ and $\mathbf{q} = \mathbf{q}^g - \mathbf{q}^c$. We refer to the values of (re)active loads and active solar generation at all non-substation buses as *grid conditions*

$$\mathbf{z} := [(\mathbf{p}^c)^\top \quad (\mathbf{q}^c)^\top \quad (\mathbf{p}^g)^\top]^\top. \tag{2.1}$$

Given $\mathbf{z}$, the task of reactive power control by DERs aims at optimally setting $\mathbf{q}^g$ to minimize a feeder-wide objective while complying with network and inverter limitations. Starting with the latter, the reactive power injected by inverter n is limited by a given $\bar{q}_n^g$ due to apparent power limits. Apparent power constraints are local and will be collectively denoted by

$$\mathbf{q}^g \in \mathcal{Q} := \{\mathbf{q} : |q_n^g| \leq \bar{q}_n^g \quad \forall n\}. \tag{2.2}$$

Regarding feeder constraints, the focus is on confining voltages within the regulation range of $[0.97, 1.03]$ per unit (pu). Albeit voltages are nonlinearly related to power injections, for simplicity we adopt a widely used linearized grid model [91]. According to this model, the vector of voltage magnitudes at all N buses is approximately

$$\mathbf{v} = \mathbf{R}\mathbf{p} + \mathbf{X}\mathbf{q} + v_0\mathbf{1} \tag{2.3}$$

where v_0 is the substation voltage, while the symmetric positive semidefinite matrices $(\mathbf{R}, \mathbf{X})$

depend on the feeder and are assumed to be known. If each voltage v_n is to be maintained within $[\underline{v}_n, \overline{v}_n]$, the reactive power injections $\mathbf{q}^g$ should satisfy the network constraints

$$\mathbf{g}(\mathbf{q}^g, \mathbf{z}) := \begin{bmatrix} \mathbf{X}\mathbf{q}^g + \mathbf{y} - \overline{\mathbf{v}} \\ -\mathbf{X}\mathbf{q}^g - \mathbf{y} + \underline{\mathbf{v}} \end{bmatrix} \leq \mathbf{0} \tag{2.4}$$

where vector $\mathbf{y} := \mathbf{R}(\mathbf{p}^g - \mathbf{p}^c) - \mathbf{X}\mathbf{q}^c + v_0\mathbf{1}$ depends on $\mathbf{z}$, and vectors $(\underline{\mathbf{v}}, \overline{\mathbf{v}})$ contain the limits $(\underline{v}_n, \overline{v}_n)$ across buses.

According to the same grid model, ohmic losses on lines can be approximated as a convex quadratic function of power injections as $\mathbf{p}^\top\mathbf{R}\mathbf{p} + \mathbf{q}^\top\mathbf{R}\mathbf{q}$; see [91] for details. Upon defining $\mathbf{b} := 2\mathbf{R}\mathbf{q}^c$, the part of ohmic losses that is dependent on the control variable $\mathbf{q}^g$ can be approximated as

$$\ell(\mathbf{q}^g, \mathbf{z}) = (\mathbf{q}^g)^\top\mathbf{R}\mathbf{q}^g - \mathbf{b}^\top\mathbf{q}^g. \tag{2.5}$$

We henceforth abuse notation and use $\mathbf{q}$ in lieu of $\mathbf{q}^g$. This should not cause any confusion since $\mathbf{q}^c$ has been included in $\mathbf{z}$. DER reactive setpoints $\mathbf{q}$ can be found as the minimizer of

$$\begin{aligned} \min_{\mathbf{q}\in\mathcal{Q}} \quad & \ell(\mathbf{q}, \mathbf{z}) \\ \text{s.to} \quad & \mathbf{g}(\mathbf{q}, \mathbf{z}) \leq \mathbf{0}. \end{aligned} \tag{2.6}$$

Under the linearized grid model, the approximate OPF task of (2.6) is a convex quadratic program (QP). Solving (2.6) can be computationally and communication-wise taxing if $\mathbf{z}$ changes frequently. Moreover, by the time (2.6) is solved and decisions are downloaded to DERs, grid conditions $\mathbf{z}$ may have changed rendering the computed setpoints obsolete.

To account for the uncertainty in $\mathbf{z}$, one may pursue a stochastic formulation such as [97]

$$\begin{aligned}\min_{\mathbf{q}\in\mathcal{Q}} \quad & \mathbb{E}[\ell(\mathbf{q},\mathbf{z})] \\ \text{s.to} \quad & \mathbb{E}[\mathbf{g}(\mathbf{q},\mathbf{z})] \leq \mathbf{0}\end{aligned} \tag{2.7}$$

where the expectation $\mathbb{E}$ is with respect to $\mathbf{z}$. Nonetheless, the obtained 'one-size-fits-all' $\mathbf{q}$ does not adapt to different $\mathbf{z}$'s.

To come up with DER setpoints that are responsive to grid conditions, we resort to *control policies or rules*, where the reactive power setpoint for each inverter n is captured by a function $\pi_n(\mathbf{w}_n;\boldsymbol{\theta}_n)$ acting upon a control input $\mathbf{w}_n$ and is parameterized by vector $\boldsymbol{\theta}_n$. Ideally, inverter control policies should be driven by the complete $\mathbf{z}$, that is $\mathbf{w}_n = \mathbf{z}$ for all n. Nevertheless, that would entail high communication overhead. If the utility knows the complete $\mathbf{z}$, it might as well solve (2.6) and communicate the optimal setpoints to inverters. For an inverter control scheme to be communication-cognizant, the inputs $\mathbf{w}_n$ should primarily involve local readings of $\mathbf{z}$, such as (p_n^g, q_n^g, p_n^g), and possibly few remote entries. Regarding the parameter vectors $\boldsymbol{\theta}_n$'s, these may be unique per inverter or share some entries as detailed in Section 2.6. To capture the aforementioned scenarios, let us abstractly refer to the vector of inverter policies $\pi_n(\mathbf{w}_n;\boldsymbol{\theta}_n)$'s as

$$\mathbf{q}(\mathbf{w}) = \boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}) \tag{2.8}$$

where $\mathbf{w}$ is the union of $\mathbf{w}_n$'s and $\boldsymbol{\theta}$ the union of $\boldsymbol{\theta}_n$'s.

The control policies for DERs can be found jointly by solving the constrained stochastic minimization

$$\begin{aligned} P^* := \min_{\boldsymbol{\theta}:\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta})\in\mathcal{Q}} \quad & \mathbb{E}[\ell(\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}),\mathbf{z})] \\ \text{s.to} \quad & \mathbb{E}[\mathbf{g}(\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}),\mathbf{z})] \leq \mathbf{0} \end{aligned} \tag{2.9}$$

over the parameter vector $\boldsymbol{\theta}$. Problem (2.9) couples policies in two ways. First, for a fixed $\mathbf{z}$, policies are coupled across inverters through the cost and constraint functions since the entries of $\mathbf{q}^g$ appearing in (2.3) and (2.5) are now computed via (2.8). Second, the expectations in (2.7) and (2.9) couple system's performance across OPF instances characterized by $\mathbf{z}$.

Local and linear policies of the form $\pi_n(\mathbf{w}_n;\boldsymbol{\theta}_n) = \boldsymbol{\theta}_n^\top \mathbf{w}_n$ have been previously studied for inverter control [46], [59], [6]. Nonetheless, the optimal policies $q_n(\mathbf{w}_n)$ are not necessarily affine in $\mathbf{w}_n$, especially when $\mathbf{w}_n$ is only a partial observation of $\mathbf{z}$. The grand challenge towards scalable inverter control is to design *nonlinear control curves*. In [48], we dealt with by modeling each $q_n(\mathbf{w}_n)$ as a kernel-based support vector machine (SVM), and designing all rules jointly under an OPF formulation. The advantage of SVM-based policies is that they can be trained to optimality using convex optimization. Nonetheless, selecting the appropriate kernel and control inputs $\mathbf{w}_n$'s can be challenging. Inspired by their field-changing performance in various engineering tasks, here we propose modeling inverter rules using DNNs, and train the parameters $\boldsymbol{\theta}$ in a data-driven physics-aware fashion.

2.5 Primal-dual DNN Learning

Solving (2.9) is challenging since it is a constrained stochastic minimization over a DNN. To train the inverter policy DNN, we adopt the stochastic primal-dual updates of [22], which are briefly reviewed next. Consider the Lagrangian function of (2.9)

$$L(\boldsymbol{\theta};\boldsymbol{\lambda}) = \mathbb{E}[\ell(\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}),\mathbf{z})] + \boldsymbol{\lambda}^\top \mathbb{E}[\mathbf{g}(\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}),\mathbf{z})] \tag{2.10}$$

where $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers corresponding to constraint (2.9). The dual problem can be posed as

$$D^* = \max_{\boldsymbol{\lambda}\geq \mathbf{0}} \min_{\boldsymbol{\theta}:\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta})\in\mathcal{Q}} L(\boldsymbol{\theta};\boldsymbol{\lambda}). \tag{2.11}$$

Standard duality results predicate that $D^* \leq P^*$. When the primal problem is convex, the previous inequality typically holds with equality. Problem (2.9) however is non-convex even if (2.6) is a convex QP, since the DNN mapping $\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta})$ is generally non-convex in $\boldsymbol{\theta}$. Nonetheless [22] establishes that: *i)* under relatively mild conditions satisfied by (2.9), and *ii)* if the underlying DNN architecture is rich enough, the duality gap $P^* - D^*$ is sufficiently small. This motivates solving (2.9) through the primal-dual updates indexed by k [22]

$$\boldsymbol{\theta}^{k+1} = \left[\boldsymbol{\theta}^k - \mu_\theta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^k;\boldsymbol{\lambda}^k)\right]_{\mathcal{Q}} \tag{2.12a}$$

$$\boldsymbol{\lambda}^{k+1} = \left[\boldsymbol{\lambda}^k + \mu_\lambda \nabla_{\boldsymbol{\lambda}} L(\boldsymbol{\theta}^{k+1};\boldsymbol{\lambda}^k)\right]_{+} \tag{2.12b}$$

where the operator $[\cdot]_{\mathcal{Q}}$ projects $\boldsymbol{\theta}^{k+1}$ such that $\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}^{k+1}) \in \mathcal{Q}$ for all $\mathbf{w}$; operator $[\cdot]_+$ ensures $\boldsymbol{\lambda} \geq \mathbf{0}$ at all times; and $(\mu_\theta, \mu_\lambda)$ are positive step sizes. Regarding $[\cdot]_{\mathcal{Q}}$, the DNN output corresponding to q_n^g can be constrained within $[-\bar{q}_n^g, +\bar{q}_n^g]$ by using $\tanh(\cdot)$ as the output activation function and then scaling by the constant $\bar{q}_n^g$.

The updates in (2.12) are complicated by the expectation operator. The pdf of $\mathbf{z}$ (and hence $\mathbf{w}$) may not be known beforehand. Even if it is known, propagating that pdf through nonlinear functions such as $\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}),\mathbf{z})$ is non-trivial. To deal with this, the primal-dual updates of

(2.12) can be surrogated by their stochastic approximation counterparts. In particular, the utility is assumed to have a set of scenarios $(\mathbf{z}^k, \mathbf{w}^k)$ indexed by $k = 1, \ldots, K$, with which the ensemble averages of (2.10) are approximated as $\mathbb{E}[\ell(\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}), \mathbf{z})] \simeq \frac{1}{K}\sum_{k=1}^{K} \ell(\boldsymbol{\pi}(\mathbf{w}^k;\boldsymbol{\theta}), \mathbf{z}^k)$. To simplify the updates of (2.12), the sample averages can be approximated by a single scenario per iteration to yield the *stochastic* primal-dual updates [22]

$$\boldsymbol{\theta}^{k+1} = \left[\boldsymbol{\theta}^k - \mu_\theta\big(\nabla_{\boldsymbol{\theta}}\ell^k - (\nabla_{\boldsymbol{\theta}}\mathbf{g}^k)^\top \boldsymbol{\lambda}^k\big)\right]_{\mathcal{Q}} \tag{2.13a}$$

$$\boldsymbol{\lambda}^{k+1} = \left[\boldsymbol{\lambda}^k + \mu_\lambda \mathbf{g}\left(\boldsymbol{\pi}(\mathbf{w}^k;\boldsymbol{\theta}^{k+1}), \mathbf{z}^k\right)\right]_+ . \tag{2.13b}$$

Here $\nabla_{\boldsymbol{\theta}}\ell^k$ is the gradient of $\ell(\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}), \mathbf{z})$ and $\nabla_{\boldsymbol{\theta}}\mathbf{g}^k$ the Jacobian matrix of $\mathbf{g}(\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}), \mathbf{z})$, both with respect to $\boldsymbol{\theta}$ and evaluated at $(\mathbf{w}^k, \boldsymbol{\theta}^k, \mathbf{z}^k)$. The updates are known to converge to a stationary point of (2.9) for sufficiently small step sizes.

For the objective and constraint functions of (2.4)–(2.5), the needed sensitivities can be computed as

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}\ell^k &= \left(\nabla_{\boldsymbol{\theta}}\boldsymbol{\pi}(\mathbf{w}^k;\boldsymbol{\theta}^k)\right)^\top \left(2\mathbf{R}\boldsymbol{\pi}(\mathbf{w}^k;\boldsymbol{\theta}^k) - \mathbf{b}^k\right)\\
\nabla_{\boldsymbol{\theta}}\mathbf{g}^k &= [\mathbf{X} \quad -\mathbf{X}]^\top \nabla_{\boldsymbol{\theta}}\boldsymbol{\pi}(\mathbf{w}^k;\boldsymbol{\theta}^k).
\end{aligned}$$

Here $\mathbf{b}^k := 2\mathbf{R}(\mathbf{q}^c)^k$ and $\nabla_{\boldsymbol{\theta}}\boldsymbol{\pi}(\mathbf{w}^k;\boldsymbol{\theta}^k)$ is the Jacobian matrix of the DNN output with respect to its weight parameters. The latter can be evaluated using gradient back-propagation across the DNN, a standard tool readily available in all DNN-related software. If the number of available grid scenarios K is relatively small, additional scenarios can be synthesized by applying small perturbations on the available $\mathbf{z}^k$'s. As customary in DNN training, the updates (2.13) can be iterated over multiple epochs or in mini-batch forms.

It is worth contrasting the DNN input $\mathbf{w}$ and the vector of grid conditions $\mathbf{z}$. Despite some possible overlap, the two vectors are used differently. The former one feeds the DNN to compute the setpoints $\mathbf{q}(\mathbf{w}) = \boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta})$. The latter one is involved in the OPF objective and constraint functions, i.e., it appears in $\mathbf{b}$ for computing $\nabla_{\boldsymbol{\theta}}\ell$ and when evaluating $\mathbf{g}(\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta}),\mathbf{z})$. While $\mathbf{z}$ should be known to the utility during training to perform the updates of (2.13), it is not needed during real-time operation. This resonates with the small/big data setup, since a utility has offline access to an extensive smart meter dataset of $\mathbf{z}$'s; yet its control center and each inverter individually are driven by limited real-time data feeds. The updates of (2.13) apply for inverters DNNs of arbitrary architecture. We next particularize the structure of $\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta})$ to comply with communication limitations in inverter control.

2.6 Communication-Cognizant DNN Architecture

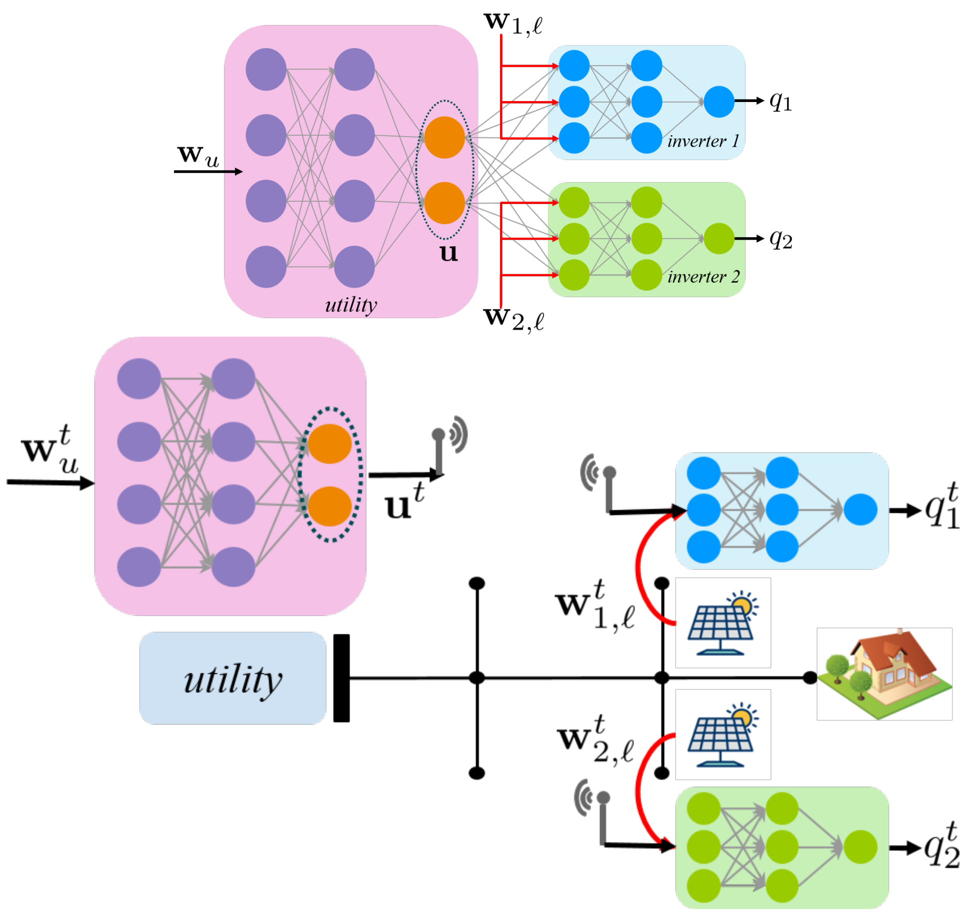


Figure 2.1: *Top*: DNN $\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta})$ is organized in one utility sub-NN and inverter sub-NNs, all trained as a single DNN by the utility offline. *Bottom*: During real-time operation, the utility sub-NN uses real-time data to compute and broadcast the control signal, while inverter sub-NNs are run at inverters.

To coordinate inverters on a tight communication budget, our proposed inverter policy DNN $\boldsymbol{\pi}(\mathbf{w};\boldsymbol{\theta})$ comes with the two-tier architecture shown on Figure 2.1 (top). Its first layers constitute the *utility sub-NN*, while the final layers constitute the *inverter sub-NNs*, one for each inverter. Figure 2.1 shows only two inverters for simplicity. The utility sub-NN (shown in purple) is fully connected, is driven by input $\mathbf{w}_u$, and outputs control $\mathbf{u}$. Inverter sub-NNs (in blue and green) are disconnected from each other and both fed with the common control

$\mathbf{u}$. Each inverter sub-NN is also fed with its own local data $\mathbf{w}_{n,\ell}$. The n-th inverter sub-NN predicts the setpoint q_n.

Inverter policy n can be expressed as $q_n(\mathbf{w}_n) = \pi_n(\mathbf{w}_n; \boldsymbol{\theta}_n)$ where $\mathbf{w}_n = [\mathbf{w}_u^\top \ \mathbf{w}_{n,\ell}^\top]^\top$ and $\boldsymbol{\theta}_n$ collects the DNN parameters for the shared utility sub-NN and inverter sub-NN n. Vectors $\mathbf{w}_{n,\ell}$ may carry local load and solar generation available on bus n. Input $\mathbf{w}_u$ carries information available to the utility control center in real time. Such information can be power flow readings from major distribution lines, transformers, and/or voltage regulators. Vector $\mathbf{w}_u$ may also carry the solar generation from a solar farm or any other DER that is telemetered in real time. Rather than actual grid measurements, vector $\mathbf{w}_u$ may also include predictions the utility can make on grid conditions. For example, that could be the case if the utility uses cameras to monitor cloud coverage as a proxy to solar generation or temperature/humidity readings to load.

During training and given grid scenario $\mathbf{z}^k$, the inputs $\mathbf{w}_u^k$ and $\mathbf{w}_n^k$'s can be: *i)* found readily as partial entries of $\mathbf{z}^k$ (loads and solar generation); *ii)* inferred from $\mathbf{z}^k$ (a line flow can be computed through the power flow equations, or approximated as the sum of all downstream power injections); or *iii)* found through historical data (dataset combining cloud coverage with solar generation). The particular structure of the proposed DNN with individualized inputs and partially connected layers can be easily implemented by skipping and masking connections, respectively.

Although trained as a whole, the inverter policy DNN $\boldsymbol{\pi}(\mathbf{w}; \boldsymbol{\theta})$ is implemented in parts; see bottom panel of Fig. 2.1. After training is completed for the upcoming 30- or 60 min period, the weights corresponding to inverter NNs are downloaded to inverters. A unique component of our DNN architecture is the control signal $\mathbf{u}$, which is broadcast from the utility NN to inverter NNs. To save on downlink (utility to inverters) communications, signal $\mathbf{u}$ is designed to be much shorter than $\mathbf{w}_u$. Considering that $\mathbf{u}$ is actually designed

Algorithm 1 Inverter control through DNN-based policies

```
Training
1: Collect grid scenarios $\{\mathbf{z}^k\}_{k=1}^K$ from smart meter data
2: Collect or calculate DNN inputs $\{\mathbf{w}^k\}_{k=1}^K$
3: Initialize $\boldsymbol{\theta}^0$ and $\boldsymbol{\lambda}^0$
4: for all $K$ scenarios and $E$ epochs do
5:     Update $\boldsymbol{\theta}$ using (2.13a)
6:     Update $\boldsymbol{\lambda}$ using (2.13b)
7: end for
8: Download $\boldsymbol{\theta}$ parameters to inverter sub-NNs
Real-time operation
1: for $t = 0, 1, \ldots, T$, do
2:     Utility receives $\mathbf{w}_u^t$ from real-time telemetry
3:     Feed $\mathbf{w}_u^t$ to utility sub-NN to compute $\mathbf{u}^t$
4:     Utility broadcasts $\mathbf{u}^t$ to inverters
5:     for each inverter $n$ do
6:         Inverter $n$ reads $\mathbf{u}^t$ and local data $\mathbf{w}_n^t$
7:         Feed $(\mathbf{u}^t, \mathbf{w}_n^t)$ to inverter sub-NN to decide $q_n^t$
8:     end for
9: end for
```

along with the operation of inverter sub-NNs through the OPF of (2.9), this signal carries all the information the utility can provide to coordinate inverters in a condensed form. Its broadcast nature further contributes to communication savings. The steps involved during the training and real-time operation of the proposed DNN are summarized in Algorithm 1.

This DNN architecture can cater to a wide range of communication specifications. If no downlink communication is allowed in real time, the utility sub-NN can be ignored all together and inverter sub-NNs are driven based on local inputs. If downlink bandwidth is abundant, inverter sub-NNs can be dropped and inverter setpoints can be decided by the utility sub-NN in real time. Practical application scenarios are expected to lie somewhere between these two extremes, whence the hybrid architecture of Fig. 2.1 becomes relevant.

2.7 Numerical Tests

The proposed DNN-based inverter control was evaluated on a single-phase version of the IEEE 13-bus feeder. Real-world active load data was extracted for March 1, 2018, on a one-minute resolution from Pecan Street. Solar generation data was also added to buses $\{1, 5, 9, 10, 11, 12\}$, out of which buses $\{9, 12\}$ were equipped with inverters. Load time series were scaled so that monthly peaks were 7.5 times the benchmark values. The same ratio was used to scale solar. Reactive loads were added with lagging power factors sampled from a uniform distribution between 0.9 and 1. The utility was assumed to have telemetry $\mathbf{w}_u$ for the active line flows feeding buses $\{2, 3, 7\}$ from their parent buses.

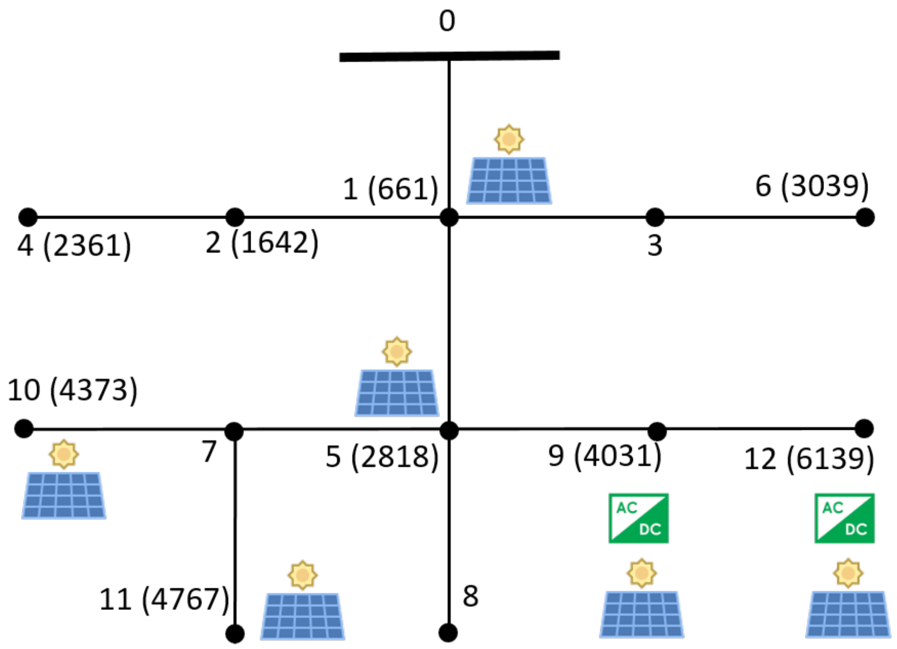


Figure 2.2: The IEEE 13-bus feeder. Numbers in parentheses indicate the house index from the Pecan Street dataset mapped to each bus.

The utility sub-NN was constructed using an input layer of dimension 3 and an output layer $\mathbf{u}$ of dimension 1. Inverter sub-NNs were made up of one input, hidden, and output layers of dimensions 5, 6 and 1, respectively. The local readings $\{p_n, q_n^c\}$ along with $\mathbf{u}$ were fed as inputs to each inverter sub-NN n. Initial values for DNN parameters were uniformly sampled from the range $[-0.1, 0.1]$ and updated using Adam with a learning rate of 0.01. The dual variables were all initialized at 0 and updated with step sizes of 1 that decayed with the square-root of the iteration index [61]. Our approach was contrasted with an optimal policy $\mathbf{q}(\mathbf{w})$ that directly solves (2.7) without being confined to a DNN parameterization using dual decomposition [61]. As in (2.7), the optimal policy regulates the average rather than the instantaneous voltages.

We assumed one-hour long control periods. Training scenarios were obtained from the 60 one-minute data observed over the preceding control period. The original grid scenarios were augmented by adding zero-mean additive white Gaussian noise to generate a total of $K = 240$ scenarios. All scenarions were then randomly shuffled. The variance of the additive noise was decided on the basis of training samples observed and was set to 10^{-6} pu for low-solar and 10^{-2} pu for high-solar hours. DNN $\boldsymbol{\pi}(\mathbf{w}; \boldsymbol{\theta})$ was trained using Alg. 1 for 30 epochs.

Figure 2.3 shows the average losses obtained during training. The losses under our solution were found to be only slightly larger than those attained by the optimal policy. During the low-solar scenario, the base case without any reactive power compensation does not experience any voltage excursions. Therefore, both the optimal policy and our solution focus on decreasing the average losses. On the other hand, when solar generation is high, the basecase experiences high voltage excursions as seen in panel 3. Consequently, the optimal policy and our solution focus on lowering average voltages by withdrawing reactive power at the expense of increased ohmic losses. As demonstrated by the third panel, the proposed scheme attained voltage deviations close to those achieved by the optimal policy. This near-optimal behavior is also shown in the bottom panel presenting the convergence of dual variables for the active constraint on bus 11 during 1:00–2:00 pm.

The DNNs trained over 12:00–1:00 am and 1:00–2:00 pm were tested on the subsequent hours 1:00–2:00 am and 2:00–3:00 pm, respectively. The results are presented in Fig. 2.4. The proposed scheme again closely matches the performance of the optimal policy in terms of both minimizing losses and imposing voltage constraints. This is remarkable especially because the optimal policy has access to perfect forecasts and incurs a large real-time communication overhead, while the DNN-based scheme is trained only on historical data and requires only 1 data point to be transmitted in real-time.

2.8 Conclusions and Ongoing Work

This work has introduced nonlinear control policies for inverter setpoints via a novel two-tier communication-cognizant DNN architecture. The DNN consists of a utility sub-NN and inverter sub-NNs, all jointly trained at the utility at the beginning of every control period, while explicitly incorporating average feeder constraints via primal-dual learning. Upon training, the weights of inverter sub-NNs are downloaded to inverters for real-time implementation. Inverter sub-NNs are driven by local inputs and a control signal broadcast by the utility. Depending on communication specifications, the proposed architecture can accommodate from purely local to centralized and hybrid protocols. Tests on real-world data validate this methodology is capable of reducing ohmic losses and enforcing feeder constraints with little communication overhead. Furthermore, the proposed DNN-based policies perform comparably to stochastic approximation-based optimal policies during both training and testing.

These promising results set the foundations for relevant generalizations. We are currently working on the following directions: *d1)* Model-free primal-dual learning of DNNs that does not require explicit knowledge of the feeder topology, parameters, and/or precise loading conditions during training; *d2)* Chance-constraint formulations; *d3)* Quantify the performance of the proposed DNN-based approach when compared to the optimal policy; *d4)* incorporating exact AC feeder models; and *d5)* testing on larger feeders to demonstrate scalability.

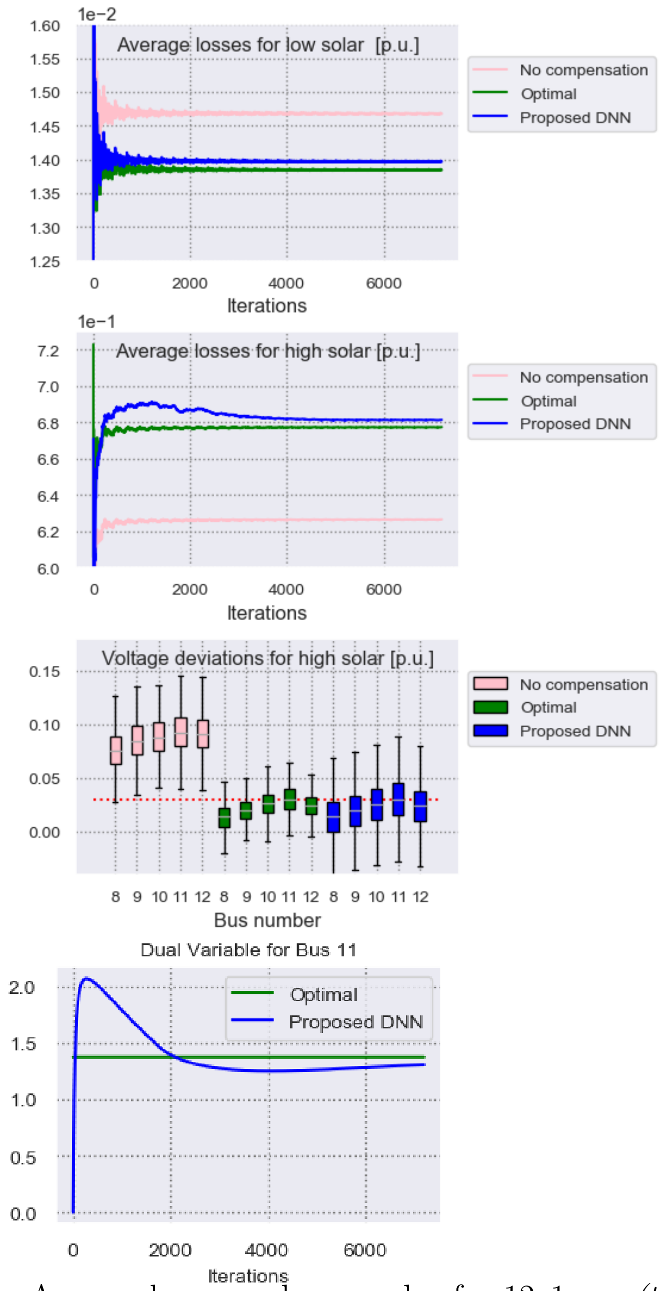


Figure 2.3: Training: Average losses under no solar for 12–1 am *(top)* and high solar for 1–2 pm *(second)*. Voltage excursions for 1–2 pm *(third)*. Dual variable for active constraint on bus 11 for 1–2 pm *(bottom)*

.

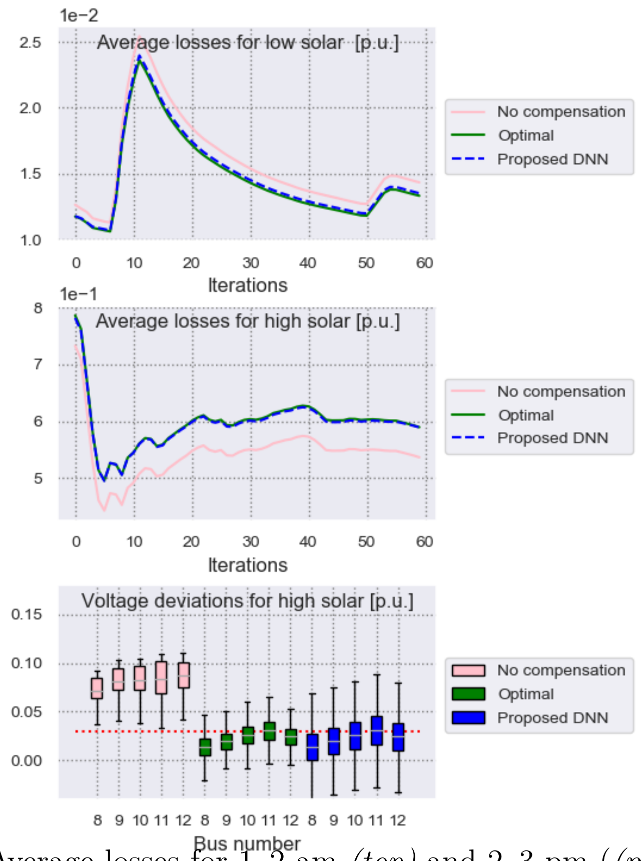


Figure 2.4: Testing. Average losses for 1–2 am *(top)* and 2–3 pm (*(middle)*); voltage excursions for 2–3 pm *(bottom)*.

Chapter 3

Controlling Smart Inverters using Proxies: A Chance-Constrained DNN-based Approach

3.1 Publication Details

S. Gupta, V. Kekatos, and M. Jin.,“ Controlling smart inverters using proxies: A chance-constrained DNN-based approach.” *IEEE Trans. Smart Grid*, 13(2):1310–1321, March 2022.

3.2 Abstract

Coordinating inverters at scale under uncertainty is the desideratum for integrating renewables in distribution grids. Unless load demands and solar generation are telemetered frequently, controlling inverters given approximate grid conditions or proxies thereof becomes a key specification. Although deep neural networks (DNNs) can learn optimal inverter

schedules, guaranteeing feasibility is largely elusive. Rather than training DNNs to imitate already computed optimal power flow (OPF) solutions, this work integrates DNN-based inverter policies into the OPF. The proposed DNNs are trained through two OPF alternatives that confine voltage deviations on the average and as a convex restriction of chance constraints. The trained DNNs can be driven by partial, noisy, or proxy descriptors of the current grid conditions. This is important when OPF has to be solved for an unobservable feeder. DNN weights are trained via back-propagation and upon differentiating the AC power flow equations. An alternative gradient-free variant is also put forth, which requires only a power flow solver and avoids computing gradients. Such variant is practically relevant when calculating gradients becomes cumbersome or prone to errors. Numerical tests compare the DNN-based inverter control schemes with the optimal inverter setpoints in terms of optimality and feasibility.

3.3 Introduction

The high penetration of DERs (such as rooftop photovoltaics, batteries, and demand response devices) introduces additional variability in distribution grid operation. Uncontrolled variations in power injections can in turn induce abrupt fluctuations in nodal voltages. Fortunately, the smart inverters interfacing DERs with the grid can propel their integration by additionally providing reactive power support. The coordinated control of DERs across a feeder can be formulated as an OPF [24], [97], [30], [31]. If the grid is modelled using the AC power flow equations, tackling this OPF using conventional optimization-based solvers becomes formidable as DERs increase in numbers and need to be redispatched frequently under dynamic conditions. Therefore, operators may not have the time and computational resources to solve an AC-OPF every few seconds. At the same time, solving an OPF presumes all problem inputs (load demands and solar generation) to be precisely known. Nonetheless, such parameters are oftentimes described stochastically, observed under noise and delays, or the operator can monitor only few of them in real time. Therefore, even if an operator can afford solving an AC-OPF every few minutes, it may not know all loads and solar generation at the level required by the AC-OPF. The need to compute reasonable DER control decisions in real time and without knowing the current grid conditions in full detail is the driving motivation of this work.

Alternatively, recent literature advocates the use of machine learning (ML) models to solve minimization problems under the *learning-to-optimize* paradigm. Due to the rich modeling and fast inference capabilities of ML models, learning-to-optimize becomes relevant to scenarios where large-scale non-linear optimization tasks have to be solved frequently enough and/or under uncertain or partially observed inputs. ML-based schemes for tackling the OPF have already been explored and can be broadly classified into the *OPF-then-learn* and *OPF-and-learn* categories. Methods within the former category involve two steps. They first generate a labelled training dataset by solving a large number of OPFs. The features and labels of this dataset consist of the OPF input parameters and outputs (optimal DER setpoints), respectively. During the second step, an ML model is trained to predict the dataset labels in the conventional supervised manner. Within the *OPF-then-learn* category, kernel–based regression has been employed to learn inverter control rules in [50], physics-informed stacked extreme learning has been utilized for OPFs [57], while DNNs have been trained to predict OPF solutions under a linearized [79, 95] and the exact AC grid models [101], [29], [100], [78].To expedite the first step, the sensitivity-informed learning method of [86] and [87] trains a DNN to match not only the OPF minimizers, but also their partial derivatives with respect to the OPF inputs. Despite the data efficiency enhancement offered by sensitivity-informed learning, the OPF-then-learn strategy lacks feasibility guarantees and presumes OPF input parameters are deterministic and known. Furthermore, generating labels by solving several OPFs incurs a significant computational overhead. Consequently, the OPF-then-learn strategy is not well suited for scenarios where the optimal policies need to be re-learned frequently on account of changing underlying data distribution.

Instead of this two-step approach of first generating OPF labels and then training the ML model to fit these labels in a least-squares (LS) sense, *OPF-and-learn* approaches learn the ML model directly while solving the OPF in a single step. This is achieved by altering the optimization algorithm used for training the ML model. Rather than fitting OPF solutions in the LS sense, the training algorithm uses the objective or the Lagrangian function of the OPF at hand. By training the ML model directly using the cost and constraint functions of a stochastic OPF, we avoid the computationally expensive step of solving numerous OPF instances beforehand to generate a labeled dataset. Because the ML model is trained now over several OPF instances, it is particularly suited for *stochastic* OPF formulations, where one is interested on the average or probabilistic performance of the learned OPF decisions or policies over uncertain and/or time-varying conditions. Hence, methods within the *OPF-and-learn* category are more appropriate for dynamic applications where ML models need to be continuously retrained. Under the *OPF-and-learn* paradigm, reference [48] adopts support vector machines (SVMs) to design inverter control rules, adjusted to grid conditions in a quasi-stationary fashion. Albeit SVM-based rules can be learned via a convex program, kernel functions have to be specified beforehand. In [99], inverter control rules are optimized along with capacitor status decisions to minimize voltage deviations using a two-timescale reinforcement learning (RL) approach yet no feeder-level constraints are involved. Enforcing network constraints is challenging for ML-based OPF methods. One could heuristically project the ML prediction for the OPF solution [101], [48], or consider penalizing constraint deviations [50], [79], [99], or include deterministic constraints per training sample and solve using dual approximation [95].

An alternative for dealing with constraints in the learning-to-optimize process is through the discounted return functions used in RL approaches. In this context, reference [104] updates DNN-based inverter policies continuously by successively linearizing feeder constraints. A similar *safe RL* learning scheme is put forth in [98], which focuses on regulating the number of voltage violations across nodes through a method of multipliers strategy. However, both these works updated the policy parameters by solving an optimization problem at every update step that might be computationally intensive, restricting their applicability on dynamic scenarios. Secondly, both [104] and [98] focus on scenarios where measurements across feeder nodes are available to policies in real time. Finally, RL-based approaches are in general more complex in implementation, which can be hindering their adoption by grid operators; e.g., the *safe RL* strategy of [98] involves nine different DNNs.

Primal-dual learning [22], provides a computationally less intensive alternative to RL for handling feeder constraints. Different from [104] and [98], primal-dual learning involves simpler gradient-based updates of the policy parameters and the related dual variables alike. Stochastic primal-dual updates were first applied towards learning the optimal control policies for smart inverters to enforce averaged voltage constraints in the conference precursor of our work [32].

Contributions: Building on the *OPF-and-learn* approach of [32], this work trains a DNN that learns a stochastic inverter control policy to provide near-optimal setpoints in real-time and without fully knowing the current grid conditions. The contributions over [32] extend in four fronts. Firstly, the underlying feeder is represented using the exact AC model rather than the approximate linearized model [93] previously employed in [32]. This extension is non-trivial as the gradients needed for the stochastic primal-dual updates of the DNN are now found in an indirect manner using the underlying AC power flow equations and the inverse function theorem. Secondly, we illustrate the versatility of stochastic primal-dual updates as they can cope with probabilistic voltage constraints via convex restrictions [73]. Thirdly, we demonstrate how primal-dual learning can also be used when the DNN has to be fed with partial information during operation. This adheres to practical setups where the utility might have real-time telemetry only over a subset of grid locations. Fourthly, we propose gradient-free counterparts of the primal-dual updates that train the DNN knowing only the values of voltages and losses, and not their gradients with respect to the control variables. Such approaches are useful when the feeder model is known but complex (due to the presence of transformers, capacitors, ZIP loads) and differentiation becomes perplex, but the utility has access to a power flow solver.

Paper outline: The rest of the paper is organized as follows. Section 3.4 formulates the task of DNN-based smart inverter control, and puts forth an *averaged* and a *probabilistic* scheme. Section 3.5 elaborates on primal-dual DNN learning for both schemes. Section 3.5 calculates the gradients needed for the stochastic updates, while the gradient-free implementation is presented in Section 3.6. The novel DNN-based inverter control strategies are evaluated using real-world data on the IEEE 37-bus feeder in Section 3.7. Conclusions are drawn in Section 3.8.

Notation: Lower- (upper-) case boldface letters denote column vectors (matrices), and calligraphic symbols are reserved for sets. Symbol $^\top$ stands for transposition and $\|\mathbf{x}\|_2$ denotes the ℓ_2-norm of $\mathbf{x}$. Vectors $\mathbf{0}$ and $\mathbf{1}$ are respectively the vectors of all zeros and ones of appropriate dimensions.

3.4 Problem Formulation

Consider a feeder with $N+1$ buses. The substation is indexed by 0 and the remaining buses comprise the set $\mathcal{N} := \{1, \dots, N\}$. Let $p_n + jq_n$ be the complex power injection at bus n. Its active power component can be decomposed as $p_n = p_n^g - p_n^c$, where p_n^g is the solar generation and p_n^c the inelastic load at bus n. Its reactive power component can be similarly expressed as $q_n = q_n^g - q_n^c$. The vectors $(\mathbf{p}, \mathbf{q})$ collecting the power injections at all non-substation buses can be decomposed as $\mathbf{p} = \mathbf{p}^g - \mathbf{p}^c$ and $\mathbf{q} = \mathbf{q}^g - \mathbf{q}^c$. For simplicity, it is assumed that each bus hosts at most one inverter, and so index n will be henceforth used for buses and inverters interchangeably.

Let vector parameter $\boldsymbol{\theta} := \{\mathbf{p}^c, \mathbf{q}^c, \mathbf{p}^g\} \subseteq \mathbb{R}^{3N}$ collect the loads (active and reactive) and active solar generation at all non-substation buses. We will henceforth term $\boldsymbol{\theta}$ as the vector of *grid conditions*. Given $\boldsymbol{\theta}$, the task of reactive power control by DERs aims at optimally setting $\mathbf{q}^g$ to minimize a feeder-wide objective while complying with network and inverter limitations. Starting with the latter, the reactive power injected by inverter n is limited by its given apparent power limit $\bar{s}_n$. Apparent power constraints are local and will be collectively denoted by

$$\mathbf{q}^g \in \mathcal{Q}_{\boldsymbol{\theta}} := \left\{ \mathbf{q}^g : |q_n^g| \leq \sqrt{\bar{s}_n^2 - (p_n^g)^2} \ \forall n \right\}. \tag{3.1}$$

where the subscript in $\mathcal{Q}_{\boldsymbol{\theta}}$ denotes that the feasible space changes as the value of solar generation $\mathbf{p}_g$ changes.

The task of coordinating inverters can be centrally handled by the utility operator. The operator finds the reactive power setpoints for DERs by minimizing ohmic losses on distribution lines while maintaining voltage magnitudes within per-bus bounds $[\underline{\mathbf{v}}, \overline{\mathbf{v}}]$ as

$$\min_{\mathbf{q} \in \mathcal{Q}_{\boldsymbol{\theta}}} \quad \ell(\mathbf{q}, \boldsymbol{\theta}) \tag{3.2}$$

$$\text{s.to} \quad \underline{\mathbf{v}} \leq \mathbf{v}(\mathbf{q}, \boldsymbol{\theta}) \leq \overline{\mathbf{v}}.$$

where $\mathbf{v}$ is the vector of bus voltage magnitudes. We will henceforth refer to voltage magnitudes as voltages unless stated otherwise. We slightly abuse notation and use $\mathbf{q}$ in lieu of $\mathbf{q}^g$ to unclutter notation, as $\mathbf{q}^c$ is included in $\boldsymbol{\theta}$ anyway. Note that expressions $\ell(\mathbf{q}, \boldsymbol{\theta})$ and $\mathbf{v}(\mathbf{q}, \boldsymbol{\theta})$ capture the dependence of losses and nodal voltages on the reactive setpoints of DERs $\mathbf{q}$ under the current grid conditions $\boldsymbol{\theta}$. It is assumed that the feeder model and the participating inverters are known and remain fixed throughout the control period.

Solving (3.2) can be computationally and communication-wise taxing if $\boldsymbol{\theta}$ changes frequently. Moreover, by the time (3.2) is solved and optimal setpoints are downloaded to DERs, grid conditions $\boldsymbol{\theta}$ may have changed rendering the computed setpoints obsolete [51], [97]. To account for the uncertainty in $\boldsymbol{\theta}$, we propose two stochastic formulations. The first formulation replaces $\ell(\mathbf{q}, \boldsymbol{\theta})$ and $\mathbf{v}(\mathbf{q}, \boldsymbol{\theta})$ with their averages:

$$\begin{aligned} \min_{\mathbf{q} \in \mathcal{Q}_{\boldsymbol{\theta}}} \quad & \mathbb{E}[\ell(\mathbf{q}, \boldsymbol{\theta})] \\ \text{s.to} \quad & \underline{\mathbf{v}} \leq \mathbb{E}[\mathbf{v}(\mathbf{q}, \boldsymbol{\theta})] \leq \overline{\mathbf{v}} \end{aligned} \tag{3.3}$$

where the expectation $\mathbb{E}$ is with respect to $\boldsymbol{\theta}$. We refer to (3.3) as the *averaged formulation*. While the averaged formulation takes care of uncertainties in $\boldsymbol{\theta}$, the obtained setpoints may violate the voltage limits in (3.2) quite frequently. This undesirable behavior results from the fact that constraining the average value of voltages does not provide strong guarantees on their per-instance values. Nevertheless, the averaged formulation has an attractive structure that permits straightforward stochastic gradient descent (SGD)-based steps to arrive at the optimal setpoints as we will see later.

A more conservative approach is possible through the *probabilistic formulation*

$$\min_{\mathbf{q}\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad \mathbb{E}[\ell(\mathbf{q},\boldsymbol{\theta})] \tag{3.4a}$$

$$\text{s.to} \quad \Pr\left[\underline{v}_n \geq v_n(\mathbf{q},\boldsymbol{\theta})\right] \leq \alpha,\ \forall n \in \mathcal{N} \tag{3.4b}$$

$$\Pr\left[\overline{v}_n \leq v_n(\mathbf{q},\boldsymbol{\theta})\right] \leq \alpha,\ \forall n \in \mathcal{N} \tag{3.4c}$$

where (3.4b)–(3.4c) ensure each bus voltage remains within the desired limits with a probability of at least $1-\alpha$ on each side. Here $\alpha \in (0,1)$ is a small *violation probability.* In contrast to (3.3), the formulation in (3.4) focuses on restricting the frequency of occurrence of voltage violations. Problem (3.4) can be rewritten as

$$\min_{\mathbf{q}\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad \mathbb{E}[\ell(\mathbf{q},\boldsymbol{\theta})] \tag{3.5a}$$

$$\text{s.to} \quad \mathbb{E}\left[\mathbb{1}(\underline{v}_n - v_n(\mathbf{q},\boldsymbol{\theta}))\right] \leq \alpha,\ \forall n \in \mathcal{N} \tag{3.5b}$$

$$\mathbb{E}\left[\mathbb{1}(v_n(\mathbf{q},\boldsymbol{\theta}) - \overline{v}_n)\right] \leq \alpha,\ \forall n \in \mathcal{N} \tag{3.5c}$$

where the indicator function $\mathbb{1}(x)$ is defined as

$$\mathbb{1}(x) = \begin{cases} 1 & ,\ x \geq 0 \\ 0 & ,\ x < 0 \end{cases} . \tag{3.6}$$

The probabilistic formulation is difficult to handle since the indicator function is neither convex nor differentiable. In quest of workable alternatives, Section 3.5 pursues convex approximations of constraints (3.5b)–(3.5c).

For now, let both formulations be represented by the general stochastic program

$$\min_{\mathbf{q}\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad \mathbb{E}[\ell(\mathbf{q},\boldsymbol{\theta})] \tag{3.7}$$

$$\text{s.to} \quad \mathbb{E}[\mathbf{g}(\mathbf{q},\boldsymbol{\theta})] \leq \mathbf{0}.$$

Note that solving (3.7) results in a single 'one-size-fits-all' $\mathbf{q}$ that does not adapt to different $\boldsymbol{\theta}$'s. To render DER setpoints responsive to grid conditions, we resort to a *control policy*, where the reactive setpoints $\mathbf{q}$ are captured by a function $\mathbf{q} = \pi(\boldsymbol{\theta}; \mathbf{w})$, which is parameterized by $\mathbf{w}$.

Ideally, the control policy is driven by the vector of grid conditions $\boldsymbol{\theta}$. Nevertheless, during real-time operation, the operator controlling the DERs may not be able to observe the complete $\boldsymbol{\theta}$. Instead, it may have to act upon a proxy $\boldsymbol{\phi}$ of the actual $\boldsymbol{\theta}$. The DER control policy driven by $\boldsymbol{\phi}$ can then be found by solving the constrained stochastic minimization

$$\begin{aligned} \min_{\mathbf{w}:\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w})\in\mathcal{Q}_{\boldsymbol{\theta}}} \quad & \mathbb{E}[\ell(\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})] \\ \text{s.to} \quad & \mathbb{E}[\mathbf{g}(\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})] \leq \mathbf{0}. \end{aligned} \tag{3.8}$$

The DER control policies found through (3.8) are adaptive to the proxy vector $\boldsymbol{\phi}$ and the optimization is over the parameters $\mathbf{w}$. Policies account for the uncertainty over $\boldsymbol{\theta}$, and correspondingly $\boldsymbol{\phi}$. Note that the expectations in (3.8) couple the system's performance across OPF instances of $\boldsymbol{\theta}$. The notation $\ell(\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})$ captures the fact that the control policy is fed by proxy $\boldsymbol{\phi}$ to determine $\mathbf{q}$, but of course ohmic losses depend on the actual grid conditions $\boldsymbol{\theta}$.

The proxy vector $\boldsymbol{\phi}$ can be chosen to represent the operational setup for which the control policies are being designed. In the absence of real-time measurements from all nodes, and/or to save on communication overhead, vector $\boldsymbol{\phi}$ can consist of active line flows from distribution lines [32]. Meteorological data such as solar irradiance and ambient temperature, which serve as surrogates for $\mathbf{p}$, can also be included in $\boldsymbol{\phi}$. One can also explore convolutional neural networks (CNNs)-based policies that accept sky images in place of solar irradiance measurements as inputs to be included in $\boldsymbol{\phi}$. Similarly, the proxy vector $\boldsymbol{\phi}$ can also represent partial, delayed, or noisy data on the grid conditions, or even aggregate versions of them. In Section 3.7, a more straightforward scenario is explored whereby measurements from a subset of buses in $\mathcal{N}$ are assumed to be available in real-time, resulting in $\boldsymbol{\phi} \subset \boldsymbol{\theta}$.

Previous works have studied linear inverter control policies of the form $\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\phi}$; see e.g., [46], [59], [6]. Nonetheless, the optimal policy $\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w})$ is not necessarily affine in $\boldsymbol{\phi}$, especially when $\boldsymbol{\phi}$ is a proxy for $\boldsymbol{\theta}$. The grand challenge towards scalable inverter control is to design *nonlinear control policies*. To this end, in [48], we modeled $\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w})$ as a support vector machine (SVM) and designed the policy through an OPF formulation. The advantage of SVM-based policies is that they can be trained to optimality using convex optimization. Nonetheless, selecting the appropriate kernel and control input $\boldsymbol{\phi}$ can be challenging. Inspired by their field-changing performance in various engineering tasks, here we propose modeling the DER control policy $\mathbf{q} = \boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w})$ by a DNN. The proxy vector $\boldsymbol{\phi}$ of grid conditions $\boldsymbol{\theta}$ is fed as an input to the DNN. Vector $\mathbf{w}$ carries the weights of the DNN across all layers. The output of the DNN $\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w})$ predicts the sought inverter setpoints $\mathbf{q}$. Figure 3.2 provides a schematic of the architecture. We propose learning weights $\mathbf{w}$ in a data-driven physics-aware fashion.

Figure 3.1 depicts the real-time operation of the proposed control strategy. The smart inverters to be controlled are located on buses 2 and 8. The proxy vector $\boldsymbol{\phi} \subset \boldsymbol{\theta}$, consisting of $\{p_n^c, q_n^c, p_n^g\}$ measurements from buses 2, 3, 4 and 8, is transmitted to the operator. The operator feeds $\boldsymbol{\phi}$ as an input to the DNN-based policy and predicts setpoints $\mathbf{q}$. These setpoints are then broadcast to DER inverters for implementation. It is worth emphasizing here that although the operator needs to know pairs of $(\boldsymbol{\theta}, \boldsymbol{\phi})$ during training, the DNN-based policy operates solely on $\boldsymbol{\phi}$.

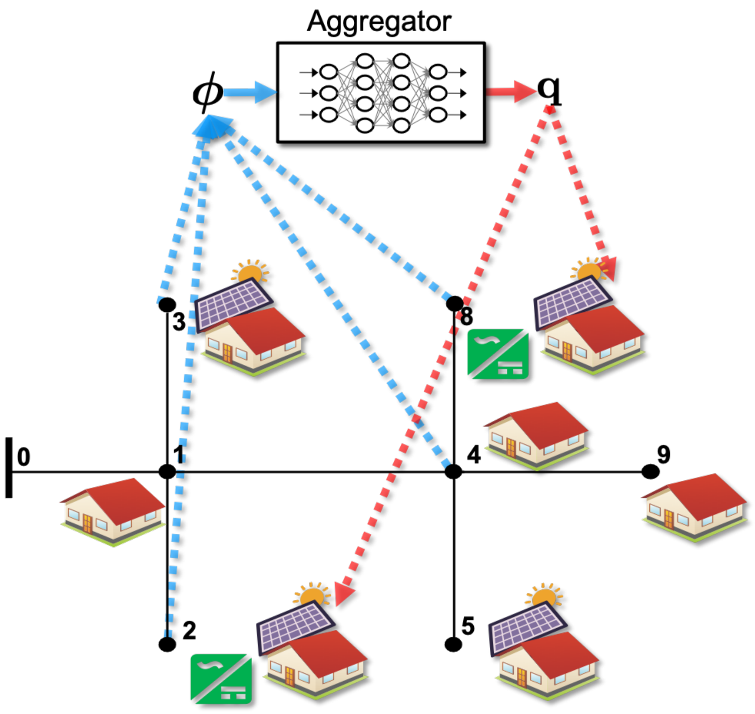


Figure 3.1: Real-time operation of the proposed DER inverter control scheme. Proxy vector $\boldsymbol{\phi}$ consisting of measurements from nodes $\{2, 3, 4, 8\}$ is transmitted to the operator. The DNN-based policy acts upon $\boldsymbol{\phi}$ to predict setpoints $\mathbf{q}$, which are then broadcast to the inverters at buses 2 and 8.

3.5 Primal-Dual DNN Training

The stochastic formulation in (3.8) is challenging to solve on account of the expectation operator in both the objective and constraints. Computing the needed expectations requires knowing the probability density functions (pdf) of $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. Even if these pdfs are known, computing the expectations is still non-trivial granted the policies $\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w})$ are non-linear in $\boldsymbol{\phi}$. These complications promulgate a stochastic approximation approach towards solving (3.8). In the conventional machine learning setup, the weights of a DNN are found by minimizing a data-fitting loss function under no constraints via stochastic gradient descent. Here, to accommodate constraints, we adopt the stochastic primal-dual updates of [22] as presented next.

Consider the Lagrangian function of the problem in (3.8)

$$L(\mathbf{w};\boldsymbol{\lambda}) := \mathbb{E}[\ell(\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})] + \boldsymbol{\lambda}^\top \mathbb{E}[\mathbf{g}(\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})] \tag{3.9}$$

where $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers corresponding to the constraints in (3.8). Vector $\boldsymbol{\lambda}$ concatenates the multipliers $\overline{\boldsymbol{\lambda}}$ and $\underline{\boldsymbol{\lambda}}$ associated with the lower and upper voltage limits in (3.3) and (3.5). A stationary point for the related dual problem

$$D^* := \max_{\boldsymbol{\lambda}\geq \mathbf{0}} \min_{\mathbf{w}:\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w})\in\mathcal{Q}_{\boldsymbol{\theta}}} L(\mathbf{w};\boldsymbol{\lambda}) \tag{3.10}$$

can be obtained iteratively using the primal-dual updates indexed by k (cf. [22]):

$$\mathbf{w}^{k+1} := \left[\mathbf{w}^k - \mu_w \nabla_{\mathbf{w}} L(\mathbf{w}^k;\boldsymbol{\lambda}^k)\right]_{\mathcal{Q}_{\boldsymbol{\theta}}} \tag{3.11a}$$

$$\boldsymbol{\lambda}^{k+1} := \left[\boldsymbol{\lambda}^k + \mu_\lambda \nabla_{\boldsymbol{\lambda}} L(\mathbf{w}^{k+1};\boldsymbol{\lambda}^k)\right]_+ \tag{3.11b}$$

where (μ_w, μ_λ) are positive step sizes. Here primal variables are updated through projected gradient descent steps on the Lagrangian function. Dual variables are updated through projected gradient ascent steps again on the Lagrangian function. The operator $[x]_+ = \max\{x, 0\}$ is applied entry-wise and ensures $\boldsymbol{\lambda} \geq \mathbf{0}$ at all times.

The operator $[\cdot]_{\mathcal{Q}_{\boldsymbol{\theta}}}$ projects $\mathbf{w}^{k+1}$ such that $\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}^{k+1}) \in \mathcal{Q}_{\boldsymbol{\theta}}$ for all $\boldsymbol{\phi}$. A direct way to confine the DNN output q_n^g within $\pm\sqrt{\overline{s}_n^2 - (p_n^g)^2}$ is to use the hyperbolic tangent (tanh) as the output activation function and then scale the output by $\sqrt{\overline{s}_n^2 - (p_n^g)^2}$. While the apparent power limit $\overline{s}_n$ is known *a priori*, the solar generation p_n is available to the DNN as an input. The required scaling is easily accommodated by minor architectural modifications to the activation layer of the DNN. Figure 3.2 illustrates this process for a single-output neuron. Ensuring that $\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}) \in \mathcal{Q}_{\boldsymbol{\theta}}$ at all times obviates the need for projecting the weight

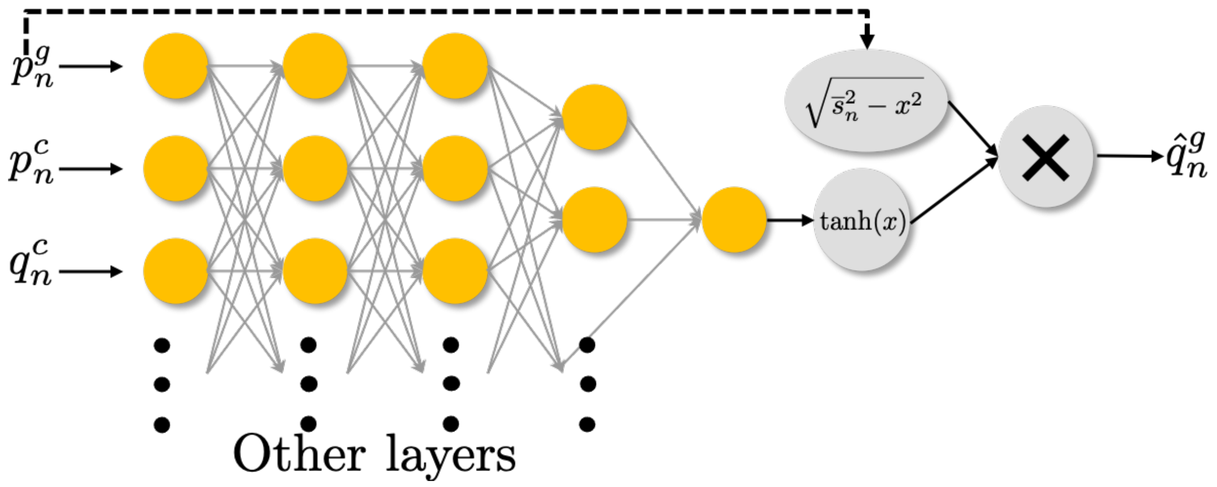


Figure 3.2: The inverter control policy $\mathbf{q} = \boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w}) \in \mathcal{Q}_{\boldsymbol{\theta}}$ has been implemented using a DNN. Vertically stacked nodes represent the nonlinear activation functions of a single layer. Edges across nodes correspond to weights applied linearly on node output to compute the inputs to the next layer. The vector of grid conditions $\boldsymbol{\theta}$ (or its proxy $\boldsymbol{\phi}$) is fed as the input to the DNN. Vector $\mathbf{w}$ collects all weights of the DNN. The DNN output provides the inverter setpoints $\mathbf{q}$. The activation function of the output layer of the DNN has been modified to ensure $\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w}) \in \mathcal{Q}_{\boldsymbol{\theta}}$ for all $\boldsymbol{\phi}$. The output neuron feeds into the tanh() activation function and then scaled by $\sqrt{\bar{s}_n^2 - (p_n^g)^2}$. The scaling operation involves a skip connection from p_n^g, which is one of the inputs to the DNN.

updates henceforth. Note that the gradient $\nabla_{\boldsymbol{\lambda}} L(\mathbf{w}; \boldsymbol{\lambda})$ in the dual variable update in (3.11b) can be substituted as $\nabla_{\boldsymbol{\lambda}} L(\mathbf{w}; \boldsymbol{\lambda}) = \mathbf{g}(\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w}), \boldsymbol{\theta})$.

Following a stochastic approximation approach, the ensemble averages in (3.9) are first surrogated by sample averages computed over a set of S scenarios $\{\boldsymbol{\phi}^s, \boldsymbol{\theta}^s\}_{s=1}^S$. The average ohmic losses for example can be approximated as

$$\mathbb{E}[\ell(\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w}), \boldsymbol{\theta})] \simeq \frac{1}{S} \sum_{s=1}^{S} \ell(\boldsymbol{\pi}(\boldsymbol{\phi}^s; \mathbf{w}), \boldsymbol{\theta}^s). \tag{3.12}$$

Even with the sample approximation in (3.12), computing the gradients needed in (3.11) remains computationally expensive as one needs to compute gradients for each one of the S

training examples. Taking ohmic losses for example, we have that

$$\mathbb{E}[\nabla_{\mathbf{w}}\ell(\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}^k),\boldsymbol{\theta})] \simeq \frac{1}{S}\sum_{s=1}^{S}\nabla_{\mathbf{w}}\ell(\boldsymbol{\pi}(\boldsymbol{\phi}^s;\mathbf{w}^k),\boldsymbol{\theta}^s). \tag{3.13}$$

Notice the two indices in (3.13): index k indexes primal/dual updates, and index s indexes data (scenarios). To perform iteration k, one has to cycle across all scenarios $s = 1, \ldots, S$, and compute the derivatives of losses with respect to the previous update $\mathbf{w}^k$ for each one of the scenarios. Stochastic approximation alleviates this burden by approximating the gradients needed in (3.11) using a *single* scenario per iteration. In other words, gradients are approximated not by (3.13), but using a single datum as

$$\mathbb{E}[\nabla_{\mathbf{w}}\ell(\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}^k),\boldsymbol{\theta})] \simeq \nabla_{\mathbf{w}}\ell(\boldsymbol{\pi}(\boldsymbol{\phi}^s;\mathbf{w}^k),\boldsymbol{\theta}^s). \tag{3.14}$$

Therefore, at iteration k, stochastic approximation selects a scenario s to compute all gradients needed in (3.11). Scenarios can be selected at random or sequentially. Either way, since each iteration k ends up using only a single scenario s, we will henceforth use symbol k to index both iterations and scenarios. This is without loss of generality as $(\boldsymbol{\phi}^k, \boldsymbol{\theta}^k)$ simply means the scenario picked at iteration k.

Given the aforesaid simplification, the gradients in (3.11) can be approximated using a single scenario per update as [22]

$$\mathbf{w}^{k+1} := \mathbf{w}^k - \mu_w\left(\nabla_{\mathbf{w}}\ell^k + \left(\nabla_{\mathbf{w}}\mathbf{g}^k\right)^\top\boldsymbol{\lambda}^k\right) \tag{3.15a}$$

$$\boldsymbol{\lambda}^{k+1} := \left[\boldsymbol{\lambda}^k + \mu_\lambda\mathbf{g}\left(\boldsymbol{\pi}(\boldsymbol{\phi}^k;\mathbf{w}^{k+1}),\boldsymbol{\theta}^k\right)\right]_+ \tag{3.15b}$$

where the shorthand notation $\nabla_{\mathbf{w}}\ell^k$ denotes the gradient of ℓ and $\nabla_{\mathbf{w}}\mathbf{g}^k$ the Jacobian matrix of $\mathbf{g}$, both with respect to $\mathbf{w}$ and both evaluated at $(\boldsymbol{\phi}^k, \mathbf{w}^k, \boldsymbol{\theta}^k)$. The rest of this section explains how the gradients appearing in (3.15a) can be computed for the averaged and probabilistic formulations, while Figure 3.3 summarizes the workflow for the training and testing (operational) phases for both formulations.

3.5.1 Averaged Formulation

For the averaged formulation, the stochastic primal-dual updates can be obtained by replacing $\mathbf{g}(\boldsymbol{\pi}(\boldsymbol{\phi};\mathbf{w}),\boldsymbol{\theta})$ with the constraint functions from (3.3) to get

$$\mathbf{w}^{k+1} := \mathbf{w}^k - \mu_w \left(\nabla_{\mathbf{w}}\ell^k + \left(\nabla_{\mathbf{w}}\mathbf{v}^k\right)^\top (\overline{\boldsymbol{\lambda}}^k - \underline{\boldsymbol{\lambda}}^k)\right) \tag{3.16a}$$

$$\underline{\boldsymbol{\lambda}}^{k+1} := \left[\underline{\boldsymbol{\lambda}}^k + \mu_\lambda \left(\underline{\mathbf{v}} - \mathbf{v}\left(\boldsymbol{\pi}(\boldsymbol{\phi}^k;\mathbf{w}^{k+1}),\boldsymbol{\theta}^k\right)\right)\right]_+ \tag{3.16b}$$

$$\overline{\boldsymbol{\lambda}}^{k+1} := \left[\overline{\boldsymbol{\lambda}}^k + \mu_\lambda \left(\mathbf{v}\left(\boldsymbol{\pi}(\boldsymbol{\phi}^k;\mathbf{w}^{k+1}),\boldsymbol{\theta}^k\right) - \overline{\mathbf{v}}\right)\right]_+ . \tag{3.16c}$$

To compute the Jacobian matrix $\nabla_{\mathbf{w}}\mathbf{v}$ of voltages with respect to DNN weights, we resort

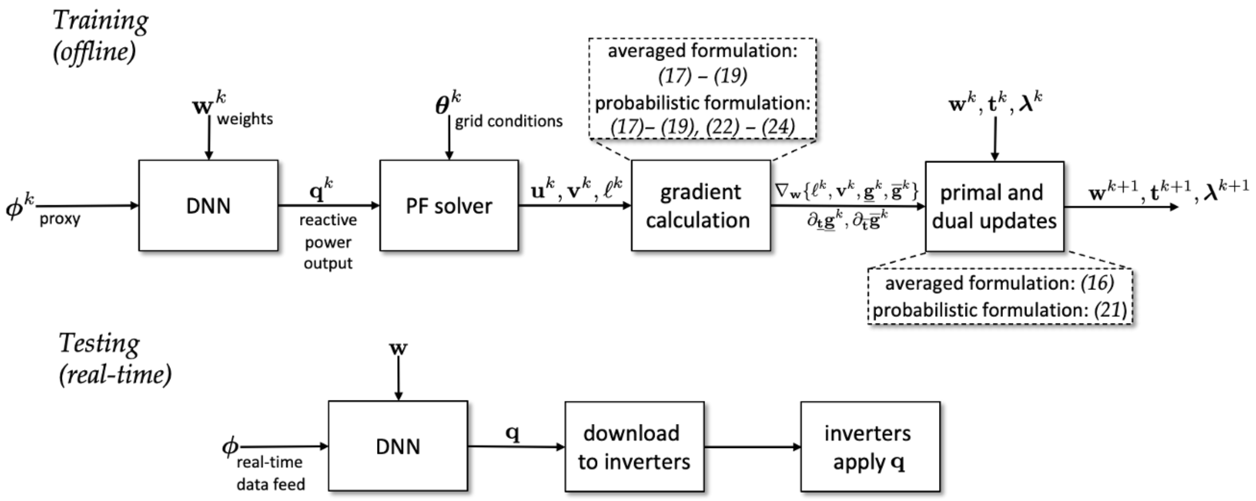

Training (offline)
$\mathbf{w}^k$ weights
$\boldsymbol{\theta}^k$ grid conditions
averaged formulation: (17) – (19)
probabilistic formulation: (17)– (19), (22) – (24)
$\mathbf{w}^k, \mathbf{t}^k, \boldsymbol{\lambda}^k$
$\boldsymbol{\phi}^k$ proxy
DNN
$\mathbf{q}^k$ reactive power output
PF solver
$\mathbf{u}^k, \mathbf{v}^k, \ell^k$
gradient calculation
$\nabla_{\mathbf{w}}\{\ell^k, \mathbf{v}^k, \mathbf{g}^k, \overline{\mathbf{g}}^k\}$ $\partial_t \underline{\mathbf{g}}^k, \partial_t \overline{\mathbf{g}}^k$
primal and dual updates
$\mathbf{w}^{k+1}, \mathbf{t}^{k+1}, \boldsymbol{\lambda}^{k+1}$
averaged formulation: (16)
probabilistic formulation: (21)
Testing (real-time)
$\mathbf{w}$
$\boldsymbol{\phi}$ real-time data feed
DNN
$\mathbf{q}$
download to inverters
inverters apply $\mathbf{q}$

Figure 3.3: Workflow for the training and testing (operation) phases of the proposed DNN-based inverter control strategy.

to the power flow equations. Let vector $\tilde{\mathbf{u}} \in \mathbb{C}^{N+1}$ collect the complex voltage phasors at all buses and define $\bar{\mathbf{u}} := [\mathrm{Re}(\tilde{\mathbf{u}})^\top \ \mathrm{Im}(\tilde{\mathbf{u}})^\top]^\top \in \mathbb{R}^{2N+2}$. Vector $\mathbf{u} \in \mathbb{R}^{2N}$ is obtained by dropping the first and $(N+1)$-th entries of $\bar{\mathbf{u}}$. These two entries correspond to the substation voltage $\tilde{u}_0$, which is assumed constant. The sought Jacobian matrix can be computed via the chain rule as

$$\nabla_{\mathbf{w}}\mathbf{v} = \nabla_{\mathbf{q}}\mathbf{v} \cdot \nabla_{\mathbf{w}}\mathbf{q} = \nabla_{\mathbf{u}}\mathbf{v} \cdot \nabla_{\mathbf{q}}\mathbf{u} \cdot \nabla_{\mathbf{w}}\mathbf{q}. \tag{3.17}$$

We next elaborate on the three Jacobian matrices needed in (3.17). The $N \times 2N$ matrix $\nabla_{\mathbf{u}}\mathbf{v}$ can be readily computed by definition of voltage magnitudes. Its (n,m)-th entry is

$$[\nabla_{\mathbf{u}}\mathbf{v}]_{n,m} = \begin{cases} \frac{u_n}{v_n} & , \ m = n \\ \frac{u_{n+N}}{v_n} & , \ m = n+N \\ 0 & , \ \text{otherwise.} \end{cases}$$

We proceed with finding $\nabla_{\mathbf{q}}\mathbf{u}$. If $\mathbf{p}+j\mathbf{q}$ is the vector of complex power injections at all buses excluding the substation, define $\mathbf{s} := [\mathbf{p}^\top \ \mathbf{q}^\top]^\top$. Per the power flow equations, every entry i of $\mathbf{s}$ can be expressed as a quadratic function of $\bar{\mathbf{u}}$, that is $s_i = \bar{\mathbf{u}}^\top \mathbf{M}_i \bar{\mathbf{u}}$ for a symmetric real-valued matrix $\mathbf{M}_i$ derived from the bus admittance matrix of the feeder; see e.g., [54] for the detailed expressions for $\mathbf{M}_i$'s. Therefore, the i-th row of the Jacobian matrix $\nabla_{\bar{\mathbf{u}}}\mathbf{s}$ is given by $2\bar{\mathbf{u}}^\top \mathbf{M}_i$. By dropping the first and $(N+1)$-th column of $\nabla_{\bar{\mathbf{u}}}\mathbf{s}$, we obtain $\nabla_{\mathbf{u}}\mathbf{s}$. Under mild technical conditions, the inverse function theorem predicates that

$$\nabla_{\mathbf{s}}\mathbf{u} = (\nabla_{\mathbf{u}}\mathbf{s})^{-1}. \tag{3.18}$$

Matrix $\nabla_{\mathbf{q}}\mathbf{u}$ can be clearly obtained by keeping only the last N rows of $\nabla_{\mathbf{s}}\mathbf{u}$. The third matrix $\nabla_{\mathbf{w}}\mathbf{q}$ can be readily computed using gradient back-propagation across the DNN.

The gradient vector of ohmic losses with respect to the DNN weights can be computed similarly as

$$(\nabla_{\mathbf{w}}\ell)^\top = (\nabla_{\mathbf{u}}\ell)^\top \cdot \nabla_{\mathbf{q}}\mathbf{u} \cdot \nabla_{\mathbf{w}}\mathbf{q}. \tag{3.19}$$

The gradient $\nabla_{\mathbf{u}}\ell$ can be easily computed by recognizing

$$\ell = \sum_{n=0}^{N} p_n = \bar{\mathbf{u}}^\top \left(\sum_{n=0}^{N} \mathbf{M}_i\right) \bar{\mathbf{u}}$$

where matrix $\mathbf{M}_0$ describes the power injection at the substation as $p_0 = \bar{\mathbf{u}}^\top \mathbf{M}_0 \bar{\mathbf{u}}$ similar to the remaining injections. It is then obvious that $\nabla_{\bar{\mathbf{u}}}\ell = 2\sum_{n=0}^{N} \mathbf{M}_i \bar{\mathbf{u}}$. The gradient $\nabla_{\mathbf{u}}\ell$ of (3.19) is found by dropping the first and $(N+1)$-th entries of vector $\nabla_{\bar{\mathbf{u}}}\ell$.

3.5.2 Probabilistic Formulation

For the probabilistic formulation of (3.5), the update steps in (3.15) cannot be applied directly. This is because constraints (3.5b)–(3.5c) involve the indicator function that is not differentiable, and so the Jacobian matrix $\nabla_{\mathbf{w}}\mathbf{g}$ does not exist. Moreover, these constraints are non-convex, thus prohibiting stochastic (sub)gradient updates. To circumvent these complications, we instead turn to the convex CVaR approximations to (3.5) using stochastic subgradient primal-dual updates.

We first briefly review the CVaR approximation of chance constraints from [73], and then compute the related subgradients. For some $\alpha \in (0,1)$, consider the chance constraint $\Pr[f_\theta(x) \geq 0] = \mathbb{E}_\theta[\mathbb{1}(f_\theta(x))] \leq \alpha$, where function f depends on the optimization variable x and a random variable θ. Note that $\mathbb{1}(f_\theta(x)) \leq [1 + f_\theta(x)/t]_+$ for all $f_\theta(x)$ and $t > 0$. This is easy to verify by checking the two cases in the definition of the indicator function in (3.6). Then, if there exists a $t > 0$ satisfying $\mathbb{E}_\theta\left[[1 + f_\theta(x)/t]_+\right] \leq \alpha$, the original chance constraint $\mathbb{E}_\theta[\mathbb{1}(f_\theta(x))] \leq \alpha$ holds too. Since $t > 0$, the restriction of the chance constraint can be alternatively expressed as $\mathbb{E}_\theta\left[[t+f_\theta(x)]_+\right] \leq \alpha t$. In fact, the requirement $t>0$ can be dropped because for all negative t, the constraint $\mathbb{E}_\theta[t+f_\theta(x)]_+ \leq \alpha t$ becomes infeasible. And for $t=0$, the restricted constraint yields $\mathbb{E}_\theta\left[[f_\theta(x)]_+\right] \leq 0$ or equivalently $f_\theta(x)<0$ for all θ, so that the original chance constraint holds trivially. Therefore, imposing the convex constraint $\mathbb{E}_\theta\left[[t+f_\theta(x)]_+\right] \leq \alpha t$ for some t constitutes a restriction of the original chance constraint $\Pr[f_\theta(x) \geq 0] \leq \alpha$.

By identifying $f_\theta(x)$ with $\underline{v}_n - v_n(\mathbf{q},\boldsymbol{\theta})$ and introducing an auxiliary variable $\underline{t}_n$ per bus n, we can now restrict the voltage chance constraint in (3.5b) by imposing constraint

$$\mathbb{E}\left[[\underline{t}_n + \underline{v}_n - v_n(\mathbf{q},\boldsymbol{\theta})]_+\right] \leq \alpha \underline{t}_n, \ \forall n \in \mathcal{N}.$$

The chance constraints of (3.5c) on upper voltage limits can be treated similarly using variables $\overline{t}_n$'s. Collecting auxiliary variables $\{\underline{t}_n, \overline{t}_n\}_{n=1}^N$ in vectors $\underline{\mathbf{t}}$ and $\overline{\mathbf{t}}$ accordingly, we can now formulate the convex restriction of (3.5) as

$$\min_{\mathbf{q}\in\mathcal{Q}_{\boldsymbol{\theta}},\underline{\mathbf{t}},\overline{\mathbf{t}}} \quad \mathbb{E}[\ell(\mathbf{q},\boldsymbol{\theta})] \tag{3.20a}$$

$$\text{s.to} \quad \mathbb{E}\left[[\underline{\mathbf{t}} + \underline{\mathbf{v}} - \mathbf{v}(\mathbf{q},\boldsymbol{\theta})]_+ - \alpha\underline{\mathbf{t}}\right] \leq \mathbf{0} \tag{3.20b}$$

$$\mathbb{E}\left[[\overline{\mathbf{t}} + \mathbf{v}(\mathbf{q},\boldsymbol{\theta}) - \overline{\mathbf{v}}]_+ - \alpha\overline{\mathbf{t}}\right] \leq \mathbf{0}. \tag{3.20c}$$

For brevity, let the expressions inside the expectation operator of (3.20b)–(3.20c) be represented by $\underline{\mathbf{g}}\left(\underline{\mathbf{t}}, \mathbf{v}(\mathbf{q},\boldsymbol{\theta})\right)$ and $\overline{\mathbf{g}}\left(\overline{\mathbf{t}}, \mathbf{v}(\mathbf{q},\boldsymbol{\theta})\right)$. Similar to (3.8), problem (3.20) can be tackled using stochastic primal-dual updates upon replacing the ensemble with sample averages over K scenarios. Different from the averaged formulation however, the constraint functions $\underline{\mathbf{g}}\left(\underline{\mathbf{t}}, \mathbf{v}(\mathbf{q},\boldsymbol{\theta})\right)$ and $\overline{\mathbf{g}}\left(\overline{\mathbf{t}}, \mathbf{v}(\mathbf{q},\boldsymbol{\theta})\right)$ are non-differentiable with respect to $(\mathbf{v}, \underline{\mathbf{t}}, \overline{\mathbf{t}})$. We use their subgradients instead.

$$\mathbf{w}^{k+1} := \mathbf{w}^k - \mu_w \left(\nabla_{\mathbf{w}} \ell^k + \left(\partial_{\mathbf{w}} \underline{\mathbf{g}}^k\right)^\top \underline{\boldsymbol{\lambda}}^k + \left(\partial_{\mathbf{w}} \overline{\mathbf{g}}^k\right)^\top \overline{\boldsymbol{\lambda}}^k \right) \tag{3.21a}$$

$$\underline{\mathbf{t}}^{k+1} := \underline{\mathbf{t}}^k - \mu_t \left(\partial_{\underline{\mathbf{t}}} \underline{\mathbf{g}}^k\right)^\top \underline{\boldsymbol{\lambda}}^k \tag{3.21b}$$

$$\overline{\mathbf{t}}^{k+1} := \overline{\mathbf{t}}^k - \mu_t \left(\partial_{\overline{\mathbf{t}}} \overline{\mathbf{g}}^k\right)^\top \overline{\boldsymbol{\lambda}}^k \tag{3.21c}$$

$$\underline{\boldsymbol{\lambda}}^{k+1} := \left[\underline{\boldsymbol{\lambda}}^k + \mu_\lambda \underline{\mathbf{g}}\left(\underline{\mathbf{t}}^{k+1}, \mathbf{v}\left(\mathbf{q}^{k+1}, \boldsymbol{\theta}^k\right)\right)\right]_+ \tag{3.21d}$$

$$\overline{\boldsymbol{\lambda}}^{k+1} := \left[\overline{\boldsymbol{\lambda}}^k + \mu_\lambda \overline{\mathbf{g}}\left(\overline{\mathbf{t}}^{k+1}, \mathbf{v}\left(\mathbf{q}^{k+1}, \boldsymbol{\theta}^k\right)\right)\right]_+ . \tag{3.21e}$$

To compute the needed subgradients, recall that a subgradient of $f(x) = [x]_+$ can be found as

$$\partial f(x) = \begin{cases} 0 & , \ x < 0 \\ 1 & , \ x \geq 0 \end{cases} = \mathbb{1}(x).$$

The subgradients involved in (3.21b)–(3.21c) can be computed using the chain rule as

$$\partial_{\underline{\mathbf{t}}} \underline{\mathbf{g}} = \mathrm{dg}\left(\mathbb{1}\left(\underline{\mathbf{t}} + \underline{\mathbf{v}} - \mathbf{v}\right)\right) - \alpha \mathbf{I}_N \tag{3.22a}$$

$$\partial_{\overline{\mathbf{t}}} \overline{\mathbf{g}} = \mathrm{dg}\left(\mathbb{1}\left(\overline{\mathbf{t}} + \mathbf{v} - \overline{\mathbf{v}}\right)\right) - \alpha \mathbf{I}_N \tag{3.22b}$$

where the indicator functions here are applied entrywise and evaluate to vectors. In turn, the subgradients appearing in (3.21a) can be found as

$$\partial_{\mathbf{w}} \underline{\mathbf{g}} = \partial_{\mathbf{v}} \underline{\mathbf{g}} \cdot \nabla_{\mathbf{w}} \mathbf{v} \tag{3.23a}$$

$$\partial_{\mathbf{w}} \overline{\mathbf{g}} = \partial_{\mathbf{v}} \overline{\mathbf{g}} \cdot \nabla_{\mathbf{w}} \mathbf{v}. \tag{3.23b}$$

The Jacobian $\nabla_{\mathbf{w}} \mathbf{v}$ has already been computed in (3.17), while the subgradients with respect to voltage magnitudes are

$$\partial_{\mathbf{v}}\underline{\mathbf{g}} = -\,\mathrm{dg}\left(\mathbb{1}\left(\underline{\mathbf{t}} + \underline{\mathbf{v}} - \mathbf{v}\right)\right) \tag{3.24a}$$

$$\partial_{\mathbf{v}}\overline{\mathbf{g}} = \mathrm{dg}\left(\mathbb{1}\left(\overline{\mathbf{t}} + \mathbf{v} - \overline{\mathbf{v}}\right)\right). \tag{3.24b}$$

Remark 3.1. During the training phase of the DNN, one may encounter ill-conditioned samples $\{\boldsymbol{\phi}^k, \boldsymbol{\theta}^k\}$ that, along with the DNN output $\mathbf{q}^k$, cause the power flow solver to diverge. One can reduce the number of such ill-conditioned samples from the training data set by sampling close to the nominal benchmark values. Nonetheless, if such ill-conditioned samples do occur, a straightforward *recourse* functionality could be adopted. First, the sample $\boldsymbol{\phi}^k$ is fed into the DNN to obtain $\mathbf{q}^k$. Then, the PF solver is called for $\boldsymbol{\theta}^k$ and $\mathbf{q}^k$. If the PF solver converges within a prescribed number of maximum iterations, we move on with the gradient calculations. Otherwise, we draw a new sample $\{\boldsymbol{\phi}^k, \boldsymbol{\theta}^k\}$ and repeat the process.

3.6 Gradient-Free Implementation

As discussed earlier, the primal-dual updates of Section 3.5 require computing the (sub)gradients of losses and voltages with respect to inverter reactive power injections. This section puts forth a gradient-free implementation of the DNN updates relying on a *digital twin* of the feeder. This implementation does not require computing gradients, but only evaluating losses and voltages. The digital twin can be a power flow solver (GridLAB-D or OpenDSS), or a hardware emulator of the feeder. Once fed with all power injections (i.e., grid conditions $\boldsymbol{\theta}$ and inverter setpoints $\mathbf{q}$), the digital twin returns the vector of nodal voltages $\mathbf{v}$ and ohmic losses ℓ.

Such gradient-free approach can be practically relevant under two settings. First, when one does not want to deal with gradients as calculating them is not straight-forward and can be prone to errors. Second, when the feeder model is complicated and computing gradients is quite complex. That could be the case in unbalanced multiphase grids; if grid devices (regulators, capacitors, transformer banks) are to be included; and/or when loads are represented by detailed ZIP or exponential models. In such cases, gradient calculations can be cumbersome. On other hand, reliable power flow modules do exist and can be used to evaluate the needed functions under either settings.

To arrive at a gradient-free implementation, we cannot compute the partial derivatives appearing in the left-hand side of (3.17) and (3.19). Nonetheless, we can aim directly for the Jacobian matrix $\nabla_{\mathbf{q}}\mathbf{v}$ and the gradient vector $\nabla_{\mathbf{q}}\ell$, and approximate them through finite differences. In detail, we resort to zeroth-order approximants (see [74]) of the needed sensitivities by querying the digital twin twice to obtain two function evaluations as:

$$\hat{\nabla}_{\mathbf{q}}\ell = \frac{\ell(\mathbf{q}+\epsilon\check{\mathbf{q}},\boldsymbol{\theta}) - \ell(\mathbf{q}-\epsilon\check{\mathbf{q}},\boldsymbol{\theta})}{2\epsilon}\check{\mathbf{q}}^\top \tag{3.25a}$$

$$\hat{\nabla}_{\mathbf{q}}\mathbf{v} = \frac{\mathbf{v}(\mathbf{q}+\epsilon\check{\mathbf{q}},\boldsymbol{\theta}) - \mathbf{v}(\mathbf{q}-\epsilon\check{\mathbf{q}},,\boldsymbol{\theta})}{2\epsilon}\check{\mathbf{q}}^\top \tag{3.25b}$$

where ϵ is the scale of perturbation, and $\check{\mathbf{q}}$ is a perturbation vector Gaussian distributed with zero-mean and standard deviation $\sigma_{\check{\mathbf{q}}}$. The quantities ϵ and $\sigma_{\check{\mathbf{q}}}$ are treated as hyper-parameters and are set during the training process.. The approximations in (3.25) are carried out in three steps. First, the DNN is presented with the input $\boldsymbol{\phi}$ and its output $\mathbf{q}$ is recorded. Second, the digital twin is presented with $(\mathbf{q},\boldsymbol{\theta})$ and computes $\ell(\mathbf{q},\boldsymbol{\theta})$ and $\mathbf{v}(\mathbf{q},\boldsymbol{\theta})$. Third, the digital twin is presented with $(\mathbf{q}+\epsilon\check{\mathbf{q}},\boldsymbol{\theta})$ and computes $\mathbf{v}(\mathbf{q}+\epsilon\check{\mathbf{q}},\boldsymbol{\theta})$. Fig 3.4 compares the steps for obtaining the quantities $\nabla_{\mathbf{w}}\ell$ and $\nabla_{\mathbf{w}}\mathbf{v}$ for the *gradient-based* and *gradient-free* approaches.

With the finite-difference approximants of (3.25) in place, the *gradient-free* primal-dual updates are straightforward to execute by approximating

$$(\hat{\nabla}_{\mathbf{w}}\ell)^\top = (\hat{\nabla}_{\mathbf{q}}\ell)^\top \nabla_{\mathbf{w}}\mathbf{q}$$
$$\hat{\nabla}_{\mathbf{w}}\mathbf{v} = \hat{\nabla}_{\mathbf{q}}\mathbf{v} \cdot \nabla_{\mathbf{w}}\mathbf{q}$$

where $\nabla_{\mathbf{w}}\mathbf{q}$ is again calculated gradient back-propagation.

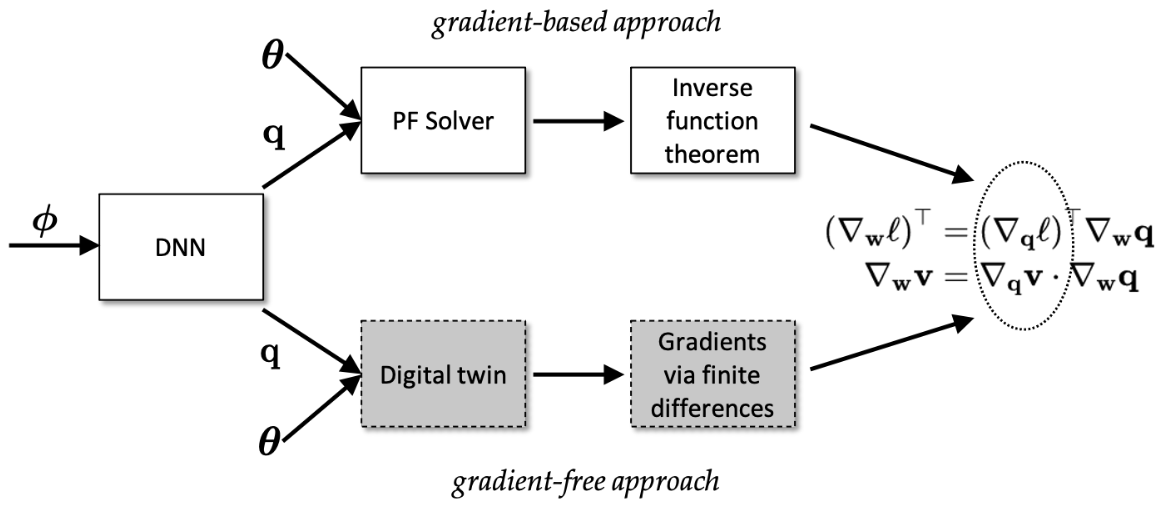

gradient-based approach
θ
$\mathbf{q}$
PF Solver
Inverse function theorem
ϕ
DNN
$(\nabla_{\mathbf{w}}\ell)^\top = (\nabla_{\mathbf{q}}\ell)^\top \nabla_{\mathbf{w}}\mathbf{q}$
$\nabla_{\mathbf{w}}\mathbf{v} = \nabla_{\mathbf{q}}\mathbf{v} \cdot \nabla_{\mathbf{w}}\mathbf{q}$
$\mathbf{q}$
Digital twin
Gradients via finite differences
θ
gradient-free approach

Figure 3.4: Steps for calculating $\nabla_{\mathbf{w}}\ell$ and $\nabla_{\mathbf{w}}\mathbf{v}$ for the *gradient-based* (top) and *gradient-free* (bottom) approaches. The *gradient-based* approach requires a solution to the power flow equations in conjunction with the inverse function theorem (Section 3.5); the *gradient-free* obtains the desired quantities by probing the digital twin and applying zero-order approximations.

3.7 Numerical Tests

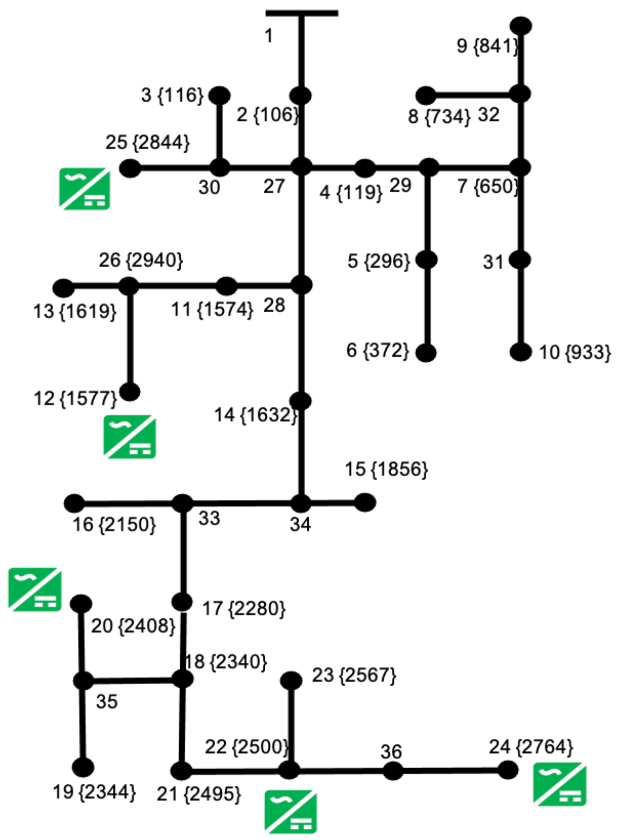


Figure 3.5: The IEEE 37-bus feeder used for the numerical tests. Node numbering follows the format `node number {panel ID}`. The inverters at nodes $\{12, 20, 22, 24, 25\}$ provide reactive power control, whereas the rest operate at unit power factor.

The performance of the proposed DNN-based control strategy was evaluated using a single-phase version of the IEEE 37-bus feeder. Real-world one-minute active load and solar generation data were extracted for April 2, 2011 from the Smart* project [9], [14]. For active loads, homes with IDs 20-369 were used. Averaged load demands were calculated by considering 10 homes at a time, and were serially allotted to buses 2-36 of Fig. 3.5. The values of active loads were scaled so their maximum active load per node matched its benchmark value. Reactive loads were then added to each of these homes by sampling lagging power factors uniformly within $[0.9, 1.0]$ and for each time interval. For solar generation, panel IDs were matched to the buses as shown in Fig. 3.5. Solar generation values were scaled so the maximum generation per panel was 2 times the benchmark value. Out of all the nodes with inverter-interfaced solar generation, those at nodes $\{12, 20, 22, 24, 25\}$ were also providing reactive power support. The extracted data points were considered as available forecasts for 4-hour control periods. Appropriate training and testing scenarios were created. Zero-mean white Gaussian noise was added to the 240 one-minute data points from the forecast to create a total of 1200 samples. The standard deviation of the Gaussian noise was set to 0.1 times the mean load forecast. Out of the 1200 samples, 960 were used as the training set and the remaining 240 formed the testing set. The training samples were additionally randomly shuffled to promote better generalization for the DNNs.

All tests were conducted on a 2.4 GHz 8-Core Intel Core i9 processor laptop computer with 64 GB RAM. Simulation scripts were written in Python and TensorFlow libraries to implement and train the DNNs. For the tests presented, four-layered fully connected DNNs were employed. The grid conditions vector $\boldsymbol{\theta} := [\mathbf{p}^g, \mathbf{p}^c, \mathbf{q}^c]^\top$ consisting of measurements from the $M \leq N$ buses equipped with smart meters were fed as inputs to the DNNs. Therefore, the input layers were chosen to have $3M$ neurons. The two subsequent hidden layers were fixed to having $3N$ and $2N$ neurons, respectively. Finally, the output layers had 5 neurons corresponding to the 5 inverters. All but the final layers of the DNNs employed the ReLU (rectified linear unit) activation with the final layers using a scaled *tanh* activation to ensure the inverter limits $\mathbf{q}^g \in \mathcal{Q}^t$. The weights for the DNN layers were initiated from a Gaussian distribution with zero mean and a unit standard deviation. The biases for the DNN layers, the dual variables, and the auxiliary variables were all initialized at zero. Additional modelling and training details are presented along with the discussions of the results as follows.

3.7.1 Averaged Formulation

For the average formulation, the DNN was fed with the complete vector of grid conditions $\boldsymbol{\theta}$ obtained from measurements collected at all buses. DNN weights were updated using the DNN optimization algorithm Adam with a learning rate of 0.001. Dual variables were updated using SGD with a learning rate of 10 that decayed with the square-root of the iteration index [61]. The model was then trained for 15 epochs over the training scenarios.

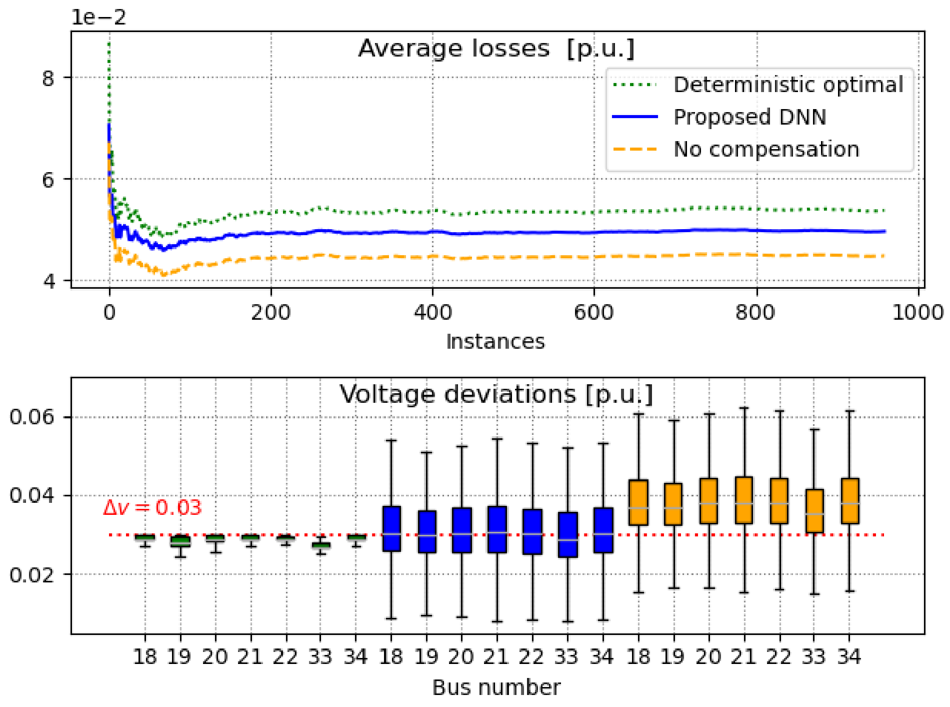


Figure 3.6: *Top:* Time-averaged losses during the 12–4 pm training interval attained by the deterministic optimal control strategy of (3.2); the proposed DNN-based inverter control; and no reactive power compensation by inverters. *Bottom:* Box plots showing the first and third quantiles of the voltage deviations experienced across buses under the three control strategies. Due to high solar generation, the feeder experiences lower ohmic losses at the expense of severe over-voltages if there is no reactive power control by inverters. The deterministic optimal inverter control strategy regulates voltages by absorbing reactive power, which increases line currents and consequently losses. The proposed strategy achieves lower average losses over deterministic optimal inverter control as voltages are not constrained within ±3% at all times.

To demonstrate the efficacy of the proposed approach, the results are compared against a no-compensation scenario, i.e., the scenario where all inverters operate at unit power factor and provide no reactive power support. The DNN-based approach is also benchmarked against an deterministic optimal approach that solves the problem in (3.2) per minute. As discussed previously, such deterministic optimal approach might not be realistic to implement in real time due to the high computational burden. Fig. 3.6 compares the average losses and bus voltages under the three scenarios over the training set and during the high solar period of 12–4 pm. Without any reactive power compensation, buses $\{18, 19, 20, 21, 22, 33, 34\}$ experience over-voltages. The proposed DNN-based approach behaves as expected by lowering the average voltages at these buses down to the acceptable range. The deterministic optimal approach also achieves the same objective but by bringing all instantaneous voltages to the desired range whenever feasible. Note that both the DNN-based approach and the deterministic optimal approach incur higher losses when compared to the no compensation scenario. This is a result of increase in the magnitude of line currents on account of reactive power withdrawals. Since the deterministic optimal approach focuses on instantaneous voltage values rather than their averages, it incurs higher losses when compared to the DNN-based approach. The trained DNN was then evaluated over unseen scenarios of the testing set. As can be seen in Fig. 3.7, the proposed approach performed remarkably well in maintaining voltages within limits and lowering average losses over the testing set.

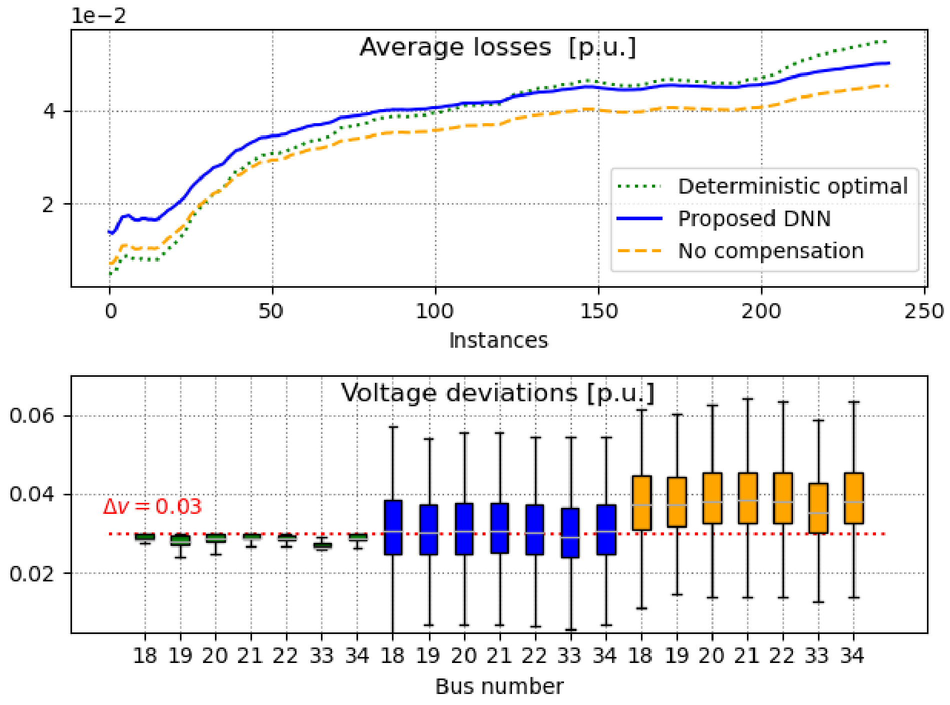


Figure 3.7: Results for averaged formulation over testing data during the interval 12–4 pm: Average losses under the deterministic optimal strategy, the proposed DNN-based approach; and no reactive power compensation are depicted on the top panel. Voltage deviations across buses under the three strategies are shown at the bottom panel.

To highlight the computational advantage of the DNN-based approach during operation, we compared it against the deterministic optimal strategy. The deterministic optimal strategy entails solving as many OPFs as the number of realization of $\boldsymbol{\theta}$'s, and was simulated by formulating a second-order conic program (SOCP) that models a convex relaxation of the original OPF [63]. The SOCP was solved for the 240 samples of the testing dataset for $12-4$ pm in MATLAB using the SeDuMi solver. The simulation was found to take 171.24 s to complete. On the other hand, the DNN-based approach, which requires only a forward pass through the DNN for each realization of $\boldsymbol{\phi}$, took only 0.85 s to compute in Python. This indicates a significant computational speedup.

Finally, the sample probabilities for voltage violations were calculated for the resulting voltages from the no compensation case and the proposed DNN-based approaches. For the buses with non-zero values for these probabilities, radar plots were drawn as shown in Fig. 3.8. The radial-axis represents the values for sample probabilities for voltage limit violations, whereas the angular markings correspond to the bus numbers. One can see that without any reactive power support, the grid faces a high probability of voltage violations over both training and testing. While the average formulation successfully regulates the average voltages, its effect on reducing the total number of occurrences of voltage violations is somewhat modest with buses $\{19, 20, 21, 22, 23\}$ violating the voltage limits for more than half of the scenarios.

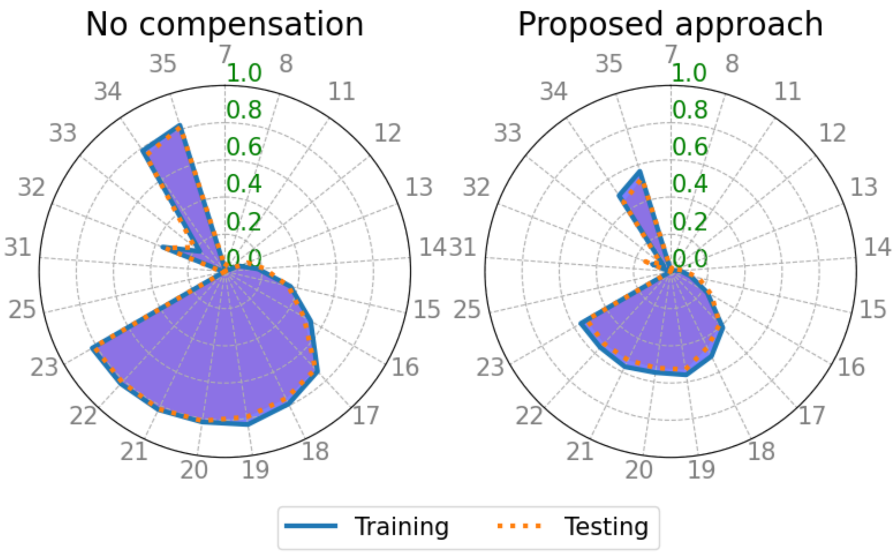


Figure 3.8: Voltage profiles during the 12–4 pm interval for averaged formulation. Results under no reactive power compensation and under the proposed DNN-based policies have been compared. Angular markings correspond to bus numbers, whereas radial markings are sampled probabilities of voltage violations.

3.7.2 Probabilistic Formulation

Employing the DNN architecture from the previous subsection and the full input vector $\boldsymbol{\theta}$, updates in (3.21) were applied to train the DNN for the probabilistic formulation. The DNN weights were updated using Adam with a learning rate of 0.001. For the auxiliary variables

$\{\underline{\mathbf{t}}, \overline{\mathbf{t}}\}$, Adam optimizer with a learning rate of 0.001 was deployed, and the dual variables were updated using SGD with a learning rate of 1 that decayed with the square-root of the iteration index. The model required a higher number of 20 epochs to converge during training because of the additional primal variables $\mathbf{t}$. The experiments were conducted for the same time period of 12–4 pm. The experiments were repeated for three different values of $\alpha = \{0.7, 0.5, 0.3\}$ and the radar plots for the resulting sample probabilities of voltage violations are shown in Fig. 3.9.

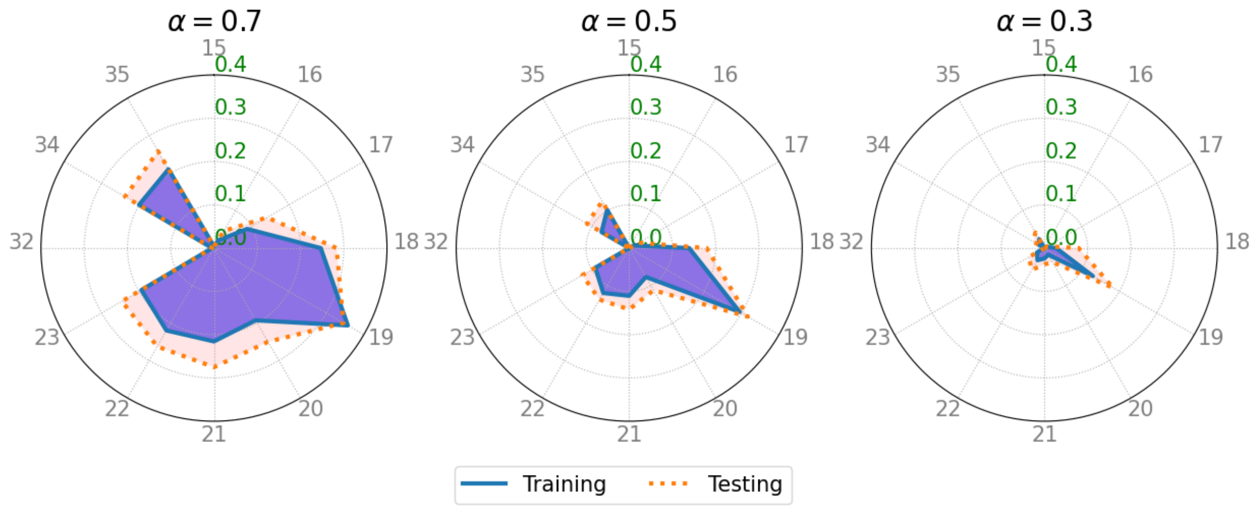


Figure 3.9: Results for probabilistic formulations for the 12–4 pm window. Voltage profiles for different values of $\alpha = \{0.7, 0.5, 0.3\}$ are depicted. Angular markings correspond to bus numbers whereas radial markings are sampled probabilities of voltage limits violation.

As desired, when compared to the averaged formulation, the occurrences of voltage violations under the probabilistic formulation were found to be drastically less for lower values of α. Since the calculated sample probabilities came out to be less than the selected α, the results in Fig. 3.9 confirm the conservative nature (being a convex restriction) of (3.20).

3.7.3 Gradient-Free Implementation

To quantify the accuracy of the gradient approximations in (3.25), the DNNs for the averaged and probabilistic formulations were trained under the gradient-free fashion. The scale of perturbation ϵ was set to 0.1, and the perturbations $\check{\mathbf{q}}$ were sampled from a zero-mean Gaussian distribution with a unit standard deviation. The gradient-free approaches were compared to their gradient-based counterparts over the same time periods and the results shown in Figs. 3.10-3.11.

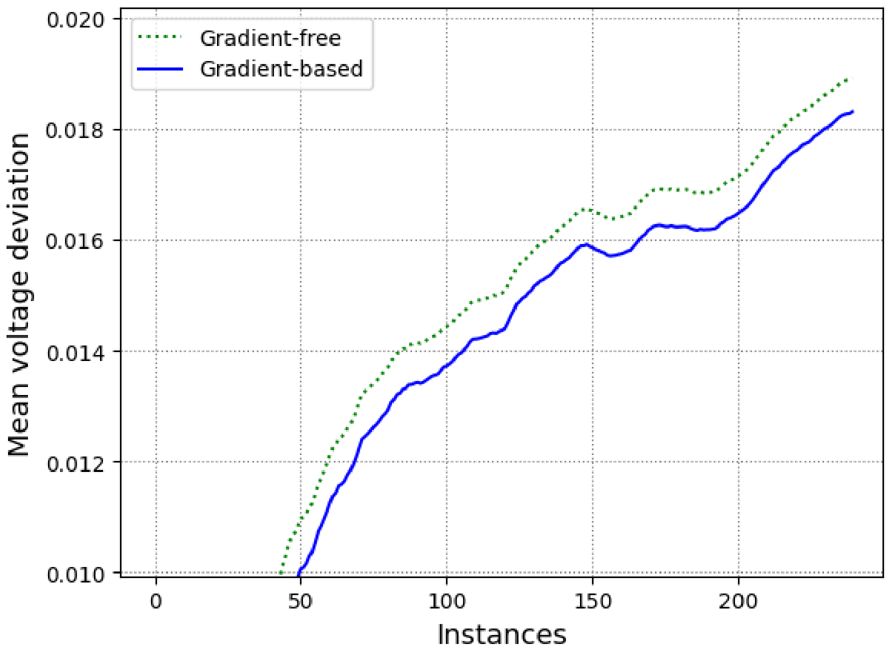


Figure 3.10: Mean voltage deviations across the buses under gradient-based and gradient-free approaches for the averaged formulation and during the testing 12–4 pm interval.

Fig. 3.10 shows the mean voltage deviations across all the buses and time under the two approaches. The gradient-free approach incurs slightly higher voltage violations, but otherwise matches the performance of the gradient-based approach surprisingly well without any explicit knowledge of the underlying relationships. Similar results were confirmed for the probabilistic formulation in Fig. 3.11, where both approaches yield similar sample probabilities for voltage violations over training and testing, with $\alpha = 0.5$.

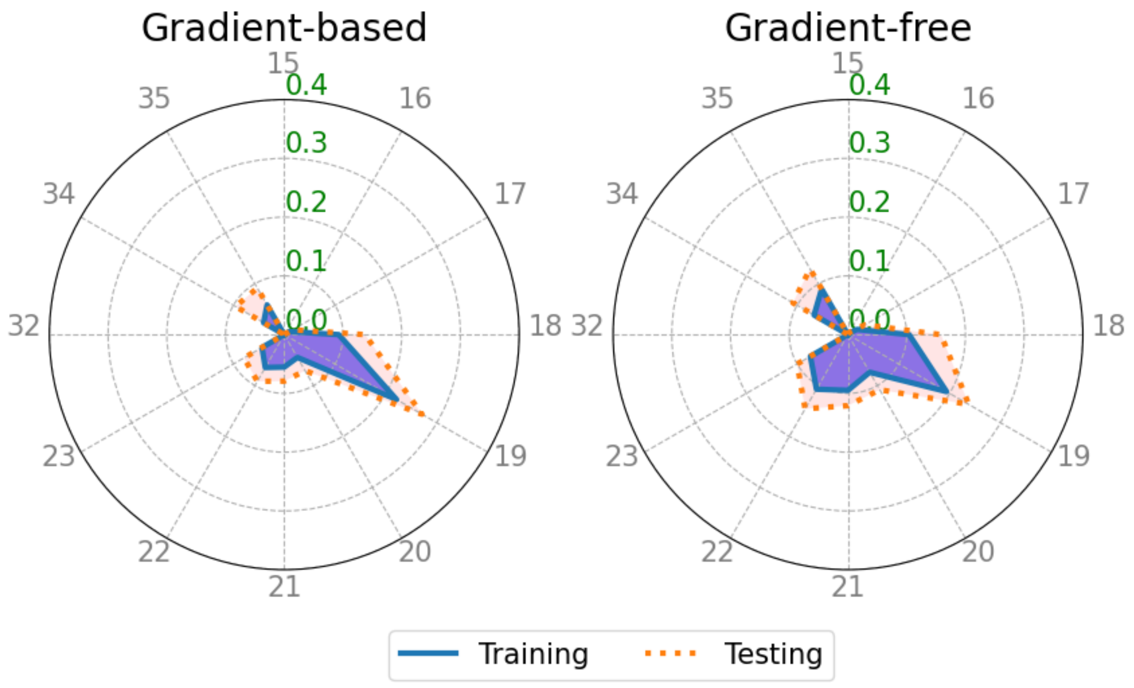


Figure 3.11: Comparison of gradient-based and gradient-free approaches for the probabilistic formulation with $\alpha = 0.5$ over the testing 12–4 pm window. Voltage profiles for both approaches are depicted using polar plots.

3.7.4 Partial Inputs

To study the effect of partial DNN inputs on the control performance of the learned DNN policy, we varied the number of nodes whose data are telemetered in real time for the probabilistic formulation. First, real-time meters were assigned to all nodes with solar generation. Then, different input scenarios were simulated by expanding the subset of inverter-equipped nodes with real-time metering moving from nodes 2–11; nodes 2–16; nodes 2–21; and a full input vector consisting of all nodes. The mean-sampled probabilities of voltage violations across time and buses were recorded. The value of α was set to 0.5 for all scenarios. Figures 3.12 and 3.13 show the results recorded respectively during training and testing.

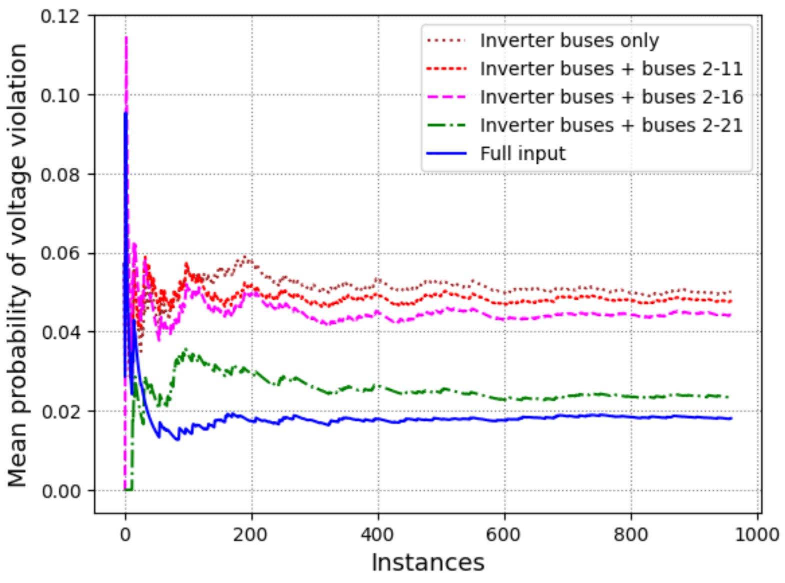


Figure 3.12: Mean sampled probabilities of voltage violations for the probabilistic formulation with $\alpha = 0.5$ during training over 12–4 pm. The dimension of the DNN input increases as the number of nodes monitored in real time increases.

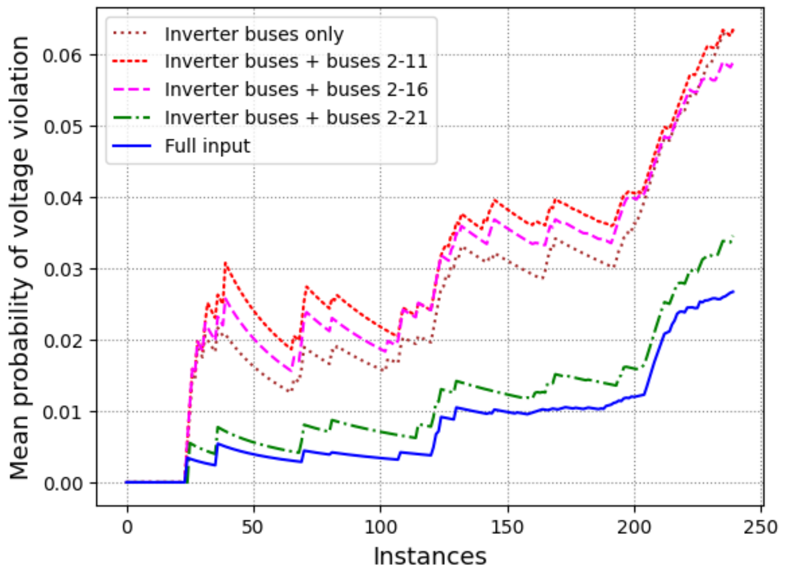


Figure 3.13: Mean sampled probabilities of voltage violations for the probabilistic formulation with $\alpha = 0.5$ during testing over 12–4 pm. The dimension of the DNN input increases as the number of nodes monitored in real time increases.

The results confirm the intuition that the performance of the control policies improves as more real-time information is provided to the DNN. As the nodes with real-time monitoring increase, the sampled probabilities of voltage violations decrease.

3.8 Conclusions

This work has presented a DNN-based approach for stochastic optimal inverter control. To capture uncertainty, the grid conditions have been modeled as random variables and the associated inverter setpoints by a stochastic policy learned through a DNN. The DNN is periodically trained offline to minimize the average ohmic losses and maintain either the average voltages within limits or the probability of voltage deviation occurrences low. Training is accomplished by adopting existing stochastic primal-dual updates and their gradient-free counterparts to the AC OPF setup. The proposed scheme not only expedites the computation of near-optimal inverter setpoints, but also resolves two practical difficulties: *d1)* How to solve an OPF using only a power flow solver or a digital twin of the feeder? and *d2)* How to deal with an OPF if grid conditions are only partially known?

Numerical tests on the benchmark IEEE 37-bus feeder showcase the salient features of the novel methodology. The adopted stochastic primal-dual updates train a DNN-based policy using a modest number of training samples. The DNN is numerically shown to produce an inverter dispatch that minimizes the OPF cost while satisfying the operating constraints. Although the averaged formulation succeeds in maintaining the voltage at each node within limits on the average, violations do occur frequently. To that end, the chance-constrained formulation may be more relevant. The latter comes at minimal extra complexity over the averaged formulation. Being a convex restriction the numerical tests corroborate that it is a conservative yet safe scheme. Our experiments have also shown how the DNN-based policy can be driven with incomplete grid conditions, and demonstrated the improvement in feasibility when more information is presented to the DNN. A final interesting outcome is that when the DNNs are trained using the gradient-free updates, the degradation in performance is minimal although the learner has less information about the feeder at its disposal.

Some open questions are to: *i)* Extend the implementation to a multiphase grid model with detailed ZIP loads, regulators, and capacitor banks; *ii)* Consider additional network constraints and/or alternative objectives; *iii)* Experiment with the frequency of re-training the DNN, the size of the training dataset, and the way it has been generated; *iv)* Utilize the DNN output only to warm-start an actual OPF solver; and *v)* Optimally select the grid proxies to improve inverter control performance under a communication budget.

Chapter 4

DNN-based Policies for Stochastic AC OPF

4.1 Publication Details

S. Gupta, S. Misra, D. Deka, and V. Kekatos, . “DNN-based Policies for Stochastic AC OPF”, *Power Systems Computation Conference, Porto, Portugal, Jun. 2022*, (also published in the Elsevier Journal of Electric Power Systems Research)

4.2 Abstract

A prominent challenge to the safe and optimal operation of the modern power grid arises due to growing uncertainties in loads and renewables. Stochastic optimal power flow (SOPF) formulations provide a mechanism to handle these uncertainties by computing dispatch decisions and control policies that maintain feasibility under uncertainty. Most SOPF formulations consider simple control policies such as affine policies that are mathematically simple and resemble many policies used in current practice. Motivated by the efficacy of machine learning (ML) algorithms and the potential benefits of general control policies for cost and constraint enforcement, we put forth a deep neural network (DNN)-based policy that pre-

dicts the generator dispatch decisions in real time in response to uncertainty. The weights of the DNN are learnt using stochastic primal-dual updates that solve the SOPF without the need for prior generation of training labels and can explicitly account for the feasibility constraints in the SOPF. The advantages of the DNN policy over simpler policies and their efficacy in enforcing safety limits and producing near optimal solutions are demonstrated in the context of a chance constrained formulation on a number of test cases.

4.3 Introduction

In current intra-day power systems operations, the optimal power flow (OPF) is solved at a time scale of 5-15 minutes using load forecasts to obtain economic generation dispatch, whereas affine/linear generation control policies are used within each OPF time period to account for real-time fluctuations in generation and demand. With growing levels of uncertainty, the design of the real-time control policies have become increasingly important for secure and economic grid operation. Such control design is the primary focus of this paper.

Motivated by the need for better uncertainty management, the stochastic AC-OPF (SOPF) [10, 80] problem that accounts for uncertainty, has received significant attention from the research community. Being a stochastic extension of the non-convex AC-OPF, the stochastic AC-OPF is highly computationally challenging. Standard stochastic optimization methods such as stochastic programming using sample average approximation, and scenario-based approaches quickly grow in size rendering them intractable. Hence, much effort has been devoted towards obtaining tractable formulations and designing efficient algorithms for obtaining fast and reliable solutions to this problem. The design of control polices however is relatively under-studied and most SOPF formulations consider *affine* functions to represent control policies. These linearly parameterized policies are algorithmically easier to handle, and serve as reasonable mathematical models for the automatic generation control and local voltage control employed in current practice. However, affine control polices are restrictive and possible sub-optimal. As such, more general control policies need to be studied as they have the potential to handle a much larger class of real-time fluctuations, possibly with limited information.

Design of general control policies poses significant technical/computational challenges and the choice of parameterization for their mathematical representation plays a pivotal role in their design. First, the expressive power of the parameterization dictates how general of a control policy it can represent. Second, the parameterization chosen must be compatible with the corresponding SOPF formulation to ensure computational tractability. In what follows, we provide a broad classification of approaches in the literature to this coupled control-parameterization and algorithm-design problem:
i) Affine policy. Here the control is restricted to be a linear function typically of the net system uncertainty, which makes it easy to model within OPF. Such affine policies have been used for convex SOPF with the DC power flow model [10] as well as for scenario-based approaches [96], and SOPF with nonlinear AC power flow model [17, 69].
ii) Non-affine policies include non-linear and hence more general policies that can ensure better feasibility and optimality enforcement over the affine ones. However they are computationally harder to incorporate within the SOPF. Examples include polynomial chaos based policies [68, 70], as well as non-liner policies with generator saturation [81].
iii) Nonlinear policies using robust optimization. This approach uses a fully optimized recourse for each uncertainty realization using a min-max-min formulation [62]. The problem is highly computationally challenging and approximation and relaxations are employed that can lead to conservative solutions. Similar robust formulations with affine control have also been investigated in the literature [56].

iv) Nonlinear policies using data-driven methods. This approach uses historical data or simulated OPF solutions to devise efficient non-linear control policies as well as to improve the computational speed of the SOPF. Examples of such methods include kernel-based control policies for voltage control [48] and active set learning approaches [18, 75] for OPF. Owing to the advancements in deep learning over the past decade, DNN-based OPF solution schemes have also been explored by the power systems community. Taking the conventional supervised learning route, DNN-based OPF solvers have been trained to match the optimal labels [15, 102]. However, solving a large number of OPFs to generate optimal labels is itself a computationally overwhelming task. Sensitivity-informed deep learning [86, 88], which matches not only OPF minimizers but also partial derivatives with respect to the inputs, partially alleviates the above concern by improving upon the data efficiency. Nonetheless, all these DNN-based solvers entail a label generation stage, which contributes significantly to training overheads.

In this paper, we consider a DNN-based control policy design for SOPF. The DNN parameterization serves as a generalization to non-linear policies, but also provides backward compatibility with previous policies since it can easily accommodate communication constraints and restricted features (e.g., control based on total/net uncertainty and/or local control). Different from the aforementioned works, our approach does not involve an offline data generation phase where a large number of OPF or OPF-like problems are solved. Instead, similar to [34, 37], the training phase of the DNN itself serves as the SOPF solver, which if deployed within current practices will replace the OPF solved at the 5-15 minutes time scale. Crucially, we also model the *stochastic* non-linear power system constraints during training to ensure feasibility of the control actions. Thus our approach uses a hybrid model that has the advantages of the NNs such as high representation power and the potential for parallelization and GPU implementation, without sacrificing feasibility like black-box data-driven efforts.

Prior works on constrained unsupervised learning for DNNs focus on deterministic OPFs. Penalty terms that grow with constraint violations have been added to the training loss function in [45]. A new set of hyperparameters corresponding to coefficients of the penalty terms are introduced and need to be pre-tuned for achieving a desirable performance. Alternatively, a constraint completion and correction strategy is discussed in [20], which improves constraint satisfaction via post-processing on DNN predictions. In contrast, our work aims to design DNN-based solvers for SOPFs where constraint violations are permissible for some samples. Instead of introducing hyperparameters, a primal-dual learning procedure is implemented which finds the DNN weights and dual variables while solving the stochastic OPFs. Lastly, during the implementation phase, no post-processing is required and DNN predictions are implemented straightaway. To the best of our knowledge, this is the first approach to explicitly model stochastic constraints inside a DNN-based OPF formulation.

The rest of the paper is organized as follows. Section 4.4 presents the mathematical formulation for SOPF with system and control constraints. Section 4.5 describes in detail our control policy design, in particular the modeling and training of our neural network with system constraints. Section 4.6 includes experimental validation of our approach and comparison with existing methods on IEEE test cases. Finally, Section 4.7 concludes the paper.

4.4 Problem Formulation

4.4.1 Grid Modeling

Consider a power transmission network modeled as a directed connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. The nodes $n \in \mathcal{N} := \{1, \dots, N\}$ of the graph $\mathcal{G}$ correspond to network buses, while the directed edges $e_{nk} \in \mathcal{E}$ to transmission lines. The complex voltage at bus n is expressed in polar coordinates as $v_n e^{j\theta_n}$, and the complex power injection at the same bus as $p_n + jq_n$. The power injections at node n are described by the AC power flow equations (AC-PF) as

$$p_n = v_n \sum_{k=1}^{N} v_k \left(G_{nk} \cos\theta_{nk} + B_{nk} \sin\theta_{nk}\right) \tag{4.1a}$$

$$q_n = v_n \sum_{k=1}^{N} v_k \left(G_{nk} \sin\theta_{nk} - B_{nk} \cos\theta_{nk}\right) \tag{4.1b}$$

where $\theta_{nk} := \theta_n - \theta_k$, and G_{nk} and B_{nk} are the entries of the real and imaginary parts of the bus admittance matrix $\mathbf{Y} = \mathbf{G} + j\mathbf{B}$. Power injections can be decomposed into their dispatchable (p_n^g, q_n^g) and inflexible (p_n^d, q_n^d) parts as

$$p_n = p_n^g - p_n^d \quad \text{and} \quad q_n = q_n^g - q_n^d.$$

The former corresponds to generators and flexible loads; the latter to inelastic loads hosted per bus n. The buses hosting dispatchable injections form set $\mathcal{N}_g \subset \mathcal{N}$, and its complement is defined as $\mathcal{N}_\ell$. Set $\mathcal{N}_g$ will be henceforth referred to as the set of *generator* buses, and $\mathcal{N}_\ell$ as the set of *load* buses. To simplify the exposition, each generator bus is assumed to host exactly one dispatchable unit. The bus indexed by $n = 1$ is designated as the slack and reference bus with $\theta_1 = 0$.

The complex power $P_{nk} + jQ_{nk}$ flowing from bus n to bus k over line e_{nk} can be expressed as

$$P_{nk} = v_n v_k \left(G_{nk} \cos\theta_{nk} + B_{nk} \sin\theta_{nk}\right) - v_n^2 G_{nk} \tag{4.2a}$$

$$Q_{nk} = v_n v_k \left(G_{nk} \sin\theta_{nk} - B_{nk} \cos\theta_{nk}\right) + v_n^2 \left(B_{nk} - B_{nk}^{\text{sh}}\right) \tag{4.2b}$$

where B_{nk}^{sh} is the shunt susceptance of line e_{nk}. The apparent power flow on the line is

$$f_{n,k} := \sqrt{P_{n,k}^2 + Q_{n,k}^2}. \tag{4.3}$$

4.4.2 Optimal Power Flow (OPF)

Let $c_n(p_n^g)$ denote the cost of generation at bus n. Given inflexible loads $\{p_n^d, q_n^d\}_{n\in\mathcal{N}}$, the OPF problem aims at minimizing the total cost of generation while operating the transmission network within limits. The OPF is posed as

$$\begin{aligned}
\min \quad & \sum_{n\in\mathcal{N}_g} c_n(p_n^g) && && \text{(4.4a)}\\
\text{over} \quad & \{v_n,\theta_n\}_{n\in\mathcal{N}}, \{p_n^g, q_n^g\}_{n\in\mathcal{N}_g} && && \\
\text{s.to} \quad & (4.1), (4.2) && \forall n\in\mathcal{N}, \forall e_{nk}\in\mathcal{E} && \\
& p_n = p_n^g - p_n^d && \forall n\in\mathcal{N}_g && \text{(4.4b)}\\
& q_n = q_n^g - q_n^d && \forall n\in\mathcal{N}_g && \text{(4.4c)}\\
& p_n = -p_n^d && \forall n\in\mathcal{N}_\ell && \text{(4.4d)}\\
& q_n = -q_n^d && \forall n\in\mathcal{N}_\ell && \text{(4.4e)}\\
& \underline{p}_n^g \leq p_n^g \leq \bar{p}_n^g && \forall n\in\mathcal{N}_g && \text{(4.4f)}\\
& \underline{q}_n^g \leq q_n^g \leq \bar{q}_n^g && \forall n\in\mathcal{N}_g && \text{(4.4g)}\\
& \underline{v}_n \leq v_n \leq \bar{v}_n && \forall n\in\mathcal{N} && \text{(4.4h)}\\
& f_{n,k} \leq \bar{f}_{n,k} && \forall e_{n,k}\in\mathcal{E} && \text{(4.4i)}\\
& \theta_1 = 0 && && \text{(4.4j)}
\end{aligned}$$

where (4.1) and (4.4b)–(4.4d) enforce the power flow equations for power injections. Constraints (4.4f)-(4.4g) represent generation limits. Constraint (4.4h) confines voltage magnitudes within given limits. Constraints (4.2) and (4.4i) ensure that apparent power flows remain within line ratings. Lastly, constraint (4.4j) fixes the phase angle at the reference bus.

Given load demands stored in vector $\boldsymbol{\phi} := \{p_n^d, q_n^d\}_{n\in\mathcal{N}}$, the system operator solves (4.4) to find the optimal voltage and active power set-points for generators, which can be collected in vector $\mathbf{x} := \{\{v_n\}_{n\in\mathcal{N}_g}, \{p_n^g\}_{n\in\mathcal{N}_g\setminus\{1\}}\}$. Given $(\boldsymbol{\phi}, \mathbf{x})$, all other grid quantities involved in the OPF can be expressed as functions of $(\boldsymbol{\phi}, \mathbf{x})$ implicitly via the AC-PF equations (4.1) and (4.2). These quantities include the bus generations, and (p_1^g, q_g^1) which we collectively represent using the variable $\mathbf{y}$. The OPF of (4.4) can be abstracted as a parametric optimization solely over variable $\mathbf{x}$ given parameters $\boldsymbol{\phi}$:

$$\min_{\mathbf{x}} \quad \sum_{n\in\mathcal{N}_g} c_n\left(\mathbf{x}, \boldsymbol{\phi}\right) \tag{4.5a}$$

$$\text{s.to} \quad \underline{\mathbf{x}} \leq \mathbf{x} \leq \bar{\mathbf{x}} \tag{4.5b}$$

$$\mathbf{y}\left(\mathbf{x}, \boldsymbol{\phi}\right) \leq \bar{\mathbf{y}}. \tag{4.5c}$$

Here (4.5b) captures the constraints on generator setpoints, that is (4.4f) and (4.4h) for $n \in \mathcal{N}_g$, while (4.5c) captures the constraints in (4.4g), (4.4h), and (4.4i).

Heed there is no need to explicitly enforce the constraints on loads as in (4.4b)–(4.4e). This is because active and reactive load demands have been included in $\boldsymbol{\phi}$, and their effect on all other grid quantities of interest is captured by the power flow equations, which are still taken into account in (4.5) indirectly through the constraint function $\mathbf{y}(\mathbf{x}, \boldsymbol{\phi})$. Note also that even though the mapping $\mathbf{y}(\mathbf{x}, \boldsymbol{\phi})$ does not feature an analytical form, we will be able to compute its partial derivatives with respect to $\mathbf{x}$ via the chain rule and the inverse function theorem later in Section 4.5.3. For now, let us explain how the OPF in (4.5) can be modified to cope with uncertainty in $\boldsymbol{\phi}$.

4.4.3 Chance-Constrained Optimal Power Flow

Solving the non-convex problem of (4.5) can be computationally taxing if $\boldsymbol{\phi}$ changes frequently. Also by the time (4.5) is solved and the optimal setpoints are communicated to generators, demands $\boldsymbol{\phi}$ may have changed rendering $\mathbf{x}$ obsolete. The above concerns can be accounted for by considering a stochastic version of the OPF [97], [55]. We consider the chance constrained OPF (CC-OPF), that treats $\boldsymbol{\phi}$ as a random variable and seeks a control policy $\mathbf{x}(\boldsymbol{\phi})$ that reacts to the realization of $\boldsymbol{\phi}$ to minimize the expected generation cost while satisfying network constraints with high probability

$$\min_{\mathbf{x}(\boldsymbol{\phi})\in\mathcal{X}} \quad \mathbb{E}\left[\sum_{n\in\mathcal{N}_g} c_n\left(\mathbf{x},\boldsymbol{\phi}\right)\right] \tag{4.6a}$$

$$\Pr\left[y_i\left(\mathbf{x},\boldsymbol{\phi}\right)\leq \bar{y}_i\right]\geq 1-\alpha, \qquad i=1:M \tag{4.6b}$$

where the expectation $\mathbb{E}$ and the probability Pr are with respect to $\boldsymbol{\phi}$; and parameter $\alpha\in(0,1)$ controls the probability of constraint violation. The deterministic constraint (4.5b) has been abstracted as $\mathbf{x}\in\mathcal{X}$. Using the indicator function, the chance constraints in (4.6b) can be recast as

$$\mathbb{E}\left[\mathbb{1}\left(y_i\left(\mathbf{x},\boldsymbol{\phi}\right)-\bar{y}_i\right)\right]\geq 1-\alpha,\ i=1:M \tag{4.7}$$

where the indicator function $\mathbb{1}(x)$ takes the values of 1 and 0 for $x\leq 0$ and $x>0$, respectively. The problem (4.6) is a variational optimization problem over the control policy $\mathbf{x}(\boldsymbol{\phi})$ and is generally intractable in its native form. Instead, one can parameterize the control policy as $\mathbf{x}(\boldsymbol{\phi}) = \boldsymbol{\pi}\left(\boldsymbol{\phi};\mathbf{w}\right)$, where $\boldsymbol{\pi}(.)$ is a chosen function of $\boldsymbol{\phi}$ determined by the parameters $\mathbf{w}$. Restricting the policy to take this parameterized form, (4.6) can be written as the constrained stochastic minimization

$$\min_{\mathbf{w}:\boldsymbol{\pi}_{\mathbf{w}}\in\mathcal{X}} \quad \mathbb{E}\left[\sum_{n\in\mathcal{N}_g} c_n\left(\boldsymbol{\pi}_{\mathbf{w}},\boldsymbol{\phi}\right)\right] \tag{4.8a}$$

$$\text{s.to} \quad \mathbb{E}\left[\mathbb{1}\left(y_i\left(\boldsymbol{\pi}_{\mathbf{w}},\boldsymbol{\phi}\right)-\bar{y}_i\right)\right]\geq 1-\alpha, \qquad i=1:M$$

where the optimization is now over the policy parameters $\mathbf{w}$, and the notation has been slightly abused by simplifying $\boldsymbol{\pi}\left(\boldsymbol{\phi};\mathbf{w}\right)$ to $\boldsymbol{\pi}_{\mathbf{w}}$. In general (4.8) is an inner approximation of (4.6) due to the restriction imposed on the policy.Given their universal approximation capabilities [44], DNNs constitute great candidates for modeling the policy $\boldsymbol{\pi}_{\mathbf{w}}$ and narrowing down the gap between (4.6) and (4.8). In the case of DNNs, the parameter vector $\mathbf{w}$ collects the weighs and biases across all layers of the DNN.

We next delineate how a DNN can be trained to solve the CC-OPF in (4.8). Unlike standard feed-forward DNNs, our approach involves an iterative primal-dual scheme that inherently satisfies the non-convex stochastic constraints involved in (4.8).

4.5 Finding Optimal Policies

This section approximates the indicator function in (4.8) with a logistic function; adopts a stochastic primal-dual scheme to learn a DNN to solve the CC-OPF. It explains how the required gradients can be computed, and provides an overview of the training and operational phase of the DNN.

4.5.1 Approximating the Indicator Function

The weights $\mathbf{w}$ for a DNN-based policy can be learned by solving (4.8). However, computing the expectations in (4.8) is challenging even if the pdf of $\boldsymbol{\phi}$ is known, given the non-linearities in the objective and constraints. Standard DNN training alleviates this issue by using stochastic gradient descent (SGD) updates over a training dataset generated by drawing random samples with respect to the distribution of $\boldsymbol{\phi}$ to learn the DNN weights. In the presence of constraints, stochastic primal-dual updates (SPD) can be used to yield a similar training process [22], [34]. Unfortunately, the indicator function $\mathbb{1}(x)$ in (4.8) prevents the application of any gradient-based approach. This is because $\mathbb{1}(x)$ is discontinuous at $x = 0$, and its derivative becomes 0 at $x \neq 0$. Hence, no useful gradients are obtained while applying SPD.

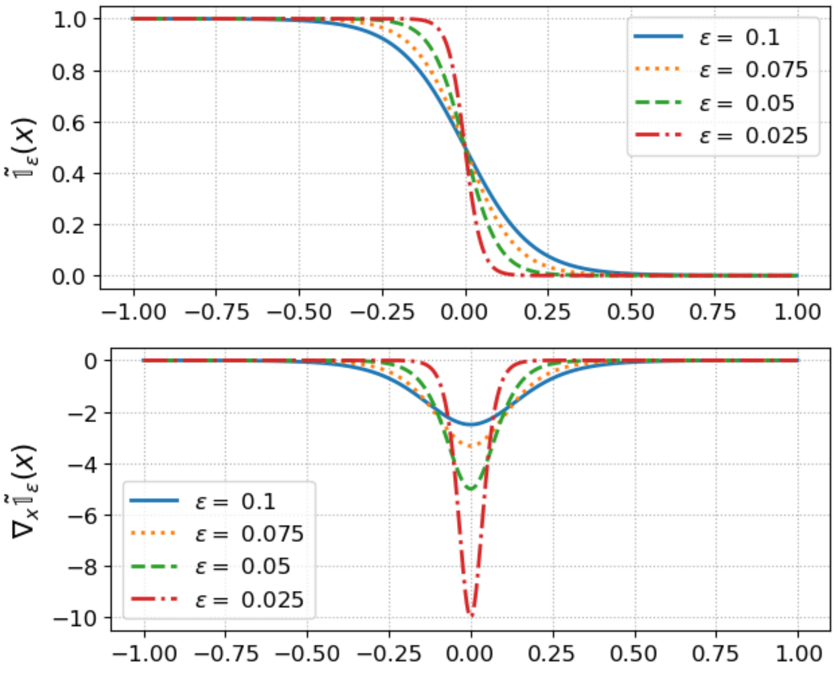


Figure 4.1: Logistic function approximation of the indicator function *(top)*, and its derivative *(bottom)* for different values of parameter ϵ.

The non-differentiability of $\mathbb{1}(x)$ has been addressed in the literature by substituting it with a differentiable approximation. Depending on the objective and constraints, the convex approximations of $\mathbb{1}(x)$ proposed in [73] may yield overall convex CC-OPF formulations [17]. Nonetheless, the computational advantage gained by introducing convexity is balanced off by the sub-optimal, conservative nature of these formulations. It is worth stressing that the problem in (4.8) has sources of non-convexity other than $\mathbb{1}(x)$, namely the policy $\boldsymbol{\pi}(\boldsymbol{\phi}; \mathbf{w})$ and the underlying non-linear power flow equations. Therefore, aiming for a more accurate and differentiable approximation of $\mathbb{1}(x)$ is pertinent, even if this approximation is non-convex. Fortunately, the logistic function, which is well known in the ML community, can serve as a smooth surrogate of $\mathbb{1}(x)$ [13]. The logistic function and its derivative are

$$\tilde{\mathbb{1}}_\epsilon(x) := \frac{e^{-x/\epsilon}}{1+e^{-x/\epsilon}}, \quad \nabla_x \tilde{\mathbb{1}}_\epsilon(x) = \frac{-\tilde{\mathbb{1}}_\epsilon(x)\left(1-\tilde{\mathbb{1}}_\epsilon(x)\right)}{\epsilon} \tag{4.9}$$

where parameter ϵ controls the accuracy of the approximation. It is easy to verify that $\tilde{\mathbb{1}}_\epsilon(x)$ tends to $\mathbb{1}_\epsilon(x)$ as $\epsilon \to 0^+$ as demonstrated in Figure 4.1. The same figure points towards a possible trade-off between approximation error and rate of convergence for SPD: As ϵ decreases, the range of values of x over which $\tilde{\mathbb{1}}_\epsilon(x)$ has non-diminishing derivative decreases as well. It is hence anticipated that for smaller values of ϵ, a larger number of SPD updates may be needed.

4.5.2 Stochastic Primal-Dual Updates

Dualizing (4.8) with $\mathbb{1}(x)$ being replaced by $\tilde{\mathbb{1}}_\epsilon(x)$ provides the Lagrangian function

$$\begin{aligned} L(\mathbf{w}; \boldsymbol{\lambda}) := \mathbb{E}\left[\sum_{n\in\mathcal{N}_g} c_n\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}\right)\right] & \\ + \boldsymbol{\lambda}^\top \left[(1-\alpha)\mathbf{1} - \mathbb{E}\left[\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}\left(\boldsymbol{\pi}_\mathbf{w}, \boldsymbol{\phi}\right) - \bar{\mathbf{y}}\right)\right]\right] & \end{aligned} \tag{4.10}$$

where $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers associated with the constraints in (4.8a). Observe that $\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}\left(\boldsymbol{\pi}_\mathbf{w},\boldsymbol{\phi}\right)-\bar{\mathbf{y}}\right)$ are vector quantities obtained by applying $\tilde{\mathbb{1}}_\epsilon(x)$ element-wise for all constraints in (4.8a). The corresponding dual problem is

$$D^* := \max_{\boldsymbol{\lambda}\geq\mathbf{0}} \min_{\mathbf{w}:\boldsymbol{\pi}_\mathbf{w}\in\mathcal{X}} .L(\mathbf{w};\boldsymbol{\lambda}) \tag{4.11}$$

A stationary point of this min/max problem can be reached via the projected primal-dual iterations indexed by k

$$\mathbf{w}_{k+1} := \left[\mathbf{w}_k - \mu\nabla_\mathbf{w} L(\mathbf{w}_k\boldsymbol{\lambda}_k)\right]_{\boldsymbol{\pi}_\mathbf{w}\in\mathcal{X}} \tag{4.12a}$$

$$\boldsymbol{\lambda}_{k+1} := \left[\boldsymbol{\lambda}_k + \nu\nabla_{\boldsymbol{\lambda}} L(\mathbf{w}_k;\boldsymbol{\lambda}_k)\right]_+ \tag{4.12b}$$

with positive step sizes μ and ν. The primal update (4.12a) involves a projection of $\boldsymbol{\pi}_\mathbf{w}$ into the feasible set $\mathcal{X}$. With $\boldsymbol{\pi}_\mathbf{w}$ being modeled by a DNN, projection onto $\mathcal{X}$ can be easily accomplished by using appropriately scaled hyperbolic tangent functions (tanh). Dual variables are projected on the non-negative orthant via the operator $[x]_+ := \max\{x,0\}$.

The expectation operator in (4.10) can be handled using stochastic approximation. This entails first surrogating each of the expected values by their sample averages over K uncertainty realizations $\{\boldsymbol{\phi}^k\}_{k=1}^K$. For the cost function for example, this approximation yields

$$\mathbb{E}\left[\sum_{n\in\mathcal{N}_g} c_n\left(\boldsymbol{\pi}_\mathbf{w},\boldsymbol{\phi}\right)\right] \simeq \frac{1}{K}\sum_{k=1}^{K}\sum_{n\in\mathcal{N}_g} c_n\left(\boldsymbol{\pi}_\mathbf{w},\boldsymbol{\phi}^k\right).$$

The *stochastic* primal-dual (SPD) updates further reduces the computational effort by using one uncertainty realization per iteration to obtain:

$$\mathbf{w}_{k+1} := \left[\mathbf{w}_k - \mu\sum_{n\in\mathcal{N}_g}\nabla_\mathbf{w} c_n^k + \mu\boldsymbol{\lambda}_k^\top\nabla_\mathbf{w}\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}^k-\bar{\mathbf{y}}\right)\right]_{\boldsymbol{\pi}_\mathbf{w}\in\mathcal{X}} \tag{4.13a}$$

$$\boldsymbol{\lambda}_{k+1} := \left[\boldsymbol{\lambda}_k + \nu(1-\alpha)\mathbf{1} - \nu\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}^k - \bar{\mathbf{y}}\right)\right]_+, \tag{4.13b}$$

where $c_n^k := c_n\left(\boldsymbol{\pi}_{\mathbf{w}}, \boldsymbol{\phi}^k\right)$ and $\mathbf{y}^k := \mathbf{y}\left(\boldsymbol{\pi}_{\mathbf{w}}, \boldsymbol{\phi}^k\right)$.

4.5.3 Computing Gradients for Primal Updates

The stochastic dual update simply requires evaluating the indicator approximation $\tilde{\mathbb{1}}(x)$ for the constraint functions $\mathbf{y}^k$. The stochastic primal update of (4.13a) however is more involved as it requires evaluating the gradients $\{\nabla_{\mathbf{w}} c_n^k\}$ and the Jacobian matrix $\nabla_{\mathbf{w}}\tilde{\mathbb{1}}_\epsilon\left(\mathbf{y}^k - \bar{\mathbf{y}}\right)$. We next explain how these gradients can be computed via the chain rule and the inverse function theorem.

We commence with gradient $\nabla_{\mathbf{w}} c_n$ of the generation cost functions for all but the generator at the reference bus:

$$(\nabla_{\mathbf{w}} c_n)^\top = \frac{\mathrm{d}c_n}{\mathrm{d}p_n^g} \cdot (\nabla_{\mathbf{x}} p_n^g)^\top \cdot \nabla_{\mathbf{w}}\mathbf{x}, \quad \forall n \in \mathcal{N}_g \setminus \{1\}. \tag{4.14}$$

From the definition of $\mathbf{x}$, the gradient $\nabla_{\mathbf{x}} p_g^n$ is simply a canonical vector. The Jacobian matrix $\nabla_{\mathbf{w}}\mathbf{x}$ contains the partial derivatives of the DNN outputs with respect to the DNN inputs, and can be computed by most deep learning platforms such as TensorFlow using gradient *back-propagation* in a computationally efficient manner.

Computing $\nabla_{\mathbf{w}} c_1$ for the generation cost at the reference bus is less direct, yet still computationally efficient. It is less direct because p_1^g is not part of the setpoints, and hence, the DNN output $\mathbf{x}$. Nonetheless, it does depend on all other generator setpoints and in turn $\mathbf{x}$ indirectly, through the power flow equations. If we introduce the vector of voltages in polar coordinates $\mathbf{u} := [v_1 \ \ldots \ v_N \ \theta_2 \ \ldots \ \theta_N]^\top$ excluding the fixed angle $\theta_1 = 0$, the sought gradient can be computed as

$$(\nabla_{\mathbf{w}} c_1)^\top = \frac{\mathrm{d}c_1}{\mathrm{d}p_1^g} \cdot (\nabla_{\mathbf{u}} p_1^g)^\top \cdot \nabla_{\mathbf{x}}\mathbf{u} \cdot \nabla_{\mathbf{w}}\mathbf{x} \tag{4.15}$$

The Jacobian matrix $\nabla_{\mathbf{x}}\mathbf{u}$ required in (4.15) can be found by solving a system of linear equations. Let vector $\mathbf{z}$ collect the active and reactive power demands at all load buses. Then, using the power flow equations in (4.1), vectors $\mathbf{x}$ and $\mathbf{z}$ can be abstractly expressed as functions of $\mathbf{u}$ as

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{g}(\mathbf{u}) \\ \boldsymbol{\ell}(\mathbf{u}) \end{bmatrix}. \tag{4.16}$$

Regarding the mapping $\mathbf{g}(\mathbf{u})$, recall that $\mathbf{x} := \{\{v_n\}_{n\in\mathcal{N}_g}, \{p_n^g\}_{n\in\mathcal{N}_g\setminus\{1\}}\}$. Then, the first $|\mathcal{N}_g|$ entries of $\mathbf{g}(\mathbf{u})$ are simply $\mathbf{e}_n^\top\mathbf{u}$, where $\mathbf{e}_n$ is the n-th canonical vector. The second $|\mathcal{N}_g|$ entries of $\mathbf{g}(\mathbf{u})$ follow from the power flow equations plus any possible load p_n^ℓ at the corresponding bus. Since p_1^g is a dependent variable, it has not been included in $\mathbf{x}$ or $\mathbf{z}$. Differentiating either sides of (4.16) with respect to $\mathbf{x}$ yields:

$$\begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \nabla_{\mathbf{u}}\mathbf{g} \\ \nabla_{\mathbf{u}}\boldsymbol{\ell} \end{bmatrix} \nabla_{\mathbf{x}}\mathbf{u} \tag{4.17}$$

where $\mathbf{I}$ is an identity matrix of dimensions $2|\mathcal{N}_g| - 1$ and $\mathbf{0}$ is a matrix of all zeroes of dimension $2|\mathcal{N}_\ell| \times (2|\mathcal{N}_g| - 1)$. From (4.17), we can now compute the Jacobian matrix $\nabla_{\mathbf{x}}\mathbf{u}$ as

$$\nabla_{\mathbf{x}}\mathbf{u} = \begin{bmatrix} \nabla_{\mathbf{u}}\mathbf{g}(\mathbf{u}) \\ \nabla_{\mathbf{u}}\boldsymbol{\ell}(\mathbf{u}) \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \tag{4.18}$$

where the matrix to be inverted here is $(2N-1)\times(2N-1)$.

The Jacobian matrix $\nabla_{\mathbf{w}}\tilde{\mathbb{1}}_{\epsilon}(\mathbf{y}-\bar{\mathbf{y}})$ appearing in the primal update of (4.13a) can be found in a similar application of the chain rule and the inverse function theorem as

$$\nabla_{\mathbf{w}}\tilde{\mathbb{1}}_{\epsilon}(\mathbf{y}-\bar{\mathbf{y}}) = \nabla_{\mathbf{y}}\tilde{\mathbb{1}}_{\epsilon}(\mathbf{y}-\bar{\mathbf{y}})\cdot\nabla_{\mathbf{u}}\mathbf{y}\cdot\nabla_{\mathbf{x}}\mathbf{u}\cdot\nabla_{\mathbf{w}}\mathbf{x}. \tag{4.19}$$

Matrix $\nabla_{\mathbf{y}}\tilde{\mathbb{1}}_{\epsilon}(\mathbf{y}-\bar{\mathbf{y}})$ is diagonal with diagonal entries provided by (4.9). Matrix $\nabla_{\mathbf{u}}\mathbf{y}$ contains the partial derivatives of all constraint functions of interest with respect to voltage magnitudes and angles. Such derivatives can be readily computed from (4.1)–(4.3).

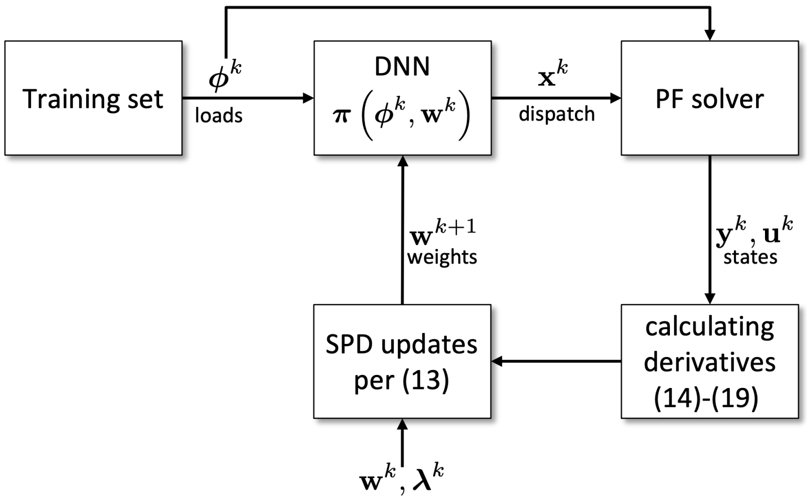

Training set
$\boldsymbol{\phi}^k$
loads
DNN
$\pi\left(\boldsymbol{\phi}^k,\mathbf{w}^k\right)$
$\mathbf{x}^k$
dispatch
PF solver
$\mathbf{y}^k,\mathbf{u}^k$
states
$\mathbf{w}^{k+1}$
weights
SPD updates per (13)
calculating derivatives (14)-(19)
$\mathbf{w}^k,\boldsymbol{\lambda}^k$

Figure 4.2: Overview of the training phase for the DNN-based policy.

Remark 4.1. In a typical OPF formulation, one would select complex voltages in rectangular or polar coordinates $\mathbf{u}$ as the optimization variables and then express each quantity in cost or constraints (injections, line flows, voltage magnitudes) as an analytic function (e.g., quadratic) of complex voltages. In the proposed formulation on the other hand, the optimization variables are the generator PV setpoints stored in $\mathbf{x}$; the PQ setpoints are known problem parameters collected in $\mathbf{z}$. Cost and constraints can be expressed as functions of the PV/PQ point $(\mathbf{x},\mathbf{z})$. Although such functions may be implicit (no analytic form), one can still compute their values and derivatives with respect to $\mathbf{x}$ at some $(\mathbf{x}, \mathbf{z})$. This is possible upon solving the power flow (PF) equations at $(\mathbf{x}, \mathbf{z})$ and using the inverse function theorem as detailed between (4.16)–(4.19), and under the tacit assumption that the PF equations are solvable at $(\mathbf{x}, \mathbf{z})$. The proposed method does not explicitly ensure PF solvability for the entire sequence of iterates $\mathbf{x}_k = \boldsymbol{\pi}(\mathbf{z}; \mathbf{w}_k)$ and across all scenarios. It is anticipated that if the iterates are initialized at a solvable PV/PQ point $(\mathbf{x}, \mathbf{z})$ and the step size μ is sufficiently small, it would be unlikely for the iterates in (4.13) to land at a PV/PQ point with no PF solution. This is intuitively justified by the continuity of the PF equations and the fact that when a PF trajectory is approaching insolvability, the PF solutions would become infeasible (e.g., voltages being well outside the desirable range). Practically, it should be noted that during our numerical tests, we did not encounter any scenario where a PV/PQ point would have no PF solution during an iteration.

4.5.4 Deployment Workflow

Deploying the DNN-based strategy consists of two phases; (i) *solving the SOPF*, which is achieved by the training phase of the DNN, and (ii) *real time computation of control actions*, which is done by simply evaluating the trained DNN and is similar to the testing phase of DNNs. For the training phase, the system operator samples K grid conditions $\{\boldsymbol{\phi}^k\}_{k=1}^{K}$ that reflect the real-time conditions to which the policy will be applied. The training samples could be sourced from historically recorded loads, simulations, or predictions. Depending on the generators that have been committed for the upcoming time-period, the operator identifies generator and load buses in sets $\mathcal{N}_g$ and $\mathcal{N}_\ell$, respectively, and defines vectors $\mathbf{x}$ and $\mathbf{y}$. The input and output layers of the DNN match the dimensions of $\boldsymbol{\phi}$ and $\mathbf{x}$, accordingly. The dimensions and the number of hidden layers are hyper-parameters to be determined for the specific CC-OPF setting. The DNN is then trained using SPD as per Section 4.5.

This entails: *i)* Sampling a $\boldsymbol{\phi}^k$; *ii)* Performing a forward pass through the DNN to obtain $\mathbf{x}^k$; *iii)* Obtaining $\mathbf{y}^k$ and $\mathbf{u}^k$ from a power flow solver; *iv)* Using $\{\theta^k, \mathbf{x}^k, \mathbf{y}^k, \mathbf{u}^k\}$ to calculate the gradients (4.14), (4.15), and (4.19); and *v)* Performing updates (4.13a)–(4.13b). These steps are repeated over the training samples for multiple epochs. Figure 4.2 depicts a block diagram of the training process. In real-time, the operator simply feeds the DNN with the real-time realizations of the uncertainty $\boldsymbol{\phi}$ to obtain the dispatch $\mathbf{x}$ at the output.

4.6 Numerical Tests

4.6.1 Experimental Setup

The performance of the proposed DNN-based policy is evaluated using transmission networks of varying sizes. The simulation scripts are written in Python and run on a 2.4 GHz 8-Core Intel Core i9 processor laptop computer with 64 GB RAM. Compatibility with the commonly used MATPOWER [110] models is achieved by interfacing the Python script with the open-source Octave engine [21]. The communication between the two platforms is enabled via the Oct2py library [2], which allows seamless calling of M-files and Octave functions from Python. Leveraging upon this functionality, in-built MATPOWER functions are called to read the networks, solve the power flow equations, and calculate the derivatives needed for the primal updates. The implementation advantages of such synergy come at the cost of the communication overhead between the two platforms. For the networks tested, this overhead is within $0.02 - 0.03$ seconds per iteration. Because these delays can be avoided by porting the required MATPOWER functions to Python, they have been eliminated from the training times reported here.

TensorFlow libraries are employed to model and train the DNN. For each of the experiments, a five-layer DNN is considered. The five layers are: *1)* input layer matching the dimensions of $\boldsymbol{\phi}$; *2)* two hidden layers, each with half the dimensions of the input layer; *3)* output layer with the same dimensions as $\mathbf{x}$; and *4)* custom designed activation layer based on tanh ensuring $\mathbf{x} \in \mathcal{X}$. All DNN weights are initialized by randomly drawing from a standard normal distribution. DNN biases and dual variables are initialized at zero. The primal updates are performed using the Adam optimizer and the dual variables are updated using SGD. The primal step size μ decays exponentially at the rate of 0.5 per epoch, whereas the dual step size ν decays with the square-root of the iteration index [61]. The values for hyper-parameters $\{\epsilon, E, \mu_0, \nu_0\}$ are identified for each network via cross-validation, and are reported with results from the experiment.

For training and testing the DNN, we generated $1,000$ samples of $\boldsymbol{\phi}$ by adding zero-mean uniformly distributed noise to the nominal loads of the MATPOWER models. Recall that training corresponds to solving the SOPF formulation in (4.8) and testing corresponds to using the computed policy for real time prediction of control actions in response to uncertainty. To ensure that the problem in (4.8) is feasible, the generated demands were truncated within a range of $[-R, R]$ around the nominal point, where R was selected per network to ensure that the corresponding deterministic formulation of the OPF in (4.5) is feasible. The $1,000$ generated samples were then randomly grouped into training and testing sets of 800 and 200 examples, respectively. Training was conducted over E epochs of the training set. The performance of the DNN-based policy was bench-marked against the *OPF-policy* that solves the OPF in (4.5) deterministically per $\boldsymbol{\phi}_k$.

4.6.2 Accuracy of the Logistic Function Approximation

We investigate the accuracy and performance of the logistic function approximation using $\tilde{\mathbb{1}}_\epsilon(x)$ to the chance constraints in (4.7) using the 14-bus "pglib_opf_case14_ieee" network. For $\alpha \in \{0.05, 0.10, 0.15, 0.20\}$ and $\{\epsilon, E, \mu_0, R\} = \{0.01, 5, 10^{-3}, 0.1\}$, Table 4.1 shows the values of the remaining hyperparameters and the training time. The performance of the DNN-based policy on unseen test samples is also divulged via three metrics: maximum sampled probability of constraint violations (abbreviated in Table 4.1 as *maximum violation* [%]), test time, and the average cost.

Table 4.1 demonstrates the advantages of the DNN policy. First, the evaluated violation probabilities closely follows the prescribed α showing that the approximation $\tilde{\mathbb{1}}_\epsilon(x)$ is sufficiently accurate while facilitating use of the efficient SPD algorithm. This is an improvement over the conservative convex approximation of the indicator function in [34], where the observed violation probabilities were considerably less than α. Second, during the real-time evaluation of the policy, the DNN-based policy is able to predict the dispatch for the 200 test samples in around 0.3 seconds. Comparing this to the OPF-policy, which has an evaluation time of 31.3 seconds and cost of \$2180.16, the DNN policy is approximately 100 times faster without sacrificing optimality.

Table 4.1: Training and testing details for the 14-bus system for different values of α.

α	ν_0	Train time (sec)	Maximum violation [%]	Test time (sec)	Average cost [\$]
0.05	$3 \cdot 10^{-4}$	97.4	3.5	0.30	2180.53
0.10	$1.5 \cdot 10^{-4}$	98.5	8.5	0.33	2180.48
0.15	$1 \cdot 10^{-4}$	100.6	16.5	0.34	2180.45
0.20	$1.8 \cdot 10^{-4}$	100.6	17.0	0.34	2180.44

4.6.3 Enforcing Communication and Complexity Constraints

While our presentation thus far has been focused on the unconstrained DNN policy where all controllable variables are allowed to respond to uncertainty realizations, the framework also allows for seamless encoding of communication constraints where only part of $\boldsymbol{\phi}$'s is observed and communicated, and complexity constraints where only a subset of the control variables respond to uncertainty. This flexibility allows us to account for any limitations of the available infrastructure.

As an example, we consider an automatic generation control (AGC) type policy, where only the active power generation of the participating generators responds to the total active load deviation in the system. This policy closely resembles the usual affine policy model in terms of input-output dependencies but allows for more general non-linear functions. This policy can be implemented within the DNN framework by modifying the input layer to have only 1 neuron that receives the signal $\sum_{n\in\mathcal{N}} p_n^d$, and making the neurons for $\{v_n^g\}_{n\in\mathcal{N}^g}$ insensitive to the input by setting the corresponding weights to 0. Note that the modified DNN still produces the average nominal values for $\{v_n^g\}_{n\in\mathcal{N}^g}$ at the output, because the biases in the output layer are learnt during training.

The AGC-type policy along with the full policy are evaluated on the 14 bus system for $\alpha = 0.1$ and two values of $R = \{0.1, 0.2\}$ corresponding to a *moderate* case with smaller uncertainty and a *stressed* case with larger uncertainty respectively. The results are presented in Table 4.2. For the moderate case, both policies are able to enforce the chance constraints with the full policy marginally out-performing the AGC-type policy in terms of cost. For the stressed case however, the AGC-type policy could not converge to anywhere near the desired $\alpha = 0.10$ even after 20 epochs and saturated at a constraint violation probability of around 26.5%. The full policy on the other hand was able to decrease the probability of constraint violations to 10.5%. As a baseline comparison, the OPF policy was evaluated and found to attain an average cost of \$ 2183.14. The stressed case demonstrates that general control policies can significantly improve cost of control actions and their ability to enforce constraints in real time.

Table 4.2: Test results comparing the full policy with an AGC-type policy for $\alpha = 0.10$, and $E = 20$.

Policy	Maximum violation [%]		Average cost [\$]	
	$R = 0.1$	$R = 0.2$	$R = 0.1$	$R = 0.2$
Full	8.5	10.5	2180.45	2184.67
AGC-type	9.0	26.5	2185.52	2189.23

4.6.4 Scalability

We test the DNN framework on a number of test networks of varying sizes. The values for hyper-parameters, load variation parameter R, and the training times of these networks are compiled in Table 4.3 and Table 4.4 reports the performance of the trained DNNs. The testing times and the average costs attained by the proposed strategy are compared against the OPF policy. For all networks, the DNN policy is consistently able to enforce the chance constraints with the specified α with the costs remaining close to that of the OPF policy indicating that the DNN policy is near-optimal. The training times in Table 4.3 show a moderate increase with size of the test networks indicating that the proposed approach is highly scalable. We remark that the exact training times reported are less indicative than the trend. The exact times are highly dependent on implementation details where parallelized implementations and GPU deployment can drastically improve these numbers. The evaluation times in Table 4.4 are orders of magnitude faster than the OPF policy, making them highly suitable for real-time computation of control actions.

Table 4.3: The values for hyper-parameters $\{\epsilon, E, \mu_0, \nu_0\}$, setting R, and the training times for different networks for $\alpha = 0.05$

Network	ϵ	E	μ_0	ν_0	R	Train time (sec)
case6ww	0.005	5	10^{-3}	$4 \cdot 10^{-2}$	0.05	73.45
case69	0.01	5	10^{-3}	10^{-2}	0.1	79.85
case118	0.07	5	10^{-3}	1	0.01	308.4
case141	0.01	5	10^{-3}	10^{-3}	0.1	104.25

Table 4.4: Test results for the full DNN-based policy (full) and OPF policy (OPF) for different networks for $\alpha = 0.05$.

Network	Maximum violation [%]	Time (sec)		Average cost [$]	
		Prop.	Opt.	Prop.	Opt.
case6ww	5	0.22	24.09	$3.17 \cdot 10^3$	$3.15 \cdot 10^3$
case69	0	0.25	26.35	80.58	80.58
case118	0.5	0.39	61.10	$1.30 \cdot 10^5$	$1.30 \cdot 10^5$
case141	0.0	0.28	40.34	251.89	251.89

4.7 Conclusions

We presented a DNN-based approach for solving the SOPF, where DNNs are used to parameterize the control policies required for real-time power balancing in response to uncertainty. Our approach does not require previously generated training labels and instead used the training phase to solve the SOPF. Stochastic primal-dual updates are employed to learn the DNN weights such that generation costs are minimized while respecting the power system constraints. Numerical tests on a variety of benchmark networks confirm that the generalized policy is able to provide high quality feasible solutions to chance constrained AC-OPF problem over a range of operating conditions, with significant improvements over an AGC-type policy in terms of cost and constraint enforcement. Comparison with the OPF policy where an OPF is solved in response to each uncertainty realization shows that the DNN policy is able to attain similar levels of feasibility and optimality, while facilitating near-instant computation of real-time control actions. In future, we plan to extend our approach to joint chance-constrained OPF problems and research scenario selection policies to aid in faster

DNN training.

Chapter 5

Deep Learning for Optimal Volt/VAR Control using Distributed Energy Resources

5.1 Publication Details

S. Gupta, S. Chatzivasileiadis, and V. Kekatos. "Deep learning for optimal volt/var control using distributed energy resources", *IEEE Trans. on Smart Grid*, (submitted Nov. 2022). URL http://arxiv.org/abs/2211.09557.

5.2 Abstract

Given their intermittency, distributed energy resources (DERs) have been commissioned with regulating voltages at fast timescales. Although the IEEE 1547 standard specifies the shape of Volt/VAR control rules, it is not clear how to optimally customize them per DER. Optimal rule design (ORD) is a challenging problem as Volt/VAR rules introduce nonlinear dynamics, require bilinear optimization models, and lurk trade-offs between stability and steady-state performance. To tackle ORD, we develop a deep neural network (DNN) that

serves as a digital twin of Volt/VAR dynamics. The DNN takes grid conditions as inputs, uses rule parameters as weights, and computes equilibrium voltages as outputs. Thanks to this genuine design, ORD is reformulated as a deep learning task using grid scenarios as training data and aiming at driving the predicted variables being the equilibrium voltages close to unity. The learning task is solved by modifying efficient deep-learning routines to enforce constraints on rule parameters. In the course of DNN-based ORD, we also review and expand on stability conditions and convergence rates for Volt/VAR rules on single-/multi-phase feeders. To benchmark the optimality and runtime of DNN-based ORD, we also devise a novel mixed-integer nonlinear program formulation. Numerical tests showcase the merits of DNN-based ORD.

5.3 Introduction

DERs such as solar photovoltaics, are being advocated as a means to battle climate change, shave peak demand, and improve reliability. Despite the obvious benefits, voltage fluctuations arising from uncertainties in DERs may hinder their large-scale adoption. Fortunately, optimal control strategies for inverter-interfaced DERs to provide voltage regulation can alleviate the previous concern. Under centralized voltage regulation, a supervisory entity consolidates load and generation measurements across the grid and solves an optimal power flow (OPF) to determine setpoints for DERs [24, 66, 105]. Albeit easier to conceive, centralized schemes suffer from considerable communication and computational overhead and privacy issues, and presume a detailed feeder model is available.

On the other hand, *local* voltage regulation schemes for DERs entail calculating control setpoints solely on the basis of data locally accessible by the DERs, such as load, solar generation, and voltage measurements at the grid interface. The IEEE 1547.8 Standard prescribes a local control scheme whereby DER setpoints are produced by *control rules* taking the form of piecewise linear functions of local measurements [3]. Local rules however are known to produce sub-optimal setpoints [12, 65]. Nevertheless, autonomy and simplicity are lucrative features of local schemes for real-time DER control. Focusing on Volt/VAR control, this work delves into the study and optimal design of local rules.

Volt/VAR rules compute reactive power setpoints for inverters based on the local deviation from the nominal voltage. Since voltages are affected by reactive setpoints, Volt/VAR rules give rise to closed-loop dynamics, which can become unstable under steep control rule slopes [26, 107, 108]. While the aforementioned works study the convergence and stability of Volt/VAR control rules, they do not address how to design such rules, i.e., select their exact shape, in the first place. Prior efforts on designing DER rules either resort to heuristics [11, 83, 90], deal with non-dynamic Watt/VAR rules [47], or restrict themselves to affine Volt/VAR rules [7, 84]. Different from above, our recent work in [71] formulates a bilevel optimization to design the slopes, deadband, saturation, and reference voltages for the Volt/VAR rules as prescribed by the IEEE 1547.8 Standard. The bilevel optimization considers a number of possible grid scenarios, and upon leveraging the properties of the system at equilibrium, it finds stationary points using projected gradient descent iterates.

Our present work extends [71], and improves upon the previously cited literature in four directions: *c1)* Most of the existing works focus on simplified single-phase models of distribution feeders and do not capture inter-phase coupling. We extend the analysis in [108] to provide a polytopic restriction on the slope of control curves, which guarantees the stability of Volt/VAR rules per the IEEE 1547 Standard to multiphase feeders; *c2)* We design a digital twin of Volt/VAR dynamics using a DNN. The digital twin accepts grid conditions as the input, the control rule parameters as weights, and computes approximate equilibrium voltages at its output. Based on the rate of convergence of Volt/VAR dynamics, we determine the minimum depth for the DNNs to simulate such dynamics up to the desired level of accuracy; *c3)* We cast the problem of optimal design of Volt/VAR rules as training the DNN-based digital twin. The training process involves stochastic projected gradient updates (SPGD) that leverage efficient, off-the-shelf Python libraries to train the DNNs; *c4)* Exploiting the bilevel structure, we also formulate ORD as a novel mixed-integer nonlinear program (MINLP). Although the MINLP-based approach does not scale well with the number of DERs and grid scenarios, it serves as a benchmark for the optimality and computational speed of DNN-based ORD.

We next expound upon how our work differs from prior works utilizing machine learning for smart inverter control. DNNs have been extensively employed before for optimal DER control under OPF formulations, with the objective of minimizing energy losses and energy costs; see e.g., [34, 86, 98, 104]. Support vector machines and Gaussian processes have also been suggested for reactive power control using smart inverters [48, 49]. However, none of the above references aim at modeling the IEEE 1547-type piecewise linear, local, Volt/VAR rules. Furthermore, the existing problem formulations preclude the presence of closed-loop dynamics and are not nuanced by stability concerns as in the present work. In terms of using a DNN to model piecewise linear Volt/VAR control rules, our work bears some similarities with [16]. Reference [16] models piecewise linear control rules, using a NN with a single hidden layer. However, the DNN-based approach is not extended to capture Volt/VAR dynamics end-to-end and model equilibrium voltages, primarily because its focus is on optimal control of transient dynamics. In contrast, the aim of the present work is to design control rules that produce equilibrium voltages close to unity across many scenarios. Moreover, reference [16] does not discuss other topics covered in this work such as the IEEE 1547-type Volt/VAR rules, their convergence speed, and the implications of Volt/VAR control in multiphase feeders.

This paper is organized as follows. Section 5.4 presents the feeder model. Section 5.5 discusses the equilibrium and convergence of Volt/VAR rules on single-phase feeders. Section 5.6 casts the problem of ORD in single-phase feeders as a deep learning task. Section 5.7 formulates the benchmark MINLP that also solves ORD for single-phase feeders. Section 5.8 extends DNN-based ORD to multiphase feeders, while providing the necessary stability and convergence claims. Section 5.9 presents numerical tests confirming the efficacy of the proposed DNN-based ORD on single- and multi-phase feeders. Conclusions and future directions are drawn in Section 5.10.

5.4 Feeder Modeling Preliminaries

Consider a feeder rooted at the substation. Although the feeder can be single-phase or multiphase, it features a tree structure in terms of buses. For multiphase feeders, a bus may be serving one to three phases; a valid pair of bus and phase will be referred to as a *node*. For single-phase feeders, the terms *bus* and *node* will be used interchangeably. The substation is indexed by 0 and is considered balanced; all remaining nodes are indexed by $n \in \mathcal{N} := \{1, \ldots, N\}$. All DERs are assumed to be single-phase and be able to provide reactive power control. For simplicity, each node is assumed to host a DER; we briefly discuss the minor modifications to deal with the more practical setting where not all nodes host DERs. Our numerical tests evaluate the latter setting.

To study the effect of power injections on voltage magnitudes, we use an approximate linearized grid model. Let the active/reactive power injections and voltage magnitudes (henceforth simply voltages) at the non-substation nodes be collected into the N-length vectors $\mathbf{p}$,

$\mathbf{q}$, and $\mathbf{v}$, respectively. The linearized grid model relates these quantities as [92]

$$\mathbf{v} \simeq \mathbf{R}\mathbf{p} + \mathbf{X}\mathbf{q} + v_0\mathbf{1} \tag{5.1}$$

where v_0 is the substation voltage, and real-valued matrices $\mathbf{R}$ and $\mathbf{X}$ depend on line impedances and feeder topology.

If $\mathbf{p}^g$ and $\mathbf{p}^\ell$ denote the active power generated by DERs and that consumed by the loads accordingly, then $\mathbf{p} = \mathbf{p}^g - \mathbf{p}^\ell$. Reactive power injections can be decomposed similarly as $\mathbf{q} = \mathbf{q}^g - \mathbf{q}^\ell$. Supposing $\mathbf{p}$ and $\mathbf{q}^\ell$ are uncontrolled and vary with time, reactive power compensation entails adjusting $\mathbf{q}^g$ to maintain $\mathbf{v}$ around one per unit (pu). To isolate the effect of DER reactive injections on voltages, rearrange (6.1) as

$$\mathbf{v} = \mathbf{X}\mathbf{q}^g + \tilde{\mathbf{v}} = \mathbf{X}\mathbf{q} + \tilde{\mathbf{v}} \tag{5.2}$$

where the notation is slightly abused by denoting $\mathbf{q}^g$ as $\mathbf{q}$ for simplicity. The uncontrolled quantities are captured in vector $\tilde{\mathbf{v}} := \mathbf{R}(\mathbf{p}^g - \mathbf{p}^\ell) - \mathbf{X}\mathbf{q}^\ell + v_0\mathbf{1}$, where $\tilde{\mathbf{v}}$ models voltages had it not been for reactive power compensation. Vector $\tilde{\mathbf{v}}$ will be henceforth termed the vector of *grid conditions*.

Given its importance in Volt/VAR control, let us summarize some properties of the sensitivity matrix $\mathbf{X}$ appearing in (5.2). For single-phase feeders, matrix $\mathbf{X}$ is known to be symmetric, positive definite, and with positive entries; see e.g., [8], [27]. For multiphase feeders however, matrix $\mathbf{X}$ is non-symmetric and has positive as well as negative entries [53]. Nonetheless, under conditions typically met in practice [53], matrix $\mathbf{X}$ remains positive definite for multiphase feeders in the sense $\mathbf{z}^\top\mathbf{X}\mathbf{z} > 0$ for all $\mathbf{z} \neq \mathbf{0}$. These nuances of $\mathbf{X}$ call for relatively different treatments of Volt/VAR control between single- (Sections 5.5–5.6) and multi-phase feeders (Section 5.8).

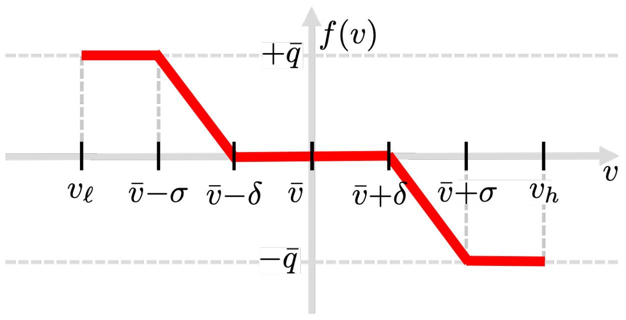


Figure 5.1: The piecewise linear Volt/VAR control rule $f(v)$ provisioned by the IEEE 1547 standard [3]. The x-axis corresponds to the local voltage magnitude and the y-axis to the inverter setpoint for reactive power injection.

5.5 Control Rules for Single-Phase Feeders

The IEEE 1547.8 standard provisions four modes of reactive power control [3]: constant power, constant power factor, Watt/VAR, and Volt/VAR. We focus on the last one as being the most grid-adaptive. This mode enables the inverters to respond to local voltage deviations via a piecewise linear control curve $f(v)$, like the one depicted in Fig. 5.1. The curve consists of a deadband of length 2δ centered around $\bar{v}$; two affine regions; and two regions wherein reactive injections saturate at $\pm\bar{q}$. The standard constraints curve parameters as

$$0.95 \leq \bar{v} \leq 1.05 \tag{5.3a}$$

$$0 \leq \delta \leq 0.03 \tag{5.3b}$$

$$\delta + 0.02 \leq \sigma \leq 0.18 \tag{5.3c}$$

$$0 \leq \bar{q} \leq \hat{q}. \tag{5.3d}$$

Per (5.3d), the saturation value $\bar{q}$ can be equal to the reactive power capability $\hat{q}$ of the inverter, but also smaller than that.

The rule of Fig. 5.1 is parameterized by $(\bar{v}, \delta, \sigma, \bar{q})$, which can be customized per bus n as

$(\bar{v}_n, \delta_n, \sigma_n, \bar{q}_n)$. The rule can be alternatively parameterized by $(\bar{v}_n, \alpha_n, \delta_n, \bar{q}_n)$, where α_n is the negative slope of the affine segment and is defined as

$$\alpha_n = \frac{\bar{q}_n}{\sigma_n - \delta_n} > 0. \tag{5.4}$$

Let vectors $(\bar{\mathbf{v}}, \boldsymbol{\alpha}, \boldsymbol{\delta}, \bar{\mathbf{q}})$ collect $(\bar{v}_n, \alpha_n, \delta_n, \bar{q}_n)$ for all $n \in \mathcal{N}$; and stack such vectors together in vector $\mathbf{z} := (\bar{\mathbf{v}}, \boldsymbol{\alpha}, \boldsymbol{\delta}, \bar{\mathbf{q}})$.

The interaction of Volt/VAR-controlled DERs with the grid results in the non-linear discrete-time dynamics

$$\mathbf{v}^t = \mathbf{X}\mathbf{q}^t + \tilde{\mathbf{v}} \tag{5.5a}$$
$$\mathbf{q}^{t+1} = \mathbf{f}_{\mathbf{z}}(\mathbf{v}^t) \tag{5.5b}$$

where vector function $\mathbf{f}_{\mathbf{z}}(\mathbf{v}^t)$ represents the action of Volt/VAR rules across all nodes and is parameterized by $\mathbf{z}$. If stable, the dynamics in (5.5) enjoy an equilibrium under any grid condition $\tilde{\mathbf{v}}$ [27]. In fact, the inverter setpoints at equilibrium coincide with the unique minimizer of the convex program [27]

$$\mathbf{q}^*(\mathbf{z}, \tilde{\mathbf{v}}) = \arg \min_{-\bar{\mathbf{q}} \leq \mathbf{q}} \quad F(\mathbf{q}) := V(\mathbf{q}) + C(\mathbf{q}). \tag{5.6}$$

The two components of the objective $F(\mathbf{q})$ are defined as

$$V(\mathbf{q}) := \frac{1}{2}\mathbf{q}^\top \mathbf{X}\mathbf{q} + \mathbf{q}^\top(\tilde{\mathbf{v}} - \bar{\mathbf{v}}) \tag{5.7a}$$
$$C(\mathbf{q}) := \sum_{n \in \mathcal{N}} \left(\frac{1}{2\alpha_n} q_n^2 + \delta_n |q_n| \right). \tag{5.7b}$$

Component $V(\mathbf{q})$ can be equivalently expressed as [27]

$$V(\mathbf{q}) = \frac{1}{2}(\mathbf{v} - \bar{\mathbf{v}})^\top \mathbf{X}^{-1}(\mathbf{v} - \bar{\mathbf{v}}) + \text{constants}. \tag{5.8}$$

Because $\mathbf{X} \succ 0$, function $V(\mathbf{q})$ is an ℓ_2-norm of $(\mathbf{v} - \bar{\mathbf{v}})$. Hence, minimizing $V(\mathbf{q})$ aims at bringing voltages close to reference voltages. Nonetheless, problem (5.7) involves also $C(\mathbf{q})$ in its cost. Based on (5.6) and to best regulate voltages, one would try setting $\boldsymbol{\alpha}$ to infinity and $\boldsymbol{\delta}$ to zero so $C(\mathbf{q}) = 0$ and the equilibrium setpoints minimize only $V(\mathbf{q})$. This course of action has a dynamic stability implication as detailed next.

Reference [108] guarantees that Volt/VAR dynamics are stable if $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 < 1$, where $\operatorname{dg}(\boldsymbol{\alpha})$ is a diagonal matrix having $\boldsymbol{\alpha}$ on its diagonal. To be satisfied as a strict inequality, the condition can be strengthened as $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 \leq 1 - \epsilon$ for some $\epsilon \in (0, 1)$.

Definition 5.1. Volt/VAR rules satisfying $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 \leq 1 - \epsilon$ for $\epsilon \in (0, 1)$ will be henceforth termed ϵ-stable.

To avoid the spectral norm condition $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 \leq 1 - \epsilon$, we have previously proposed a polytopic restriction [71]:

$$\mathbf{X}\boldsymbol{\alpha} \leq (1 - \epsilon)\mathbf{1} \tag{5.9a}$$

$$\alpha_n \leq \frac{1 - \epsilon}{\sum_{m \in \mathcal{N}} X_{nm}}, \quad \forall n \in \mathcal{N}. \tag{5.9b}$$

The next section develops methods for selecting the Volt/VAR rule parameters $\mathbf{z}$ so that a voltage regulation objective is minimized for a set of grid scenarios. For single-phase feeders, Section 5.6 reformulates (ORD) as the problem of training a neural network, while Section 5.7 tackles ORD as a mixed-integer nonlinear program. For multiphase feeders, solving ORD is dealt with in Section 5.8.

5.6 ORD for 1ϕ Feeders via Deep Learning

Because Volt/VAR rules are used so inverters can operate autonomously, it is reasonable to assume that rule parameters $\mathbf{z}$ are updated infrequently, say every 2 hours. Then, rules $\mathbf{z}$ should be optimized while considering the possibly diverse loading conditions the feeder may experience over those 2 hours. To account for such conditions, suppose we are given a set of S load/solar scenarios $\{(\mathbf{p}_s^g, \mathbf{p}_s^\ell, \mathbf{q}_s^\ell)\}_{s=1}^S$. Each scenario is related to grid condition vector [see (5.1)]

$$\tilde{\mathbf{v}}_s := \mathbf{R}(\mathbf{p}_s^g - \mathbf{p}_s^\ell) - \mathbf{X}\mathbf{q}_s^\ell.$$

Let $\mathbf{q}^*(\mathbf{z}, \tilde{\mathbf{v}}_s)$ or simply $\mathbf{q}_s^*(\mathbf{z})$ denote the equilibrium setpoints reached by stable Volt/VAR rules parameterized by $\mathbf{z}$ under grid conditions $\tilde{\mathbf{v}}_s$. Unfortunately, setpoints $\mathbf{q}_s^*(\mathbf{z})$ cannot be expressed as in closed form. They can be computed by either iterating (5.5), or as the minimizer of (5.6). The related equilibrium voltage is $\mathbf{v}_s^*(\mathbf{z}) := \mathbf{X}\mathbf{q}_s^*(\mathbf{z}) + \tilde{\mathbf{v}}_s$ from (5.1).

We pose the ORD task as a minimization problem over $\mathbf{z}$:

$$\begin{aligned} \min_{\mathbf{z}} \quad & \frac{1}{2S}\sum_{s=1}^{S} \|\mathbf{X}\mathbf{q}_s^*(\mathbf{z}) + \tilde{\mathbf{v}}_s - \mathbf{1}\|_2^2 \\ \text{s.to} \quad & (5.3), (5.9) \end{aligned} \tag{5.10}$$

to minimize the Euclidean distance of equilibrium voltages from unity, averaged across scenarios. Constraints (5.3) and (5.9) ensure rules are stable and compliant with the IEEE 1547.

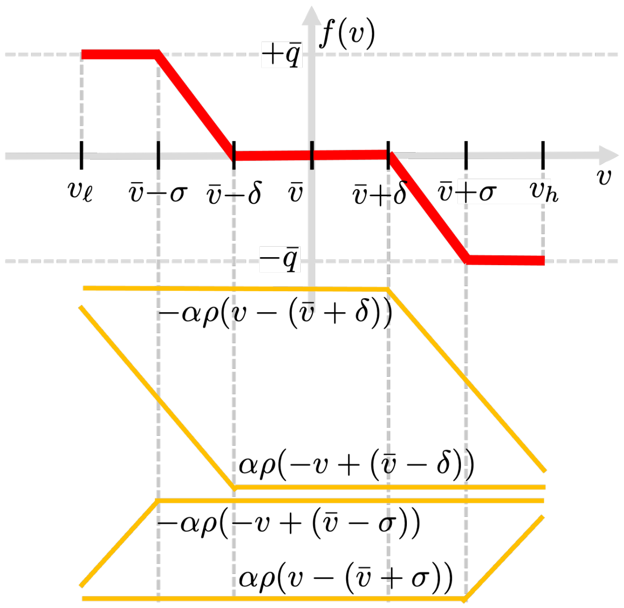
One may wonder why are we not satisfied with the fact that any stable rule $\mathbf{z}$ settles at the minimizer of (5.6), which is seemingly a meaningful equilibrium. Such equilibrium may be insufficient due to three reasons: *i)* The term $V(\mathbf{q})$ is a *rotated* ℓ_2-norm of $(\mathbf{v} - \bar{\mathbf{v}})$, so that voltage deviations are weighted unequally across buses; *ii)* If DERs are sited only on a subset $\mathcal{G} \subset \mathcal{N}$ of nodes, the cost $V(\mathbf{q})$ gets modified as $V_{\mathcal{G}}(\mathbf{q}_{\mathcal{G}}) = \frac{1}{2}(\mathbf{v}_{\mathcal{G}} - \bar{\mathbf{v}}_{\mathcal{G}})^\top \mathbf{X}_{\mathcal{G}\mathcal{G}}^{-1}(\mathbf{v}_{\mathcal{G}} - \bar{\mathbf{v}}_{\mathcal{G}})$, where subscript $\mathcal{G}$ denotes the subvectors/submatrix obtained by keeping the rows/columns corresponding to $\mathcal{G}$; see [71]. Such cost may not be representative of $\|\mathbf{v} - \bar{\mathbf{v}}\|_2^2$; and *iii)* As discussed earlier, stability limitations do not allow us to set $\boldsymbol{\alpha}$ to infinity although it seems desirable from a voltage regulation standpoint. The aforesaid reasons motivate the need to optimally design $\mathbf{z}$ so the induced equilibrium voltages $\mathbf{v}_s^*(\mathbf{z})$ are better regulated.

Albeit simply stated, problem (5.10) is computationally challenging as $\mathbf{q}_s^*(\mathbf{z})$ is the solution of the *inner* minimization problem (5.6), which is parameterized by $\mathbf{z}$. Thereby, the ORD task is a *bilevel optimization* over $\mathbf{z}$: The *outer* problem (5.10) depends on S inner problems of the form (5.6), one per scenario.

Our first strategy towards tackling (5.10) is to replace the inner problem with a DNN that simulates the Volt/VAR dynamics. This DNN has $\mathbf{z}$ as weights, accepts $\tilde{\mathbf{v}}_s$ as input, and outputs the equilibrium voltages $\mathbf{v}_s^*(\mathbf{z})$. Let the DNN output be denoted by $\Phi(\tilde{\mathbf{v}}_s; \mathbf{z})$. The key idea is that if $\Phi(\tilde{\mathbf{v}}_s; \mathbf{z})$ are the equilibrium voltages for rule $\mathbf{z}$ over scenario s, then problem (5.10) becomes the supervised training task:

$$\begin{aligned} \min_{\mathbf{z}} \quad & L(\mathbf{z}) := \frac{1}{2S} \sum_{s=1}^{S} \|\Phi(\tilde{\mathbf{v}}_s; \mathbf{z}) - \mathbf{1}\|_2^2 \\ \text{s.to} \quad & (5.3), (5.9). \end{aligned} \tag{5.11}$$

To draw a useful analogy, scenarios $\tilde{\mathbf{v}}_s$ are analogous to feature vectors in regression problems; equilibrium voltages $\mathbf{v}_s^* = \Phi(\tilde{\mathbf{v}}_s; \mathbf{z})$ are the predictions for feature vectors; and $\mathbf{1}$ is the (constant) target label for the prediction. Formulating (5.10) as (5.11) allows us to leverage efficient DNN libraries for optimizing $\mathbf{z}$. With this motivation in mind, we next design the DNN, and then describe the steps to train it.



$f(v)$
$+\bar{q}$
$-\bar{q}$
v
v_ℓ
$\bar{v}-\sigma$
$\bar{v}-\delta$
$\bar{v}$
$\bar{v}+\delta$
$\bar{v}+\sigma$
v_h
$-\alpha\rho(v-(\bar{v}+\delta))$
$\alpha\rho(-v+(\bar{v}-\delta))$
$-\alpha\rho(-v+(\bar{v}-\sigma))$
$\alpha\rho(v-(\bar{v}+\sigma))$

Figure 5.2: Volt/VAR rule $f(v)$ expressed as a sum of ReLUs.

5.6.1 Designing a Digital Twin of Volt/VAR Dynamics

The Volt/VAR curve of Fig. 5.1 can be interpreted as a superposition of four piecewise-linear functions, each with a single breakpoint, as shown in Fig. 5.2. These functions can be thought of as the outputs of rectified linear units (ReLU) $\rho(x)$, which return x for $x > 0$; and 0 otherwise. To get the different breakpoints and slopes as in Figure 5.2, the ReLU units need the appropriate inputs and scaling. The required mathematical operations can be implemented through the DNN of Fig. 5.3, which takes v_n^t as input and computes the setpoint q_n^{t+1} at its output. The input and output layers have one neuron each. The hidden layer consists of four neurons. The weights of the hidden layer are fixed to $[1, 1, -1, -1]^\top$, but its bias vector is trainable and given by $[-(\bar{v} + \delta), -(\bar{v} + \sigma), \bar{v} - \delta, \bar{v} - \sigma]^\top$. Each of the four neurons in the hidden layer is equipped with a ReLU unit. The output layer has a trainable weight vector $[-\alpha, \alpha, \alpha, -\alpha]^\top$ and the bias is fixed at 0.

Heed that the NN of Fig. 5.3 implements the Volt/VAR curve for a single inverter and a

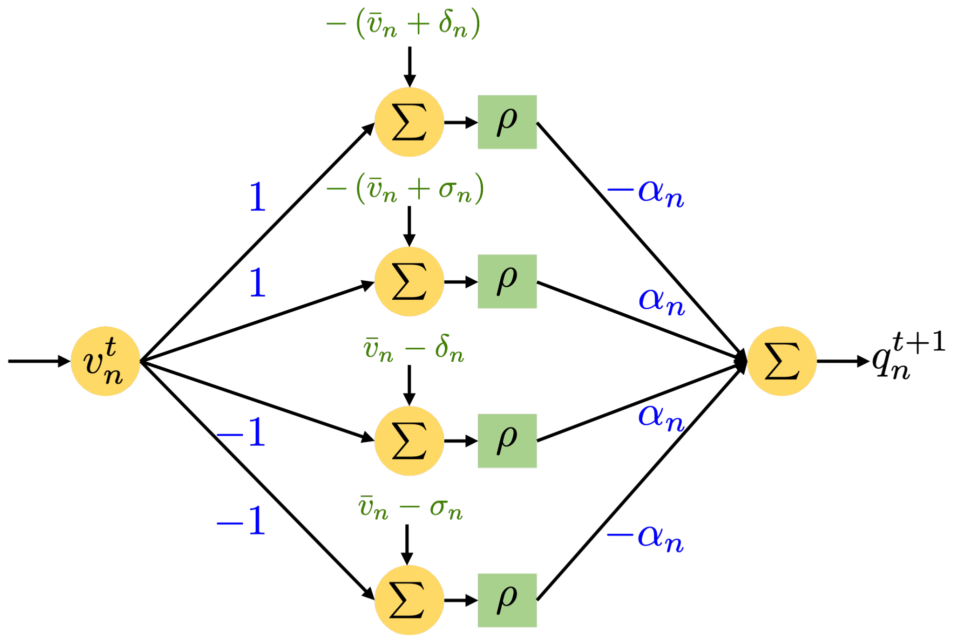


Figure 5.3: Volt/VAR rule $f(v)$ model using a DNN with 1 hidden layer.

single time step as $q_n^{t+1} = f_n(v_n^t)$. To simulate the entire Volt/VAR network dynamics of (5.5), we will treat the NN of Fig. 5.3 as a building block and replicate it across inverters and time. Let VC_n represent the NN module running one time step for inverter n. This module is parameterized by $(\bar{v}_n, \alpha_n, \sigma_n, \delta_n)$. With a slight abuse of terminology, let the collection of VC_n's for all inverters be labeled as a single *ch5:layer*. These modules are stacked vertically as shown in Fig. 5.4. Each one of these layers implements (5.5b) by receiving $\mathbf{v}^t$ as input, and

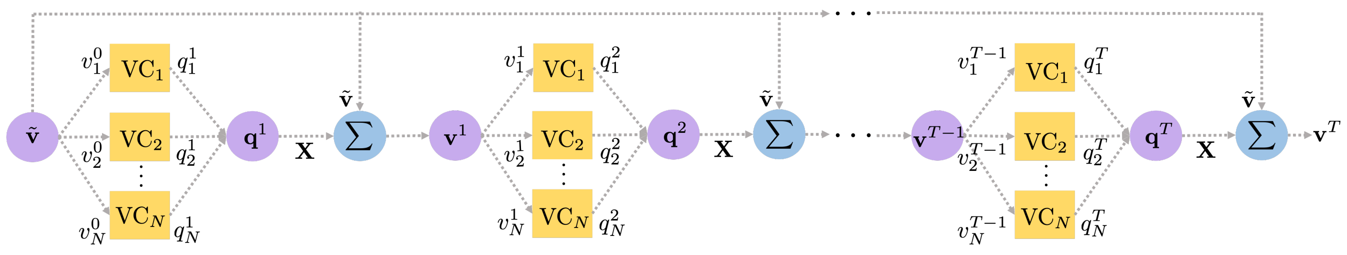


Figure 5.4: DNN-based digital twin for the Volt/VAR dynamics of (5.5). The DNN is structured so that T time steps are arranged horizontally. The modules VC_n's implementing the Volt/VAR curves for each one of the N inverters are stacked vertically. Skip connections propagate the input vector (grid scenario) $\tilde{\mathbf{v}}$ to each time instant to implement $\mathbf{v}^{t+1} = \mathbf{X}\mathbf{q}^{t+1} + \tilde{\mathbf{v}}$.

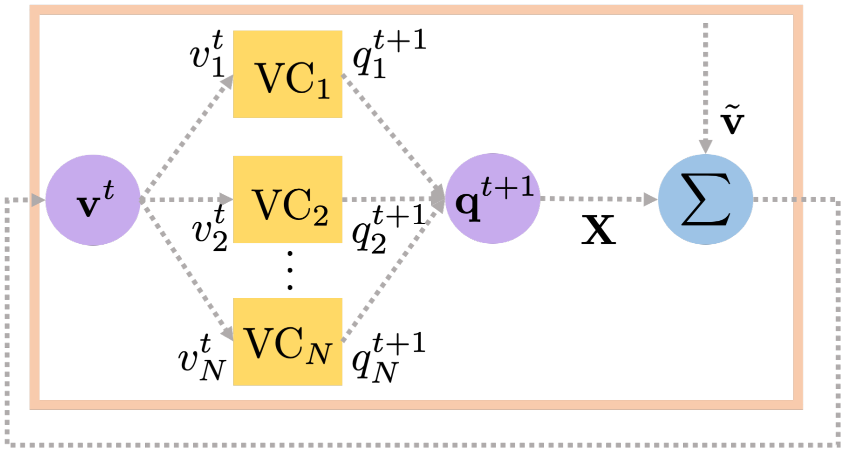


Figure 5.5: Recurrent representation (RNN) of the digital twin of Fig. 5.4.

producing setpoints $\mathbf{q}^{t+1}$ as outputs for a single time t. The setpoints $\mathbf{q}^{t+1}$ in turn produce voltages $\mathbf{v}^{t+1} = \mathbf{X}\mathbf{q}^{t+1} + \tilde{\mathbf{v}}$ per the grid model (5.5a). To simulate the dynamics over time, the new voltages $\mathbf{v}^{t+1}$ are passed to the next layer, and the process is repeated for T steps. Structurally, these interactions result in a larger DNN with T repeating layers, one layer per iteration of (5.5), as shown in Figure 5.4. The simulation of dynamics over T iterations is then simply equivalent to performing a forward pass through the larger DNN with $\tilde{\mathbf{v}}$ as the input. To implement (5.5a), the input $\tilde{\mathbf{v}}$ (grid scenario vector) is also propagated to the inner layers via so termed *skip connections*.

It is worth stressing that each module VC$_n$ is replicated horizontally across the T times. This implies significant *weight sharing* across the T layers. Therefore, the number of trainable parameters $\hat{\mathbf{z}} := (\bar{\mathbf{v}}, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\sigma})$ remains fixed at $4N$, irrespective of the DNN depth T. This weight-sharing aspect results in computational and memory-related efficiencies for DNN storing, prediction, and training, and has been instrumental in the success of architectures such as recurrent (RNN), convolutional (CNN), or graph (GNN) neural networks. In fact, it is possible to obtain a recurrent *'rolled'* representation of the larger DNN of Fig. 5.4, as shown in Fig. 5.5, allowing one to utilize RNN-specific functionalities in DNN libraries.

In a nutshell, the DNN of Fig. 5.4 simulates Volt/VAR dynamics across T times. In other words, once fed with a grid condition vector $\tilde{\mathbf{v}}_s$, its output will approximate the equilibrium voltages $\mathbf{v}_s^*$ reached by Volt/VAR dynamics under rule $\mathbf{z}$. As a result, optimizing over $\mathbf{z}$ by training the DNN so that equilibrium voltages $\{\mathbf{v}_s^*(\mathbf{z})\}_{s=1}^S$ come close to one pu, serves the purposes of ORD. Surrogating Volt/VAR dynamics by the DNN is effective only if the DNN depth T is sufficiently large. *How deep should the DNN be so that its output $\Phi\left(\tilde{\mathbf{v}}_s;\hat{\mathbf{z}}\right)$ is close to $\mathbf{v}_s^*$?* Because a DNN of depth T simulates exactly the Volt/VAR dynamics up to time T, the answer for selecting T is apparently the settling time of the Volt/VAR dynamics as detailed next and shown in the appendix.

Proposition 5.2. *Suppose ϵ-stable Volt/VAR rules are described by $\mathbf{z}$. The depth T of the DNN in Fig. 5.4 required to ensure $\|\Phi\left(\tilde{\mathbf{v}};\mathbf{z}\right)-\mathbf{v}^*(\mathbf{z})\|_2 \leq \epsilon_1$ for all grid conditions $\tilde{\mathbf{v}}$ is*

$$T \geq \frac{\log \frac{2\|\mathbf{X}\|_2\|\hat{\mathbf{q}}\|_2}{\epsilon_1}}{\log(1-\epsilon)^{-1}}.$$

The result implies that the minimum depth T grows logarithmically with the desired accuracy ϵ_1 and the stability margin ϵ. Plugging in the typical values $\epsilon_1 = 10^{-4}$, $\|\mathbf{X}\|_2 = 4.63\cdot 10^{-1}$ for IEEE 37-bus feeder, $\|\hat{\mathbf{q}}\|_2 = 0.1$, and $\epsilon = 0.3$, the bound yields a comfortably small number of $T \geq 20$ layers. For $\epsilon_1 = 10^{-6}$, the number of layers T increases to 32, demonstrating the scalability of the approach.

5.6.2 DNN Training

With rule parameters $\hat{\mathbf{z}}$ embedded as DNN weights and biases, the optimal Volt/VAR curves are obtained by training $\Phi\left(\tilde{\mathbf{v}}_s;\hat{\mathbf{z}}\right)$. Conventional DNN training uses stochastic gradient descent (SGD) to update the DNN parameters and eventually minimize the *loss function* in (5.11). However, parameters $\hat{\mathbf{z}}$ should satisfy constraints (5.3) and (5.9). Plain SGD may fail to return a feasible $\mathbf{z}$. This can be circumvented by using projected stochastic gradient (PSGD) updates. PSGD updates first compute an intermediate quantity $\hat{\mathbf{x}}^{i+1}$ via gradient descent

$$\hat{\mathbf{x}}^{i+1} = \hat{\mathbf{z}}^i - \frac{\mu}{2B}\nabla_{\hat{\mathbf{z}}^i}\left(\sum_{s\in\mathcal{B}_i}\|\Phi(\tilde{\mathbf{v}}_s;\hat{\mathbf{z}}) - \mathbf{1}\|_2^2\right) \tag{5.12}$$

where $\mu > 0$ is the step size; set $\mathcal{B}_i$ is a batch of B scenarios (a subset of the original S scenarios); and $\nabla_{\hat{\mathbf{z}}^i}(\cdot)$ is the gradient of the loss function with respect to $\hat{\mathbf{z}}$ evaluated at $\hat{\mathbf{z}} = \hat{\mathbf{z}}^i$. The gradient term in (5.12) is calculated efficiently thanks to *gradient back-propagation*.

The second step for PSGD updates entails projecting $\hat{\mathbf{x}}^{i+1}$ into the feasible space defined by (5.3) and (5.9). To this end, we first transform $\hat{\mathbf{x}}^{i+1}$ from parameter space $(\bar{\mathbf{v}}, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\sigma})$ to space $(\bar{\mathbf{v}}, \mathbf{c}, \boldsymbol{\delta}, \boldsymbol{\sigma})$, where vector $\mathbf{c}$ has entries $c_n := 1/\alpha_n$. Variable $\hat{\mathbf{x}}^{i+1}$ transformed in the new space is called $\tilde{\mathbf{x}}^{i+1}$. The transformation is a one-to-one mapping between the two spaces, and is used so that the feasibility set induced by (5.3) and (5.9) is convex, and so it is easy to project onto it. We proposed this transformation in [71]. We review it here for completeness. Using (5.4), constraint (5.3d) is expressed as

$$0 \leq \boldsymbol{\sigma} - \boldsymbol{\delta} \leq \mathbf{c} \odot \hat{\mathbf{q}} \tag{5.13}$$

where $\odot$ means element-wise multiplication. Constraints (5.9) can be expressed in terms of $\mathbf{c}$ instead of $\boldsymbol{\alpha}$ as [71]

$$\mathbf{c} \geq \frac{1}{1-\epsilon}\mathbf{X1} \tag{5.14a}$$

$$\mathbf{Xa} \leq (1-\epsilon)\cdot\mathbf{1} \tag{5.14b}$$

$$\mathbf{a} \odot \mathbf{c} \geq \mathbf{1}, \quad \forall\, n \in \mathcal{N} \tag{5.14c}$$

where $\mathbf{a}$ is an auxiliary variable. Constraint (5.14c) can be rewritten as a second-order cone. In [71], we show how (5.14) is equivalent to (5.9). The quantity $\tilde{\mathbf{x}}^{i+1}$ can now be projected onto the feasible space via the convex minimization

$$\begin{aligned} \tilde{\mathbf{z}}^{i+1} = \arg\min_{\mathbf{z}} \quad & \|\tilde{\mathbf{x}}^{i+1} - \mathbf{z}\|_2^2 \\ \text{s.to} \quad & (5.3a) - (5.3c), (5.13), (5.14). \end{aligned} \tag{5.15}$$

The PSGD update is completed by transforming $\tilde{\mathbf{z}}^{i+1}$ from space $(\bar{\mathbf{v}}, \mathbf{c}, \boldsymbol{\delta}, \boldsymbol{\sigma})$ back to space $(\bar{\mathbf{v}}, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\sigma})$ to get $\hat{\mathbf{z}}^{i+1}$.

The proposed DNN training can be implemented in Python using DNN libraries such as PyTorch [1]. Step (5.12) is the standard SGD update pertaining to the loss function of (5.11) over the batch of training labels $\{\tilde{\mathbf{v}}_s, 1\}_{\mathcal{B}_i}$. As with standard DNN training, adaptive moment-based algorithms such as Adam can be used to enable fast convergence and avoid saddle points. The DNN weights and biases are transformed between the parameter spaces, and then passed to a convex optimization module to implement the projection step of (5.15). In the last step, DNN weights and biases are updated with the new projected parameters, upon transformation to the original space. The steps are repeated for several epochs.

5.7 ORD for 1ϕ Feeders as an MINLP

A second approach towards solving the bilevel program in (5.10) is to replace each inner problem by its first-order optimality conditions and append these conditions as constraints to the outer problem. To capture complementary slackness, we will introduce binary variables and use the so-termed big-M trick to eventually express the outer problem as a mixed-integer nonlinear program (MINLP). The process is delineated next. Although this MINLP approach does not scale gracefully with the number of DERs and/or scenarios, it serves as a benchmark for the DNN-based ORD.

We first transform (5.6) to a differentiable form as

$$\begin{aligned}
\min_{\mathbf{q},\mathbf{w}} \quad & \frac{1}{2}\mathbf{q}^\top(\mathbf{X}+\mathrm{dg}(\mathbf{c}))\,\mathbf{q}+\mathbf{q}^\top(\tilde{\mathbf{v}}_s-\bar{\mathbf{v}})+\boldsymbol{\delta}^\top\mathbf{w} && (5.16a)\\
\text{s.to} \quad & -\mathbf{w}\leq\mathbf{q}\leq\mathbf{w}: \qquad (\underline{\boldsymbol{\lambda}},\overline{\boldsymbol{\lambda}}) && (5.16b)\\
& -\bar{\mathbf{q}}\leq\mathbf{q}\leq\bar{\mathbf{q}}: \qquad (\underline{\boldsymbol{\mu}},\overline{\boldsymbol{\mu}}) && (5.16c)
\end{aligned}$$

where vector $\mathbf{c}$ has entries $c_n := 1/\alpha_n$, and variable $\mathbf{w}$ has been introduced to deal with the non-differentiable terms $|q_n|$ in (5.6). Slightly abusing notation, denote the optimal primal/-dual variables of (5.16) by $(\mathbf{q},\mathbf{w};\underline{\boldsymbol{\lambda}},\overline{\boldsymbol{\lambda}},\underline{\boldsymbol{\mu}},\overline{\boldsymbol{\mu}})$. Although the variables vary per scenario, we suppress subscript s for simplicity. These variables satisfy the optimality conditions

$$\begin{aligned}
(\mathbf{X}+\mathrm{dg}(\mathbf{c}))\,\mathbf{q}+\tilde{\mathbf{v}}_s-\bar{\mathbf{v}}-\underline{\boldsymbol{\lambda}}+\overline{\boldsymbol{\lambda}}-\underline{\boldsymbol{\mu}}+\overline{\boldsymbol{\mu}}&=\mathbf{0} && (5.17a)\\
\boldsymbol{\delta}-\underline{\boldsymbol{\lambda}}-\overline{\boldsymbol{\lambda}}&=\mathbf{0} && (5.17b)\\
-\mathbf{w}\leq\mathbf{q}&\leq\mathbf{w} && (5.17c)\\
-\bar{\mathbf{q}}\leq\mathbf{q}&\leq\bar{\mathbf{q}} && (5.17d)\\
\underline{\boldsymbol{\lambda}},\overline{\boldsymbol{\lambda}},\underline{\boldsymbol{\mu}},\overline{\boldsymbol{\mu}}&\geq\mathbf{0} && (5.17e)\\
\overline{\boldsymbol{\lambda}}\odot(\mathbf{q}-\mathbf{w})&=\mathbf{0} && (5.17f)\\
\underline{\boldsymbol{\lambda}}\odot(-\mathbf{q}-\mathbf{w})&=\mathbf{0} && (5.17g)\\
\overline{\boldsymbol{\mu}}\odot(\mathbf{q}-\bar{\mathbf{q}})&=\mathbf{0} && (5.17h)\\
\underline{\boldsymbol{\mu}}\odot(-\mathbf{q}-\bar{\mathbf{q}})&=\mathbf{0}. && (5.17i)
\end{aligned}$$

Equalities (5.17a)–(5.17b) follow from Lagrangian optimality; and inequalities (5.17c)–(5.17e) from primal/dual feasibility. Inequalities (5.17f)–(5.17i) are complementary slackness conditions.

The bilevel problem in (5.10) can be now reduced to a single-level formulation upon appending conditions (5.17) as constraints to (5.10) per scenario s. Such constraints essentially ensure that $\mathbf{q}_s^*(\mathbf{z})$ is indeed the minimizer of (5.6). Nonetheless, constraint (5.17a) and the complementary conditions introduce bilinear terms. Bilinearity can be partially addressed by handling complementary slackness conditions through the big-M trick. For example, condition (5.17f) can be expressed as

$$\mathbf{0} \leq \overline{\boldsymbol{\lambda}} \leq M_1 \mathbf{b} \tag{5.18a}$$

$$\mathbf{0} \leq \mathbf{q} - \mathbf{w} \leq M_2(\mathbf{1} - \mathbf{b}) \tag{5.18b}$$

where $\mathbf{b}$ is an N-dimensional binary variable, and (M_1, M_2) are large positive constants. The latter can be selected as $M_2 = 2\bar{\mathbf{q}}$, while the former can be set to a numerically estimated upper bound of the corresponding dual variables $\underline{\boldsymbol{\lambda}}$.

Since (5.17)–(5.18) contain $\mathbf{c}$ and $\bar{\mathbf{q}}$, the constraints (5.3) and (5.9) need to be rewritten in terms of $\mathbf{c}$ and $\bar{\mathbf{q}}$ as well. To this end, we chose the parameterization $\tilde{\mathbf{z}} := (\bar{\mathbf{v}}, \mathbf{c}, \boldsymbol{\delta}, \bar{\mathbf{q}})$. In this new parameterization, constraint (5.3c) is replaced by

$$0.02 \cdot \mathbf{1} \leq \mathbf{c} \odot \bar{\mathbf{q}} \leq 0.18 \cdot \mathbf{1} - \boldsymbol{\delta} \tag{5.19}$$

which introduces bilinear terms too. Stability constraints (5.9) have already been transformed from $\boldsymbol{\alpha}$ to $\mathbf{c}$ in (5.14).

Putting everything together, the bilevel ORD problem of (5.10) can be solved as the MINLP:

$$\begin{aligned}
\tilde{\mathbf{z}}^* = \arg\min_{\tilde{\mathbf{z}}} \quad & \frac{1}{2S}\sum_{s=1}^{S}\|\mathbf{X}\mathbf{q}_s + \tilde{\mathbf{v}}_s - \mathbf{1}\|_2^2 && && \text{(5.20a)}\\
\text{over} \quad & \tilde{\mathbf{z}} := (\bar{\mathbf{v}}, \mathbf{c}, \boldsymbol{\delta}, \bar{\mathbf{q}}) && && \text{(5.20b)}\\
\text{s.to} \quad & \text{(5.3a)}, \text{(5.3b)}, \text{(5.3d)}, \text{(5.14)}, \text{(5.19)} && && \text{(5.20c)}\\
& \text{(5.17a)} - \text{(5.17e)} && \forall\ s && \text{(5.20d)}\\
& \text{(5.17f)} - \text{(5.17i) as in (5.18)} && \forall\ s. && \text{(5.20e)}
\end{aligned}$$

The bilinear terms in (5.17a) and (5.19), and the binary variables in (5.20e) increase with the number of inverters and scenarios.

Remark 5.3. The Volt/VAR curve of Fig. 5.1 has four degrees of freedom that control the center, deadband, slope, and saturation of the curve. These degrees of freedom are amenable to different equivalent parameterizations, such as $(\bar{\mathbf{v}}, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\sigma})$ and $(\bar{\mathbf{v}}, \mathbf{c}, \boldsymbol{\delta}, \boldsymbol{\sigma})$ that we used in Sec. 5.6; $(\bar{\mathbf{v}}, \boldsymbol{\alpha}, \boldsymbol{\delta}, \bar{\mathbf{q}})$; or $(\bar{\mathbf{v}}, \mathbf{c}, \boldsymbol{\delta}, \bar{\mathbf{q}})$. We used the last one in (5.20b) as it yielded significantly shorter solution times during our tests.

Although the MINLP approach can solve the ORD task to near-global optimality (modulo the bilinear terms left to be handled internally by the solver), it was found to scale unfavorably with the number of DERs and/or scenarios of Section 5.9.

5.8 ORD for 3ϕ Feeders via Deep Learning

Under transposed lines and balanced injections, one could deal with ORD using the single-phase formulations discussed earlier. Under imbalance conditions however, a linearized multiphase feeder model would be a better approximation. DERs would still implement local

Volt/VAR rules, yet sensitivity matrix $\mathbf{X}$ now has different properties as discussed in (5.1). For the multiphase case, we were not able to come up with an optimization problem whose minimizer coincides with the equilibrium setpoints $\mathbf{q}^*$ similar to (5.6). Nonetheless, we show in the appendix that the Volt/VAR rules of Fig. 5.1 do converge to a unique equilibrium under the following polytopic conditions, which form a restriction of $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 \leq 1-\epsilon$.

Proposition 5.4. *Consider the Volt/VAR dynamics of* (5.5) *operating over a multiphase feeder modeled by* (5.1)*. If the Volt/VAR slope vector* $\boldsymbol{\alpha}$ *satisfies*

$$|\mathbf{X}|^\top \boldsymbol{\alpha} \leq (1-\epsilon)\cdot \mathbf{1} \tag{5.21a}$$

$$\alpha_n \leq \frac{1-\epsilon}{\sum_{m\in\mathcal{N}} |X_{nm}|}, \quad \forall n \in \mathcal{N}. \tag{5.21b}$$

for $\epsilon \in (0,1)$*, the dynamics in* (5.5) *exhibit a unique equilibrium* $\mathbf{q}^*$ *to which they converge exponentially fast as*

$$\|\mathbf{q}^t - \mathbf{q}^*\|_2 \leq 2\|\hat{\mathbf{q}}\|_2 \cdot (1-\epsilon)^t. \tag{5.21c}$$

The absolute value $|\mathbf{X}|$ applies entry-wise. The result generalizes (5.9) and [108, Th. 3] to multiphase feeders, wherein $\mathbf{X}$ is non-symmetric and with some of its entries being negative. It provides linear constraints on $\boldsymbol{\alpha}$ to ensure stability.

The ORD task for multiphase feeders can be formulated as in (5.6) with the appropriate modification of the sensitivity matrix $\mathbf{X}$. Since equilibrium setpoints cannot be expressed as the minimizer of an inner optimization, the MINLP approach of Section 5.7 cannot be adopted here. Alternatively, one may pursue an MINLP formulation along the lines of [89], though scalability is still expected to be an issue. Fortunately, the DNN-based approach for ORD remains applicable with the next minor modifications: *i)* Sensitivity matrices are modified accordingly; *ii)* Every layer now consists of $3N$ building modules corresponding to bus/phase (node) combinations; and *iii)* Use the stability constraints of (5.21) instead of (5.9).

Proposition 5.2 on minimum depth T of DNNs for Volt/VAR rules in single-phase feeders carries over to multiphase feeders. This is easily confirmed by applying the steps from the proof of Proposition 5.2 to the results from Proposition 5.4.

5.9 Numerical Tests

The proposed ORD methods were evaluated on single- and multi-phase feeders. Real-world data of active load and solar generation at one-minute frequency was sourced from the Smart* project on April 2, 2011 [14]. The set consists of active loads from 444 homes and generation from 43 solar panels. Loads from multiple homes were averaged to better simulate loads at buses of the primary distribution network. Each averaged load was normalized so its peak value during the day coincided with the nominal active power load of its hosting node. For each time interval, reactive loads were added by randomly sampling lagging power factors within $[0.9, 1]$. Similarly to loads, each solar generation signal was normalized so its peak value was twice that of the nominal active load of the hosting bus. Apparent power limits for inverters were set to 1.1 times the peak active generation.

The control rules were designed and evaluated in Python on a 2.4 GHz 8-Core Intel Core i9 processor laptop computer with 64 GB RAM. Pytorch was selected as the library to design and train DNNs, as it implements computation graphs dynamically [1]. That is quite important for our purposes, as dynamic computation graphs imply that the number of layers T does not need to be fixed beforehand. It is rather decided on the fly based on the convergence of rules for the given $\tilde{\mathbf{v}}_t$. This flexibility enables limiting the DNN to lower depths. Convergence was determined based on the change in objective value with the addition of a layer. Specifically, the rules were assumed to have converged if the objective changed by less than $1 \cdot 10^{-7}$ within consecutive layers. All DNNs were trained using the Adam optimizer.

The projection step (5.15) was performed by solving a SOCP using the CVXPY library in Python with GUROBI as the solver. The MINLP (5.20) was implemented in MATLAB using YALMIP [60] with GUROBI, and used to benchmark the results for optimality and running time. Other specific details such as learning rates for DNN training, initialization of design parameters, load and solar panel assignments, and time period for scenario sampling are presented along with the corresponding results.

5.9.1 Tests on Single-Phase Feeder

The first set of tests was conducted on the single-phase equivalent of the IEEE 37-bus feeder. Homes with IDs 20-369 were averaged 10 at a time and successively added as active loads to buses $2-26$ as shown in Fig. 5.6. Active generation from solar panels was also added, as per the mapping in Fig. 5.6. Additionally, buses $\{6, 9, 11, 12, 15, 16, 20, 22, 24, 25\}$ were equipped with DERs capable of reactive power control.

The DNN-based rules were optimized using 80 grid scenarios sampled from the high-solar period $3:00-3:20$ pm, and were trained with the learning rate of 0.003 over 200 epochs. The design parameters $\mathbf{z} := (\bar{\mathbf{v}}, \boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$ were initialized at the feasible point $(\bar{v}_n, \delta_n, \sigma_n, \alpha_n) = (0.95, 0.1, 0.3, 1.5)$ for all n. Figure 5.7 shows the convergence of Volt/VAR rule parameters for DERs at nodes $\{12, 22, 29\}$, for $\epsilon = 0.5$, during training. To accommodate different ranges of magnitudes, all plots are normalized with respect to their initial values.

Figure 5.8 highlights the efficacy of the optimized Volt/VAR rules in improving the voltage

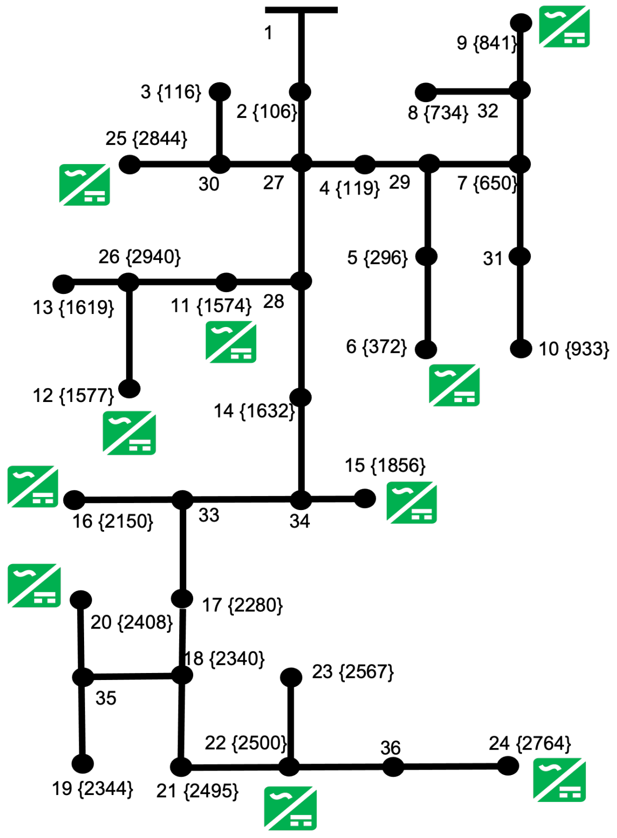


Figure 5.6: The IEEE 37-bus feeder used for the tests. Node numbering follows the format `node number {panel ID}`. DERs at nodes $\{6, 9, 11, 12, 15, 16, 20, 22, 24, 25\}$ provide reactive power control; the rest operate at unit power factor.

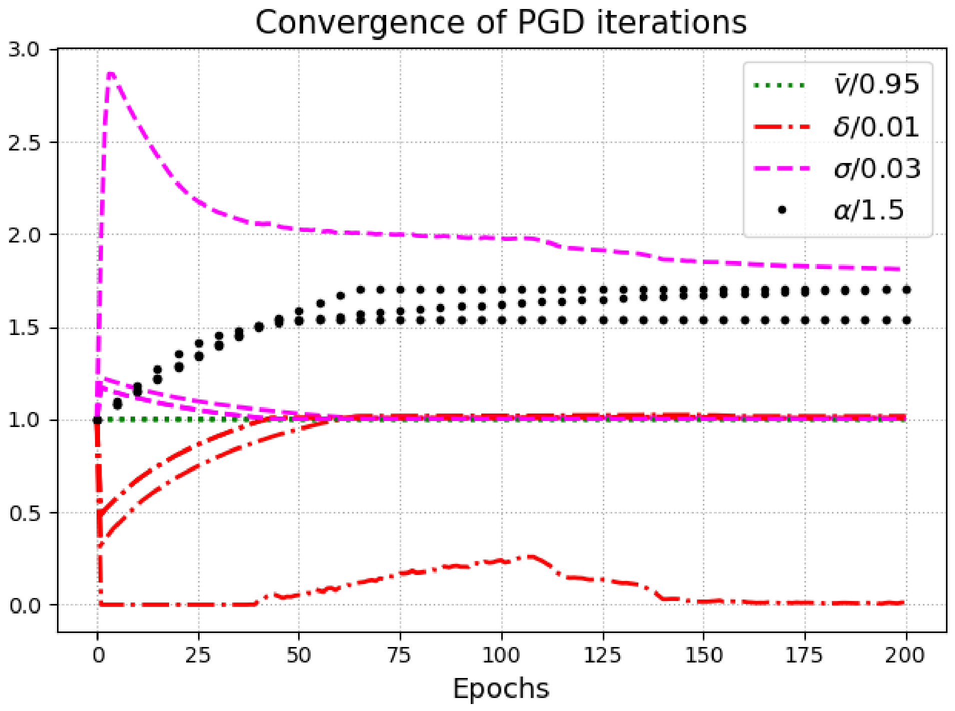


Figure 5.7: Convergence of PGD iterations (5.12)–(5.15) for Volt/VAR rules with $\epsilon = 0.5$. Values of the rule parameters for DERs 12, 22, and 29 are plotted against training epochs. Plots have been normalized with respect to their initial values.

profile across the feeder. Voltages across buses are plotted under three setups: voltages without DER reactive power support, voltages under the default settings $(\bar{v}_n, \delta_n, \sigma, \bar{q}_n) = (1, 0.02, 0.08, \hat{q}_n)$ from IEEE 1547.8 [3]; and voltages under control rules with optimal $\mathbf{z}$. For each bus, voltages for all $S = 80$ scenarios have been marked. The default control rules were found to only marginally improve voltage profiles. On the other hand, optimally designed control rules were successful in significantly lowering voltages and bringing them close to unity at all buses.

We next studied the impact of the stability margin ϵ on the optimal cost $L(\mathbf{z})$ of (5.11) under Volt/VAR rules. Recall that ϵ determines the feasible space of design parameters via (5.9). The larger the ϵ, the more restricted problem (5.10) is. Table 5.1 confirms this by presenting the objectives during training for a range of ϵ values. Table 5.1 also lists the chosen initial

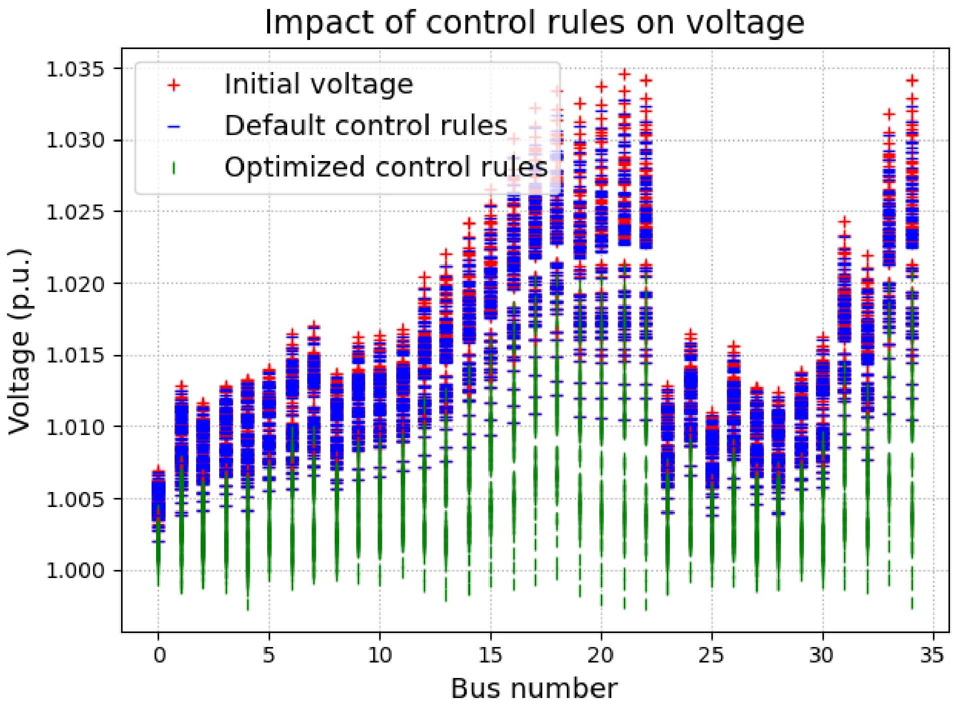


Figure 5.8: Voltages across the buses of the IEEE 37-bus feeder for 80 scenarios. Voltages are highest without reactive power compensation. While the control rules with the default settings only marginally lower voltages closer to 1, they are significantly outmatched by control rules with optimal design parameters.

value for $\boldsymbol{\alpha}$, represent by $\boldsymbol{\alpha}_{\text{init}}$, that renders the initial $\mathbf{z}$ feasible for the corresponding value of ϵ. The objective converged to the highest value for $\epsilon = 0.9$ and the lowest for $\epsilon = 0.5$. Note that for the studied scenarios, reducing ϵ below 0.5 did not impact the optimal value of the objective, which indicates that the feasible space for $\epsilon = 0.5$ contains the optimizers for all $\epsilon \leq 0.5$ as well. Consequently, the value of ϵ has been fixed at 0.5 for the subsequent results on the 37-bus feeder.

To verify the optimality and scalability of DNN-based ORD, we benchmarked them against the MINLP formulation of (5.20). The MINLP was allowed to run until completion or till 300 seconds, whichever happened earlier. Scaling with respect to both the number of scenarios as well as DERs was studied. Table 5.2 reports the results for the case when the number

Table 5.1: Test results capturing the effect of ϵ on the optimal objective value for Volt/VAR control rules. The smaller the ϵ, the larger the feasible region for rule parameters is, and so lower voltage regulation values can be attained.

ϵ	$\boldsymbol{\alpha}_{\text{init}}$	Objective (p.u.)
0.9	0.4	$2.22 \cdot 10^{-3}$
0.8	0.5	$1.37 \cdot 10^{-3}$
0.7	1	$1.06 \cdot 10^{-3}$
0.6	1.5	$9.73 \cdot 10^{-4}$
0.5	1.5	$8.50 \cdot 10^{-4}$

Table 5.2: Tests comparing the MINLP with the DNN-based ORD for different numbers of scenarios S, with $N_G = 5$ smart DERs.

S	MINLP			DNN	
	Solved	Time (s)	Obj. (p.u.)	Time (s)	Obj. (p.u.)
10	Yes	2.45	$9.82 \cdot 10^{-4}$	18.16	$9.82 \cdot 10^{-4}$
20	Yes	3.64	$1.57 \cdot 10^{-3}$	20.08	$1.58 \cdot 10^{-3}$
40	Yes	123.32	$2.78 \cdot 10^{-3}$	20.63	$2.78 \cdot 10^{-3}$
80	No	300	$2.68 \cdot 10^{-3}$	22.17	$2.62 \cdot 10^{-3}$

of smart DERs was fixed to $N_G = 5$ and scenarios were increased from $S = 10$ to 80. As evident from Table 5.2, the DNN-based ORD scaled much better than the MINLP for larger S, as expected. Furthermore, the DNN-based ORD was able to achieve the same objective as the MINLP across all tested values of S, which is remarkable since SGD for non-convex problems can only guarantee convergence to stationary points. Similar conclusions can be drawn from Table 5.3 where we fixed $S = 80$ and varied N_G from 2 to 10. The MINLP was faster than the DNN-based approach for $N_G = 2$, but could not be solved within 300 seconds if more inverters were added. On the other hand, the DNN-based ORD scaled gracefully with the N_G and achieved lower objectives for all $N_G \geq 4$.

The scalability of the DNN-based control rules was also confirmed by implementing them for the larger IEEE 123-bus feeder of Fig. 5.9. Active load data was generated by averaging homes with IDs 20-386, three at a time, and were serially assigned them to buses 2-123. Solar generation from 10 panels with IDs $\{106, 116, 119, 296, 372, 650, 734, 841, 933, 1574\}$

Table 5.3: Comparing MINLP with the DNN-based approach for different N_G and $S = 80$.

N_G	MINLP			DNN	
	Solved	Time (s)	Obj. (p.u.)	Time (s)	Obj. (p.u.)
2	Yes	3.90	$3.62 \cdot 10^{-3}$	14.12	$3.62 \cdot 10^{-3}$
4	No	300	$3.22 \cdot 10^{-3}$	17.96	$3.18 \cdot 10^{-3}$
6	No	300	$2.77 \cdot 10^{-3}$	21.95	$2.35 \cdot 10^{-3}$
8	No	300	$1.40 \cdot 10^{-3}$	33.42	$1.16 \cdot 10^{-3}$
10	No	300	$1.20 \cdot 10^{-3}$	39.76	$8.50 \cdot 10^{-4}$

Table 5.4: Test results comparing the MINLP with the DNN-based ORD approach for the single-phase IEEE 123-bus feeder, across different time periods for $N_G = 10$ DERs and $S = 80$ scenarios.

Time	MINLP			DNN	
	Solved	Time (s)	Obj. (p.u.)	Time (s)	Obj. (p.u.)
1 pm	No	500	$9.26 \cdot 10^{-4}$	28.6	$8.95 \cdot 10^{-4}$
2 pm	No	500	$6.69 \cdot 10^{-4}$	30.18	$6.40 \cdot 10^{-4}$
3 pm	No	500	$4.17 \cdot 10^{-4}$	27.55	$3.92 \cdot 10^{-4}$
4 pm	No	500	$2.17 \cdot 10^{-4}$	29.83	$2.09 \cdot 10^{-4}$
5 pm	No	500	$2.98 \cdot 10^{-3}$	27.53	$2.87 \cdot 10^{-4}$

was added to buses $\{17, 29, 32, 39, 50, 71, 78, 96, 100, 114\}$, respectively. All buses with solar were equipped with smart DERs for reactive power support. The DNNs for Volt/VAR rules were trained with the learning rate of 0.01, with ϵ set to 0.5. The design parameters $\mathbf{z} := (\bar{\mathbf{v}}, \boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$ were initialized at the feasible point $(1.05, 0.1, 0.3, 1.5)$. With N_G and S fixed at 10 and 80, respectively, the DNN-based ORD was compared to the MINLP one. For this larger network, the MINLP solver was allowed to run until 500 seconds. To ensure repeatability, the results were repeated across several time periods between $1 - 6$ PM, and have been compiled in Table 5.4. For all time periods, the DNN-based solver scaled well in terms of the DNN training time. The MINLP solver could not converge within 500 seconds and was outperformed by the DNN-based solver in terms of final objective values across all setups.

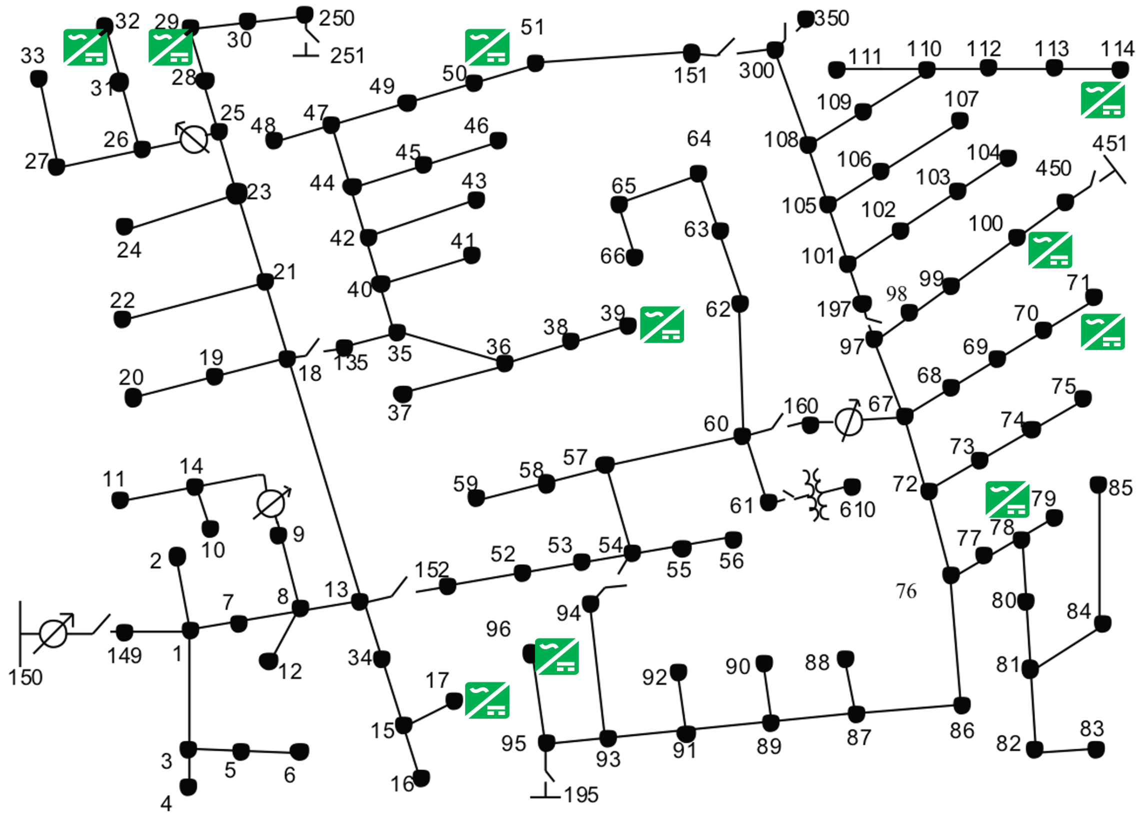


Figure 5.9: Inverter siting on the IEEE 123-bus distribution feeder.

5.9.2 Tests on a Multiphase Feeder

The DNN-based control rules were also tested on the multiphase IEEE 13-bus feeder. Active loads from homes with IDs 20-379 were averaged ten homes at a time. The resulting 36 averaged loads were added to buses 1-12, allocating all three phases for a bus before moving on to the next one. Solar generation was added to nodes as per the panel assignments shown in Fig. 5.10. Values in red, green, and blue correspond to panel IDs assigned to Phases A, B, and C, respectively. Reactive power compensation was provided by nine inverters added across phases, and bus indices, as shown in Fig 5.10, with the colors indicating the corresponding phase.

The learning rate for DNN-based control rules was set to 0.1, with the design parameters

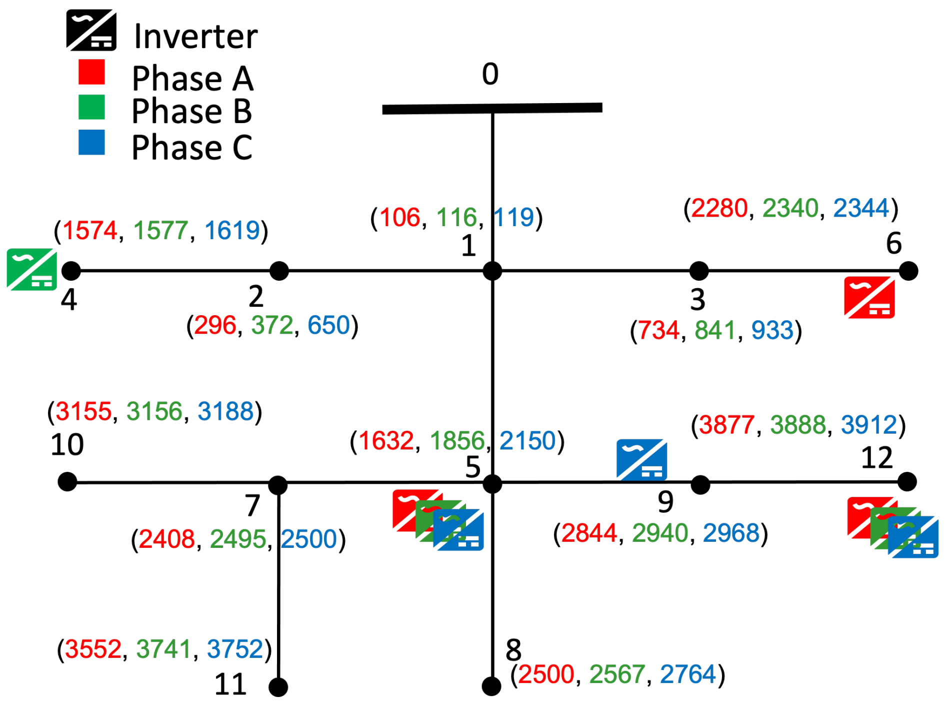


Figure 5.10: The three-phase IEEE 13-bus distribution feeder system.

$\mathbf{z} := (\bar{\mathbf{v}}, \boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$ initialized to feasible values $(0.95, 0.01, 0.3, 1.5)$. In the absence of an MINLP solver, the optimized DNN-based control rules were benchmarked against control rules with the default settings from the IEEE 1547.8 standard. Table 5.5 collects the values for the objective (5.10) for $S = 80$ scenarios, across different windows of time from $1-5$ pm, under three control schemes– no reactive power compensation, optimized control rules, and default control rules. The default control rules did not manage to significantly reduce the objective (5.10), as the grid conditions $\tilde{\mathbf{v}}$ were observed to frequently fall in the deadband of the default control rules. In contrast, the optimized control rules took the grid conditions $\tilde{\mathbf{v}}$ into consideration while designing the deadband, and hence improved voltage profiles considerably.

Table 5.5: Test results on the multiphase IEEE 13-bus feeder for $N_G = 9$ inverters and $S = 80$ scenarios. Comparing the objective (5.10) under three scenarios: no reactive power compensation, optimized control rules, and the default rules per IEEE 1547.

Time	$\mathbf{q} = 0$	Optimized	Default
1 pm	$2.51 \cdot 10^{-3}$	$1.15 \cdot 10^{-3}$	$2.31 \cdot 10^{-3}$
2 pm	$1.48 \cdot 10^{-3}$	$6.89 \cdot 10^{-4}$	$1.42 \cdot 10^{-4}$
3 pm	$6.89 \cdot 10^{-4}$	$4.94 \cdot 10^{-4}$	$6.89 \cdot 10^{-4}$
4 pm	$8.03 \cdot 10^{-4}$	$5.26 \cdot 10^{-4}$	$8.03 \cdot 10^{-4}$
5 pm	$5.51 \cdot 10^{-4}$	$4.11 \cdot 10^{-4}$	$5.51 \cdot 10^{-4}$

5.10 Conclusions

This work has genuinely reformulated the ORD problem to the task of training a DNN using grid scenarios as training data, unit voltages as desired targets for equilibrium voltages, and Volt/VAR rule parameters as weights. The proposed DNN-based ORD framework is general enough to accommodate Volt/VAR rules on single- and multi-phase feeders. We have also reviewed and extended results on the stability and convergence rates of Volt/VAR control rules. For benchmarking purposes, we have also developed an MINLP approach to ORD. The suggested approaches have been validated using real-world data on IEEE test feeders. The tests show that DNN-based ORD seems to outperform the MINLP approach in terms of optimality under time budgets and that optimized ORD curves outperform the default values. Our findings form the foundations for exciting research directions, such as: *d1)* Can the DNN-based ORD framework be extended to designing *incremental* Volt/VAR control rules with favorable stability characteristics? *d2)* What are the appropriate Volt/VAR control rules for three-phase (probably large-scale utility-owned) DERs? *d3)* How can ORD be jointly performed with topology reconfiguration or with optimally setting regulator and capacitor settings? *d4)* Could other voltage regulation or OPF formulations be combined with the DNN-based methodology? *d5)* Can more detailed grid models be incorporated in ORD?

5.11 Appendix

Proof of Proposition 5.2: By a contraction mapping argument, reference [108] proves that as long as stable, the Volt/VAR dynamics $\mathbf{q}^t$ enjoy exponential convergence to the equilibrium $\mathbf{q}^*$. That means that if $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 < 1$, then

$$\|\mathbf{q}^t - \mathbf{q}^*\|_2 \leq \|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 \cdot \|\mathbf{q}^{t-1} - \mathbf{q}^*\|_2.$$

Propagating the previous claim across time and for ϵ-stable rules $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 \leq 1-\epsilon$, we get that

$$\|\mathbf{q}^t - \mathbf{q}^*\|_2 \leq \|\mathbf{q}^0 - \mathbf{q}^*\|_2 \cdot (1-\epsilon)^t \leq 2\|\hat{\mathbf{q}}\|_2 \cdot (1-\epsilon)^t$$

since the initial distance to the equilibrium can be upper bounded by $\|\mathbf{q}^0 - \mathbf{q}^*\|_2 \leq 2\|\hat{\mathbf{q}}\|_2$. Because $\mathbf{v} = \mathbf{X}\mathbf{q} + \tilde{\mathbf{v}}$, translate injection distances to voltage distances

$$\|\mathbf{v}^t - \mathbf{v}^*\|_2 \leq 2\|\mathbf{X}\|_2\|\hat{\mathbf{q}}\|_2(1-\epsilon)^t.$$

To ensure the voltage approximation error at time T is smaller than ϵ_1, or $\|\mathbf{v}^T - \mathbf{v}^*\|_2 \leq 2\|\mathbf{X}\|_2\|\hat{\mathbf{q}}\|_2(1-\epsilon)^T \leq \epsilon_1$, it suffices to select T as

$$T\log(1-\epsilon) \leq \log\frac{\epsilon_1}{2\|\mathbf{X}\|_2\|\hat{\mathbf{q}}\|_2}.$$

The claim follows by noticing that $\log(1-\epsilon) < 0$. □

Proof of Proposition 5.4: Reference [108, Th. 3] shows that the Volt/VAR rules of $\mathbf{f}(\cdot)$ are Lipschitz continuous in $\mathbf{q}$ with $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2$ as the Lipschitz constant, that is

$$\|\mathbf{f}(\mathbf{q}) - \mathbf{f}(\mathbf{q}')\|_2 \leq \|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 \cdot \|\mathbf{q} - \mathbf{q}'\|_2 \tag{5.22}$$

for any $\mathbf{q}$ and $\mathbf{q}'$ obeying (5.3d). From Hölder's inequality for matrix norms, it holds that

$$\begin{aligned}\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2^2 &\leq \|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_1 \cdot \|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_\infty \\ &= \|\operatorname{dg}(\boldsymbol{\alpha})|\mathbf{X}|\|_1 \cdot \|\operatorname{dg}(\boldsymbol{\alpha})|\mathbf{X}|\|_\infty\end{aligned}$$

where $\|\cdot\|_1$ and $\|\cdot\|_\infty$ are defined as the maximum absolute sums column-wise and row-wise, respectively. The equality holds because $\boldsymbol{\alpha}$ has positive entries. It is easy to check that $\|\operatorname{dg}(\boldsymbol{\alpha})|\mathbf{X}|\|_1$ is the maximum entry of vector $|\mathbf{X}|^\top\boldsymbol{\alpha}$, and $\|\operatorname{dg}(\boldsymbol{\alpha})|\mathbf{X}|\|_\infty$ is the maximum entry of vector $\operatorname{dg}(|\mathbf{X}|\mathbf{1})\boldsymbol{\alpha}$. Consequently, enforcing (5.21) results in $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 \leq (1-\epsilon)$. Substituting $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 < (1-\epsilon)$ in (5.22) yields

$$\|\mathbf{f}(\mathbf{q}) - \mathbf{f}(\mathbf{q}')\|_2 \leq (1-\epsilon)\|\mathbf{q} - \mathbf{q}'\|_2 \tag{5.23}$$

Since $\epsilon \in (0,1)$, the above relation is a contraction mapping over the space $\mathbf{q} \in [-\bar{\mathbf{q}}, \bar{\mathbf{q}}]$ with respect to the ℓ_2-norm. The latter establishes the existence and uniqueness of the equilibrium, as well as the exponential convergence of Volt/VAR dynamics. To explicitly derive the convergence result (5.21c), note that $\mathbf{q}^t = \mathbf{f}(\mathbf{q}^{t-1})$ and $\mathbf{q}^* = \mathbf{f}(\mathbf{q}^*)$. From (5.23), we get

$$\|\mathbf{q}^t - \mathbf{q}^*\|_2 \leq (1-\epsilon)\|\mathbf{q}^{t-1} - \mathbf{q}^*\|_2 \leq (1-\epsilon)^t\|\mathbf{q}^0 - \mathbf{q}^*\|_2.$$

The claim follows as $\|\mathbf{q}^0 - \mathbf{q}^*\|_2 \leq 2\|\hat{\mathbf{q}}\|_2$. □

Chapter 6

Scalable Optimal Design of Incremental Volt/VAR Control using Deep Neural Networks

6.1 Publication Details

S. Gupta, A. Mehrizi-Sani, S. Chatzivasileiadis, and V. Kekatos. "Scalable Optimal Design of Incremental Volt/VAR Control using Deep Neural Networks", *IEEE Control Systems Letters*, (submitted Dec. 2022).

6.2 Abstract

Incremental Volt/VAR control rules offer an opportunity to provide decentralized voltage regulation by distributed energy resources (DERs). Unlike their non-incremental counterparts, incremental rules do not suffer from a trade-off between stability and steady-state performance. Despite this desirable characteristic, there is a gap in the literature on the optimal rule design (ORD) of incremental rules. We address this gap by first designing digital twins that mimic the Volt/VAR dynamics of incremental rules, and then utilizing

these digital twins for a deep learning-based ORD. The proposed approach is shown to be effective for both single and multiphase feeders, and stability and convergence analysis is provided for both cases. The attractiveness of incremental rules is confirmed with numerical tests, whereby they achieve lower voltage regulation objectives than their non-incremental counterparts.

6.3 Introduction

Local Volt/VAR control facilitates voltage regulation on distribution grids by providing reactive power compensation from DERs equipped with smart inverters. Different from centralized control schemes which incur large computational and communication burden [24, 66], decide inverter setpoints based on local measurements. Local Volt/VAR control rules can be categorized into *non-incremental* and *incremental* ones. The former compute reactive setpoints based on local voltage readings. IEEE 1547.8 standard prescribes one such non-incremental control rule, whereby the DER setpoints are expressed as piecewise linear functions of voltages [3]. On the other hand, incremental Volt/VAR rules compute the *change* in reactive setpoints as a function of local voltage [25, 53, 58, 103, 106, 109].

The existing literature on designing Volt/VAR control rules can be classified into *stability*- and *optimality*-centric works. Stability-centric works study the effect of Volt/VAR rules as a closed-loop dynamical system, which may be rendered unstable under steep slopes of non-incremental rules [26, 108]. In fact, to ensure stability, non-incremental rules may have to compromise on the quality of the steady-state voltage profile they can attain [53]. On the other hand, incremental rules do not experience stability limitations and can thus, achieve improved voltage profiles compared to the non-incremental rules perhaps at the expense of longer settling time of the associated Volt/VAR dynamics [108].

Optimality-centric works focus on designing stable control rules to minimize a voltage regulation objective. To this end, optimization-based strategies have been employed to design affine non-incremental rules using heuristics [11, 83, 90]. Recently, reference [16] has addressed the optimal design of general incremental rules. It uses DNNs with a single hidden layer to model piecewise-linear functions (potentially with infinite breakpoints) and formulates ORD as a reinforcement learning task. For non-incremental rules, two of our recent works in [72] and [38] have addressed the problem of optimally designing the slope, deadband, saturation, and reference voltage. Reference [72] performs ORD via a bilevel optimization applicable to single-phase feeders. Reference [38] proposes DNN-based digital twins that emulate Volt/VAR dynamics and reformulates ORD as a DNN training task for single- and multiphase feeders alike.

This work extends [38] to incremental rules. While [16] also utilizes DNNs for incremental rule design, we delineate from it in several ways. Reference [16] focuses on voltage control during transient dynamics, whereas this work aims at ORD to drive steady-state voltages closer to unity and over different grid loading scenarios. Reference [16] utilized a DNN to merely model the piecewise-linear mapping of the rule. In contrast, this work develops a DNN-based digital twin that emulates end-to-end Volt/VAR dynamics. Lastly, we provide stability and convergence analysis for single- and multiphase feeders, whereas [16] applies only to single-phase feeders.

6.4 Volt/VAR Control Rules

This section presents some grid modeling preliminaries and contrasts incremental to non-incremental Volt/VAR rules. Consider a radial feeder serving N buses equipped with DERs, indexed by n. Let $(\mathbf{q}^\ell, \mathbf{q})$ collect reactive loads and generations at all nodes. In single-phase

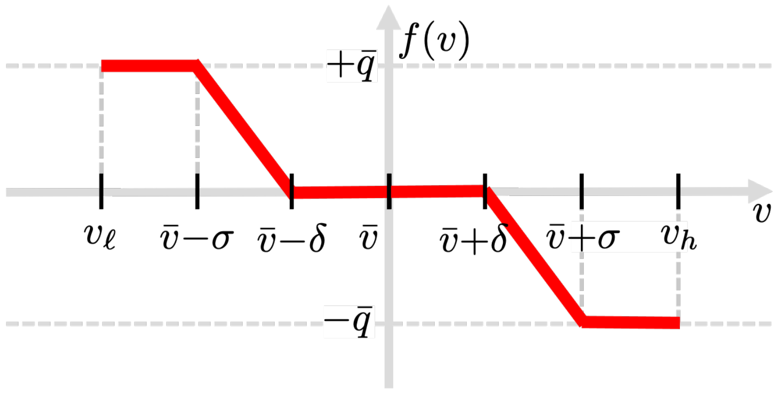


Figure 6.1: Volt/VAR control rule provisioned by the IEEE 1547 Standard [3].

feeders, a node corresponds to a bus. In multiphase feeders, a node corresponds to bus/phase pairs. Vectors $\mathbf{p}$ and $\mathbf{v}$ collect the net active power injections and voltage magnitudes at all nodes. The impact of $\mathbf{q}$ on $\mathbf{v}$ can be approximately captured using the linearized grid model [38]

$$\mathbf{v} \simeq \mathbf{X}\mathbf{q} + \tilde{\mathbf{v}} \tag{6.1}$$

where $\tilde{\mathbf{v}} := \mathbf{R}\mathbf{p} - \mathbf{X}\mathbf{q}^\ell + v_0\mathbf{1}$ models the underlying *grid conditions*, and v_0 is the substation voltage. Vector $\tilde{\mathbf{v}}$ represents the impact of non-controlled quantities $(\mathbf{p}, \mathbf{q}^\ell)$ on voltages. Matrices $(\mathbf{R}, \mathbf{X})$ depend on the feeder topology. For single-phase feeders, they are symmetric positive definite with positive entries [92]. For multiphase feeders, they are non-symmetric and have and negative entries too [38, 53].

Vector $\mathbf{q}$ in (6.1) carries the reactive injections by DERs we would like to control. Per the non-incremental rules of the IEEE 1547 Standard [3], DER setpoints are decided based on the Volt/VAR curve of Fig. 6.1. The standard further specifies that $\mathbf{z}$ should belong to a prespecified polytope $\mathcal{P}$; see [3, 72]. The negative slope of the linear segment can be expressed as $\alpha := \bar{q}/(\sigma - \delta)$. The interaction of Volt/VAR rules with the feeder gives rise to nonlinear dynamics. Volt/VAR dynamics are stable if $\|\operatorname{dg}(\boldsymbol{\alpha})\mathbf{X}\|_2 < 1$, where $\operatorname{dg}(\boldsymbol{\alpha})$ is a diagonal matrix carrying the rule slopes across all buses on its diagonal [27]. If stable, the equilibrium of Volt/VAR dynamics cannot be expressed in closed form. Interestingly, the equilibrium reactive setpoints coincide with the minimizer of convex problem [27]:

$$\min_{-\bar{\mathbf{q}}\leq\mathbf{q}\leq\bar{\mathbf{q}}} \frac{1}{2}\mathbf{q}^\top\mathbf{X}\mathbf{q}+\mathbf{q}^\top(\tilde{\mathbf{v}}-\bar{\mathbf{v}})+\frac{1}{2}\mathbf{q}^\top\operatorname{dg}^{-1}(\boldsymbol{\alpha})\mathbf{q}+\boldsymbol{\delta}^\top|\mathbf{q}| \tag{6.2}$$

where $|\mathbf{q}|$ applies the absolute value on $\mathbf{q}$ entrywise. Problem (6.2) depends on rule parameters $(\bar{v},\delta,\sigma,\alpha)$ collected across all buses on the $4N$-long vector $\mathbf{z}:=(\bar{\mathbf{v}},\boldsymbol{\delta},\boldsymbol{\sigma},\boldsymbol{\alpha})$. Let $\mathbf{q}_{\mathbf{z}}(\tilde{\mathbf{v}})$ denote the equilibrium injections, and

$$\mathbf{v}_{\mathbf{z}}(\tilde{\mathbf{v}}):=\mathbf{X}\mathbf{q}_{\mathbf{z}}(\tilde{\mathbf{v}})+\tilde{\mathbf{v}}$$

the associated equilibrium voltages reached by Volt/VAR rules parameterized by $\mathbf{z}$ under grid conditions $\tilde{\mathbf{v}}$.

Optimal rule design (ORD) could be broadly defined as the task of selecting $\mathbf{z}$ to attain desirable voltage profiles at equilibrium. In particular, the operator could sample S representative grid loading scenarios $\{\tilde{\mathbf{v}}_s\}_{s=1}^S$ and find $\mathbf{z}$ to minimize the distance of equilibrium voltages from unity, that is $\frac{1}{S}\sum_{s=1}^S\|\mathbf{v}_{\mathbf{z}}(\tilde{\mathbf{v}}_s)-\mathbf{1}\|_2^2$. When ORD is solved for non-incremental rules, vector $\mathbf{z}$ is confined to satisfy the IEEE 1547 polytopic constraints on $\mathbf{z}$ as well as stability constraints specifically on $\boldsymbol{\alpha}$.

This trade-off between stability and steady-state voltage regulation performance can be surpassed with *incremental* control rules [53, 108]. Incremental rules express the *change* rather than the actual value in q-setpoints as a function of voltage. One option for incremental rules is to implement a proximal gradient descent (PGD) algorithm to solve the QP in (6.2) as suggested in [53]. The PGD iterations yield the rule

$$y_n^t:=\tilde{\alpha}_n\cdot\big(q_n^t-\mu(v_n^t-\bar{v}_n)\big) \tag{6.3a}$$

$$q_n^{t+1} := g_n\left(y_n^t\right) \tag{6.3b}$$

where $\tilde{\alpha}_n := \frac{1}{1+\mu/\alpha_n}$ and $\mu > 0$ is a step-size. Further, the function $g_n(y_n)$ is the proximal operator given by

$$g_n(y_n) := \begin{cases} +\overline{q}_n & , \; y_n > \overline{q}_n + \mu\tilde{\delta}_n \\ y_n - \mu\tilde{\delta}_n & , \; \mu\tilde{\delta}_n < y_n \leq \overline{q}_n + \mu\tilde{\delta}_n \\ 0 & , \; -\mu\tilde{\delta}_n \leq y_n \leq \mu\tilde{\delta}_n \\ y_n + \mu\tilde{\delta}_n & , \; -\overline{q}_n - \mu\tilde{\delta}_n \leq y_n < -\mu\tilde{\delta}_n \\ -\overline{q}_n & , \; y_n < -\overline{q}_n - \mu\tilde{\delta}_n . \end{cases} \tag{6.4}$$

where $\tilde{\delta}_n := \frac{\delta_n}{1+\mu/\alpha_n}$. As with the non-incremental rules of Figure 6.1, the rules in (6.4) are local as they are driven by local data. However, now q_n^{t+1} depends on both (v_n^t, q_n^t), and not only v_n^t as in Figure 6.1.

The PGD iterations solve (6.2), but in a different way compared to non-incremental rules. Hence, incremental and non-incremental rules converge to the same equilibrium. The advantage is that as long as $\mu < 2/\lambda_{\max}(\mathbf{X})$, incremental rules converge to this equilibrium without any restriction on $\boldsymbol{\alpha}$ [53]. Parameter $\boldsymbol{\alpha}$ appears as slopes of non-incremental rules, but as memory-related terms of incremental rules. Either way, they are the same parameters appearing in (6.2).

Accelerated incremental control rules. Although the rules of (6.3) cope with the stability-performance trade-off, their settling time can be long. Thanks to the convergence rate of PGD, we know that the incremental rules attain an ε-optimal cost of (6.2) within $-\frac{2\log\varepsilon}{\log 2}\kappa\left(\mathbf{X}\right)$ iterations. Here $\kappa(\mathbf{X}) = \lambda_{\max}(\mathbf{X})/\lambda_{\min}(\mathbf{X})$ is the condition number of $\mathbf{X}$. Because $\kappa(\mathbf{X})$ can be large (e.g., 10^{-4} for benchmark feeders), references [53] put forth *accelerated* incremental rules based on Nesterov's accelerated PGD (APGD). The accelerated incremental control rules take $-\frac{2\log\varepsilon}{\log 2}\sqrt{\kappa\left(\mathbf{X}\right)}$ iterations to attain an ε-optimal cost for (6.2). They are given as

$$\tilde{y}_n^t := \left(1+\beta_t\right) y_n^t - \beta_t y_n^{t-1} \tag{6.5a}$$

$$q_n^{t+1} := g_n\left(\tilde{y}_n^t\right) \tag{6.5b}$$

where $\beta_t := \frac{t-1}{t+2}$, while y_n^t and $g_n(y_n)$ are as defined in (6.3a) and (6.4). Updates (6.5) remain local, but introduce additional memory as q_n^{t+1} depends on (v_n^t, q_n^t) and (v_n^{t-1}, q_n^{t-1}).

6.5 ORD for Incremental Rules for 1ϕ Feeders

We utilize the ORD approach from [38] to design incremental rules. Consider a setup where the Volt/VAR rules are to be designed to provide voltage regulation over a given set of S load/solar scenarios $\{(\mathbf{p}_s^g, \mathbf{p}_s^\ell, \mathbf{q}_s^\ell)\}_{s=1}^S$. Such a setup allows the Volt/VAR rules to be updated periodically, while operating autonomously between those time periods. Working with an alternate parameterization $\tilde{\mathbf{z}} = (\bar{\mathbf{v}}, \tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\delta}}, \bar{\mathbf{q}})$, for reasons that will become evident shortly, let $\tilde{\mathcal{Z}}$ represent the feasible space of parameters defined by constraints $\tilde{\mathbf{z}} \geq \mathbf{0}$, $\bar{\mathbf{q}} \leq \hat{\mathbf{q}}$, and $\tilde{\boldsymbol{\alpha}} \leq \mathbf{1}$. Together these constraints ensure that the original curve parameters $(\bar{\mathbf{v}}, \boldsymbol{\delta}, \boldsymbol{\sigma}, \bar{\mathbf{q}})$ remain non-negative. Next, consider a DNN-based digital twin $\Phi(\tilde{\mathbf{v}}_s; \tilde{\mathbf{z}})$ that simulates the incremental Volt/VAR dynamics (6.3) (or accelerated dynamics (6.5)) across all the DERs. The digital-twin $\Phi(\tilde{\mathbf{v}}_s; \tilde{\mathbf{z}})$ accepts the grid conditions vector $\tilde{\mathbf{v}}_s$ as the input and produces the equilibrium voltage $\mathbf{v}_s^*$ as the output. If the curve parameters $\tilde{\mathbf{z}}$ also parameterize the DNN $\Phi(\tilde{\mathbf{v}}_s; \tilde{\mathbf{z}})$, by appearing in its weights and biases, then ORD can be formulated as the

DNN training task [38]

$$\min_{\tilde{\mathbf{z}} \in \tilde{\mathcal{Z}}} \; L(\tilde{\mathbf{z}}) := \frac{1}{2S} \sum_{s=1}^{S} \left\| \Phi\left(\tilde{\mathbf{v}}_s; \tilde{\mathbf{z}}\right) - \mathbf{1} \right\|_2^2 \tag{6.6}$$

Since the problem (6.6) contains simple box constraints, one can leverage efficient deep learning toolkits, such as Pytorch, to perform the above DNN training task. Now the question remains whether a DNN $\Phi(\tilde{\mathbf{v}}_s; \tilde{\mathbf{z}})$, that has the above listed properties, can even be created.

Fortunately, the answer to this question is a *yes*. We focus on the accelerated incremental rules (6.5) as they require a DNN of smaller depth T, thanks to their faster convergence. The key observation is that the operator $g(y)$ in (6.4) is a piecewise linear function with four breakpoints [53], and is expressible as a superposition of the four single-breakpoint piecewise linear curves drawn in Fig. 6.2. Therefore, the incremental rule (6.5) for one inverter can be modeled by the DNN of Fig. 6.3 with four hidden layers parameterized by $(\bar{v}_n, \tilde{\alpha}_n, \tilde{\delta}_n, \bar{q}_n)$. The DNN has two outputs: the intermediate quantity y_n^t that is required for the next iteration (6.5a), and the setpoint q_n^{t+1}. The plain incremental rule can be obtained by simply removing the third hidden layer (setting $\beta^t = 0$) and ignoring output y_n^t.

We will use the DNN of Fig. 6.3 as a building block and call it IC$_n$. To simulate dynamics with the control rule (6.5), block IC$_n$ has to be implemented across inverters and time steps. Let the collection of all IC$_n$'s be termed a layer. At each time step t, a layer of IC$_n$'s receives inputs $\mathbf{q}^t$, $\mathbf{v}^t$, and $\mathbf{y}^{t-1}$, and produces the new setpoint $\mathbf{q}^{t+1}$ as well as the intermediate quantity $\mathbf{y}^t$ as outputs. The inverters implement the $\mathbf{q}^{t+1}$ to obtain the new voltage $\mathbf{v}^{t+1}$ by (6.1), and the process is repeated over T time steps. These interactions result in a larger DNN parameterized by $\tilde{\mathbf{z}} := (\bar{\mathbf{v}}, \tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\delta}}, \bar{\mathbf{q}})$, whose recurrent representation is shown in Fig. 6.4. Since the DNN of Fig. 6.4 replicates the same IC$_n$s across all T layers, it enables *weight*

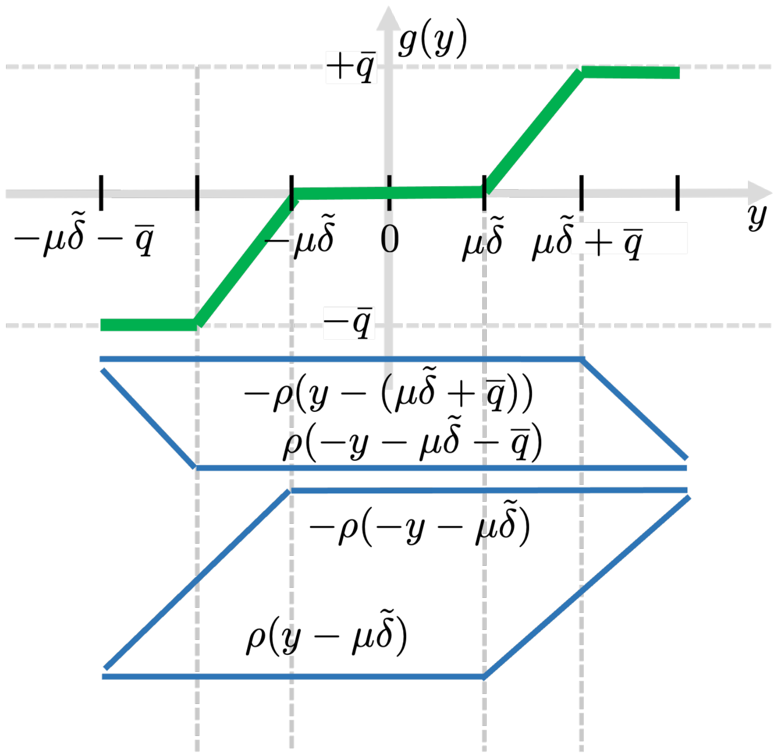

$g(y)$
$+\bar{q}$
$-\mu\tilde{\delta}-\bar{q}$
$-\mu\tilde{\delta}$
0
$\mu\tilde{\delta}$
$\mu\tilde{\delta}+\bar{q}$
y
$-\bar{q}$
$-\rho(y-(\mu\tilde{\delta}+\bar{q}))$
$\rho(-y-\mu\tilde{\delta}-\bar{q})$
$-\rho(-y-\mu\tilde{\delta})$
$\rho(y-\mu\tilde{\delta})$

Figure 6.2: Operator g(y) used in incremental rules as a sum or ReLUs.

sharing and the number of trainable parameters of the DNN remain fixed at only $4N$.

The output of the DNN of Fig. 6.4 simulates well the accelerated control rule only if T is sufficiently large. Recall that each DNN layer corresponds to a (A)PGD iteration. *How many such iterations T are needed for the simulated dynamic voltages $\mathbf{v}^T$ to get sufficiently close to equilibrium voltages?*

Proposition 6.1. *Let the DNN in Fig. 6.4 implement the plain incremental rules (6.3). Let $\kappa := \lambda_{\max}(\mathbf{X})/\lambda_{\min}(\mathbf{X})$. The DNN depth T required to ensure $\|\Phi\left(\tilde{\mathbf{v}};\mathbf{z}\right)-\mathbf{v}^*(\mathbf{z})\|_2 \leq \epsilon_1 \;\forall\; \tilde{\mathbf{v}}$ is*

$$T \geq \left(\frac{\kappa-1}{2}\right)\log\frac{2\|\mathbf{X}\|_2\|\hat{\mathbf{q}}\|_2}{\epsilon_1} \tag{6.7}$$

Plugging the values $\|\mathbf{X}\|_2 = 4.63\cdot 10^{-1}$ and $\kappa = 8.48\cdot 10^2$ for the IEEE 37-bus feeder, $\|\hat{\mathbf{q}}\|_2 =$

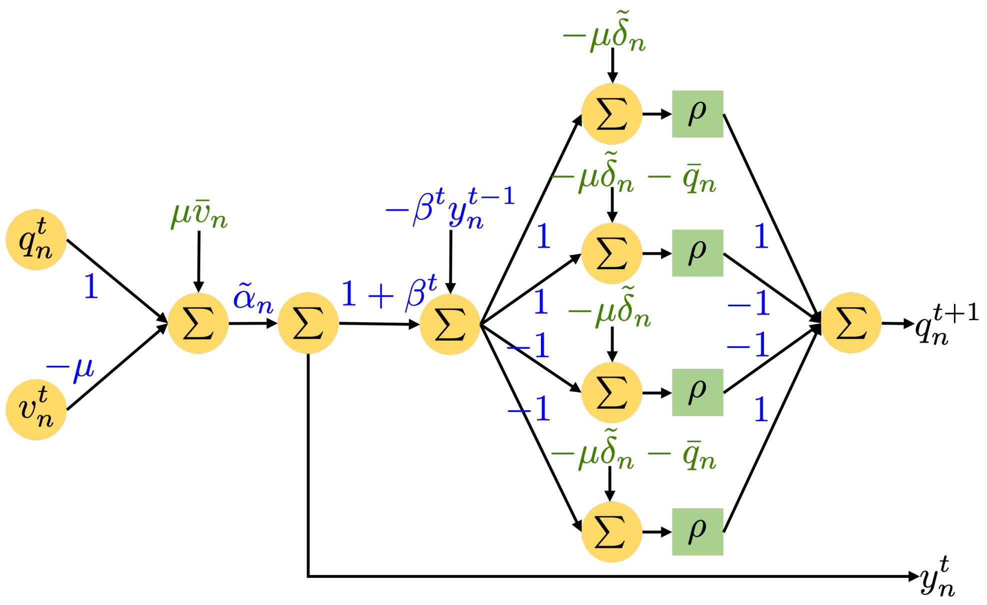


Figure 6.3: Accelerated incremental control rules (6.5) modeled using a DNN with 4 hidden layers. Plain incremental rules can be obtained by dropping the second layer (setting $\beta^t = 0$) and ignoring output y_n^t.

0.1, and $\epsilon_1 = 10^{-5}$ in (6.7), yields $T \geq 2892$, which is significantly large. A key contributor to this large T is the $\mathcal{O}(\kappa)$ term in (6.7). This promulgates the adoption of accelerated incremental rules (6.5), which are known to have $\mathcal{O}(\sqrt{\kappa})$ dependence. Interestingly, during implementation, one need not fix T to the above worst-case bounds.Leveraging dynamic computational graphs offered by Python libraries such as Pytorch, one may determine T *on the fly* depending on the convergence of $\mathbf{v}_s$.

The DNN of Fig. 6.4 can be trained using projected stochastic gradient descent iterations (SPGD) [38]

$$\tilde{\mathbf{z}}^{i+1} = \left[\tilde{\mathbf{z}}^i - \frac{\mu}{2B}\nabla_{\tilde{\mathbf{z}}^i}\left(\sum_{s\in\mathcal{B}_i}\|\Phi(\tilde{\mathbf{v}}_s;\tilde{\mathbf{z}}) - \mathbf{1}\|_2^2\right)\right]_{\tilde{\mathcal{Z}}} \tag{6.8}$$

where $\mu > 0$ is the step size; set $\mathcal{B}_i$ is a batch of B scenarios; and $[\cdot]_{\tilde{\mathcal{Z}}}$ represents the projection into the feasible space $\tilde{\mathcal{Z}}$. Since $\tilde{\mathcal{Z}}$ is formed by box-constraints, the operation $[\cdot]_{\tilde{\mathcal{Z}}}$

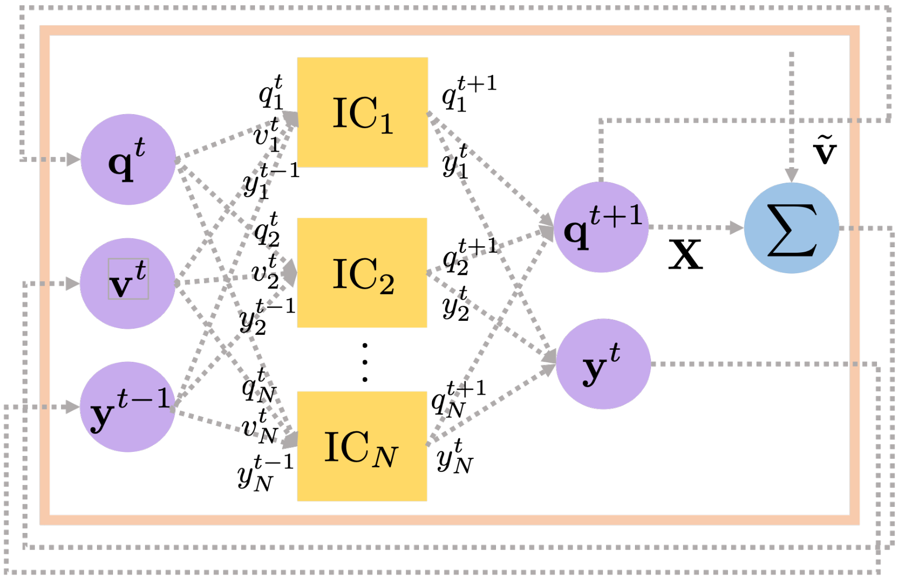


Figure 6.4: Recurrent DNN implementation for accelerated incremental rules.

is executed by simply *clipping* the values to lie within the allowed box. Lastly, the quantity $\nabla_{\tilde{\mathbf{z}}^i}(\cdot)$ represents the gradient with respect to $\tilde{\mathbf{z}}$ evaluated at $\tilde{\mathbf{z}} = \tilde{\mathbf{z}}^i$, and is calculated efficiently thanks to *gradient back-propagation*.

Because incremental rules enforce fewer constraints on Φ than non-incremental rules (c.f. [38, Sec. IV]), they can attain lower objective values in (6.6) and therefore better voltage profiles. This advantage is verified during our numerical tests.

6.6 ORD for 3ϕ Feeders via Deep Learning

For 3ϕ Feeders, the sensitivity matrix $\mathbf{X}$ is non-symmetric and has negative entries. Therefore, the rule design process and analysis from previous sections needs to be modified. Recall that for single-phase rules, incremental rules were obtained as the PGD iterations solving problem (6.2). Lacking an equivalent optimization problem for multiphase feeders precludes a similar approach for multiphase feeders. Nonetheless, the incremental rules of (6.3) can be shown to be stable and converging for multiphase feeders. The ensuing proposition follows from the proof of Proposition 6.1 and [53, Prop. 6].

Proposition 6.2. *Let* $\mathbf{U}\ \mathbf{U}^\top$ *be the eigendecomposition of matrix* $\mathbf{X}\mathbf{X}^\top$*. The incremental rules of* (6.3) *are stable and converging for the multiphase feeder for* $\mu \in \left(0, \lambda_{\min}\left(\ ^{-1/2}\mathbf{U}^\top\left(\mathbf{X}+\mathbf{X}^\top\right)\mathbf{U}\ ^{-1/2}\right)\right.$

For proof, from reference [53, Prop. 6], the quantity $\|\mathbf{I}-\mu\mathbf{X}\|_2 < 1$ when μ is selected as above. Consequently, from the proof of Prop. 6.1 [c.f. (6.9)], the incremental rules (6.3) are stable and converging to the equilibrium $\mathbf{q}^*$.

Similar to the single-phase case, incremental rules in multiphase feeders allow us to bypass the stability/performance trade-off. It is worth stressing that different from the single-phase case, incremental and non-incremental rules on multiphase feeders are not guaranteed to converge to the same equilibrium, as confirmed by the numerical tests.

The ORD task for multiphase feeders can be formulated as deep learning task, as in (6.6), with appropriate modifications. Firstly, the sensitivity matrices $\mathbf{R}$ and $\mathbf{X}$ need to appropriately modified for the multiphase feeders. Secondly, the DNNs for multiphase feeders have $12N$ trainable parameters, since each layer consists of $3N$ building modules corresponding to bus/phase (node) combinations. Lastly, the step size has to be selected per Proposition 6.2. The DNN depth needed for incremental rules in multiphase feeders is derived next.

Proposition 6.3. *Let the DNN of Fig. 6.4 implement incremental rules* (6.3) *for multiphase feeders. Let* μ *be selected as per Prop. 6.2. The DNN depth* T *ensuring* $\|\Phi\left(\tilde{\mathbf{v}};\mathbf{z}\right)-\mathbf{v}^*(\mathbf{z})\|_2 \leq \epsilon_1$ *is*

$$T \geq \frac{\log \frac{\epsilon_1}{2\|\mathbf{X}\|_2\|\hat{\mathbf{q}}\|_2}}{\log\|\mathbf{I}-\mu\mathbf{X}\|_2}.$$

where proof simply follows from the proof of Prop. 6.1.

Table 6.1: Comparing non-incremental and incremental rules for the single-phase IEEE 37-bus feeder.

Time	$\mathbf{q} = 0$	Non-incremental		Incremental	
	Obj. (p.u.)	Time (s)	Obj. (p.u.)	Time (s)	Obj. (p.u.)
1 pm	$3.01 \cdot 10^{-3}$	37.98	$3.68 \cdot 10^{-4}$	39.39	$3.66 \cdot 10^{-4}$
2 pm	$3.13 \cdot 10^{-3}$	42.93	$4.26 \cdot 10^{-4}$	37.91	$4.25 \cdot 10^{-4}$
3 pm	$4.24 \cdot 10^{-3}$	45.02	$8.59 \cdot 10^{-4}$	34.97	$8.50 \cdot 10^{-4}$
4 pm	$2.12 \cdot 10^{-3}$	48.30	$1.47 \cdot 10^{-4}$	38.52	$1.48 \cdot 10^{-4}$
5 pm	$8.53 \cdot 10^{-4}$	47.37	$9.70 \cdot 10^{-5}$	374.01	$6.90 \cdot 10^{-5}$

6.7 Numerical Tests

We benchmark the performance of DNN-based incremental rules against that of non-incremental rules from reference [38] for both single and multiphase feeders. Real-world data was sourced from the Smart* project on April 2, 2011 [14], as explained in [38] . The DNNs were implemented and trained using Pytorch.

We first compare the non-incremental and incremental rules, both designed by the DNN-based approach, for the single-phase IEEE 37-bus feeder of Figure 6.5. Homes with IDs $20-369$ were averaged 10 at a time and successively added as active loads to buses $2-26$ as shown in Fig. 6. Active generation from solar panels was also added, as per the mapping in Fig. 6. Buses $\{6, 9, 11, 12, 15, 16, 20, 22, 24, 25\}$ were assumed to host DERs with Volt/VAR control. Incremental rules were simulated in their accelerated version. Both sets of rules were trained over $S = 80$ scenarios and 200 epochs, with a learning rate of 0.001, using the Adam optimizer. The step size μ for the incremental rules was fixed at 1. To ensure repeatability, the results were repeated across several time periods between $1-6$ PM, and have been compiled in Table 6.1. Across all time periods, incremental rules obtained marginally lower objectives than non-incremental ones, with a somewhat significant difference for the 5 pm time window. This behavior is explained because $\mathbf{z}$ has a larger feasible space to lie in with incremental rules.

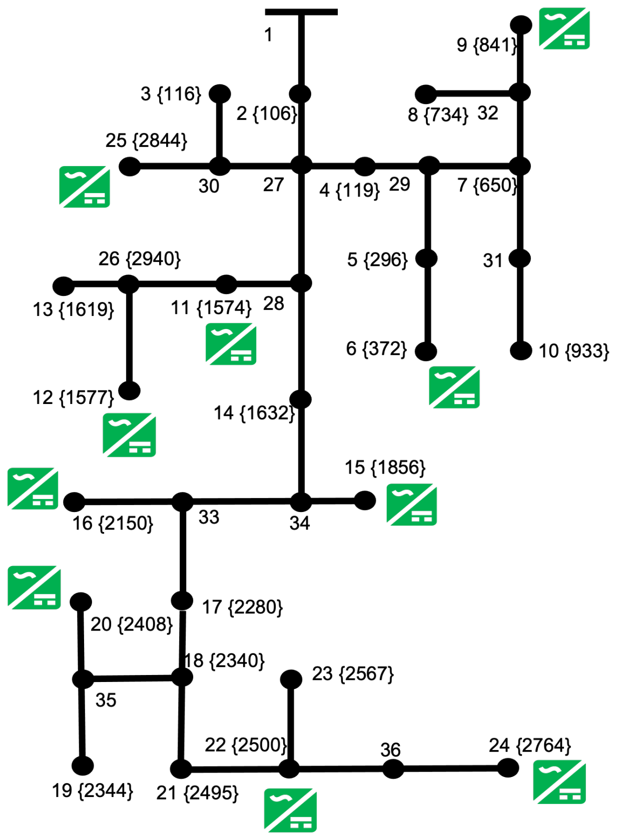


Figure 6.5: The IEEE 37-bus feeder used for the numerical tests. Node numbering follows the format `node number {panel ID}`. The inverters at nodes $\{6, 9, 11, 12, 15, 16, 20, 22, 24, 25\}$ provide reactive power control; the rest operate at unit power factor.

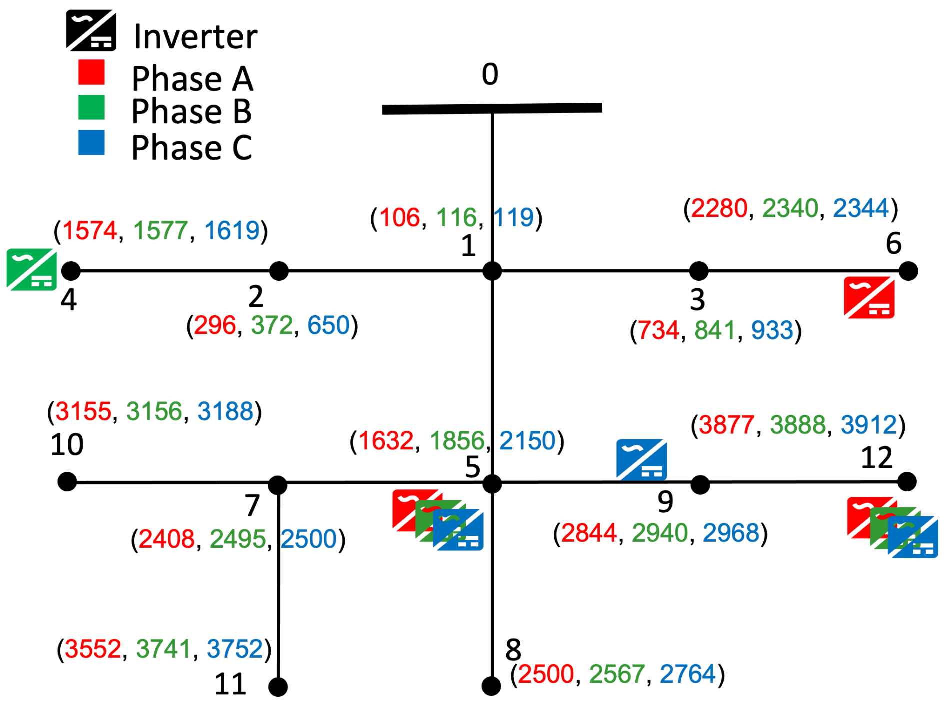


Figure 6.6: The three-phase IEEE 13-bus distribution feeder system.

The DNN-based incremental control rules were also benchmarked against the non-incremental counterpart on the multiphase IEEE 13-bus feeder, using the testing setup from [38]. Active loads were sampled 10 at a time from homes with IDs 20-379 and added to all three phases for the buses 1-12. Figure 6.6 also shows the solar panel assignments shown in Fig 6.6 for solar generation. Lastly, nine DERs with inverters were added across phases, and bus indices, as shown in Fig 6.6. The learning rates for non-incremental and incremental DNNs were set as 0.1 and 0.001, respectively, with the design parameters $\mathbf{z} := (\bar{\mathbf{v}}, \boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$ initialized to feasible values $(0.95, 0.01, 0.3, 1.5)$. Table 6.2 compares the performance of two categories of rules over multiple periods, for $S = 80$ and 300 epochs. While incremental rules took longer times to train, they were successful in lowering the objective (6.6) by more than 50%, thus yielding improved voltage profiles across all time periods.

Table 6.2: Test results comparing non-incremental and incremental rules for the multiphase IEEE 13-bus feeder for samples from 5 PM.

Time	$\mathbf{q}=0$	Non-incremental		Incremental	
	Obj. (p.u.)	Time (s)	Obj. (p.u.)	Time (s)	Obj. (p.u.)
1 pm	$2.51\cdot10^{-3}$	64.65	$1.15\cdot10^{-3}$	199.24	$4.11\cdot10^{-4}$
2 pm	$1.48\cdot10^{-3}$	66.60	$6.89\cdot10^{-4}$	209.92	$3.03\cdot10^{-4}$
3 pm	$6.89\cdot10^{-4}$	74.68	$4.94\cdot10^{-4}$	263.37	$2.16\cdot10^{-4}$
4 pm	$8.03\cdot10^{-4}$	68.32	$5.26\cdot10^{-4}$	126.81	$2.47\cdot10^{-4}$
5 pm	$5.51\cdot10^{-4}$	62.58	$4.11\cdot10^{-4}$	129.71	$1.95\cdot10^{-4}$

6.8 Conclusions

This work extends the deep learning-based ORD from [38] to implement incremental Volt/-VAR control rules. A DNN-based digital twin is proposed that simulates the Volt/VAR dynamics of the incremental control rules. Results on step size selection and convergence rates, were presented for both single-phase and multiphase grids. These results were used to determine the minimum depths of the DNN implementing the digital twins. Numerical tests showed that optimal incremental control rules were able to achieve better objectives, and hence better voltage profiles, when compared to optimal non-incremental control rules, especially for 3ϕ feeders. These observations motivate further research in the direction of characterizing the equilibria non-incremental and incremental control rules for multiphase feeders.

6.9 Appendix

Proposition 6.1: From the rule (6.3b), we get

$$\begin{aligned}\|\mathbf{q}^t-\mathbf{q}^*\|_2 &= \|\mathbf{g}\left(\mathbf{y}^t\right)-\mathbf{g}\left(\mathbf{y}^*\right)\|_2\\ &\le \|\mathbf{y}^t-\mathbf{y}^*\|_2\end{aligned}$$

$$
\begin{aligned}
&= \| \operatorname{dg}(\tilde{\boldsymbol{\alpha}})(\mathbf{I} - \mu\mathbf{X}) \left(\mathbf{q}^{t-1} - \mathbf{q}^*\right) \|_2 \\
&\leq \| \operatorname{dg}(\tilde{\boldsymbol{\alpha}})\|_2 \cdot \|\mathbf{I} - \mu\mathbf{X}\|_2 \cdot \|\mathbf{q}^{t-1} - \mathbf{q}^*\|_2 \\
&\leq \|\mathbf{I} - \mu\mathbf{X}\|_2 \cdot \|\mathbf{q}^{t-1} - \mathbf{q}^*\|_2. \qquad (6.9)
\end{aligned}
$$

The first inequality stems from the non-expansive property of the proximal operator $\mathbf{g}$. The next equality follows from (6.3a). The second inequality from the sub-multiplicative property of the spectral norm. The last inequality follows by the definition of spectral norm and because $\tilde{\alpha}_n \leq 1$ for all n.

If $\|\mathbf{I} - \mu\mathbf{X}\|_2 < 1$, the inequality (6.9) yields (6.3) as a non-expansive mapping, establishing the stability of incremental rules and their convergence to $\mathbf{q}^*$. The inequality $\|\mathbf{I} - \mu\mathbf{X}\|_2 < 1$ holds true when $\mu < 2/\lambda_{\max}(\mathbf{X})$. Furthermore, the norm $\|\mathbf{I} - \mu\mathbf{X}\|_2$ achieves its minimum value of $\left(1 - \frac{2}{\kappa+1}\right)$ when $\mu = \frac{2}{\lambda_{\max}(\mathbf{X}) + \lambda_{\min}(\mathbf{X})}$. Plugging this value for μ in (6.9) and unfolding the dynamics over t provides

$$
\|\mathbf{q}^t - \mathbf{q}^*\|_2 \leq \left(1 - \tfrac{2}{\kappa+1}\right)^t \|\mathbf{q}^0 - \mathbf{q}^*\|_2 \leq 2\left(1 - \tfrac{2}{\kappa+1}\right)^t \|\hat{\mathbf{q}}\|_2
$$

To ensure that the voltage approximation error at time T is smaller than ϵ_1, we need

$$
\|\mathbf{v}^T - \mathbf{v}^*\|_2 \leq 2\|\mathbf{X}\|_2 \cdot \|\hat{\mathbf{q}}\|_2 \cdot \left(1 - \frac{2}{\kappa+1}\right)^T \leq \epsilon_1.
$$

This can be achieved by selecting T such that

$$
T \cdot \log\left(1 - \frac{2}{\kappa+1}\right) \leq \log \frac{\epsilon_1}{2\|\mathbf{X}\|_2 \|\hat{\mathbf{q}}\|_2}.
$$

multiplying both sides of the inequality by -1 gives

$$T \geq \frac{\log \frac{2\|\mathbf{X}\|_2\|\hat{\mathbf{q}}\|_2}{\epsilon_1}}{\log\left(1+\frac{2}{\kappa-1}\right)} \geq \left(\frac{\kappa-1}{2}\right)\log\frac{2\|\mathbf{X}\|_2\|\hat{\mathbf{q}}\|_2}{\epsilon_1}$$

where the last inequality follows from $\log(1+x) \leq x$.

$\square$

Chapter 7

A Quantum Approach for Stochastic Constrained Binary Optimization

7.1 Publication Details

S. Gupta and V. Kekatos.“ A quantum approach for stochastic constrained binary optimization”, *IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, Rhodes, Greece, Jun. 2023*, (submitted Oct. 2022).

7.2 Abstract

Analytical and practical evidence indicates the advantage of quantum computing solutions over classical alternatives. Quantum-based heuristics relying on the variational quantum eigensolver (VQE) and the quantum approximate optimization algorithm (QAOA) have been shown numerically to generate high-quality solutions to hard combinatorial problems, yet incorporating constraints to such problems has been elusive. To this end, this work puts forth a quantum heuristic to cope with stochastic binary quadratically constrained quadratic programs (QCQP). Identifying the strength of quantum circuits to efficiently generate samples from probability distributions that are otherwise hard to sample from, the variational

quantum circuit is trained to generate binary-valued vectors to approximately solve the aforesaid stochastic program. The method builds upon dual decomposition and entails solving a sequence of judiciously modified standard VQE tasks. Tests on several synthetic problem instances using a quantum simulator corroborate the near-optimality and feasibility of the method, and its potential to generate feasible solutions for the deterministic QCQP too.

7.3 Introduction

Quantum computers exhibit an innate ability to handle exponentially large computations in a parallel fashion yet with a strong probabilistic flavor. Quantum algorithms such as Shor's integer factorization, Grover's search, and the linear system solver of Harrow-Hassidim-Lloyd (HHL) can attain polynomial or even exponential speedups over the best known algorithms on classical computers [76]. Nonetheless, some of these quantum algorithms presume a large number of qubits on fault-tolerant quantum computers. In the *near-term intermediate scale* (NISQ) era, quantum computers are noisy and thus oftentimes limited in terms of number of gates and/or qubits. With such limitations in mind, *variational* quantum algorithms have been suggested as promising tools to practically showcase quantum advantage in the NISQ setup [85].

Variational quantum computers involve a sequence of parameterized gates. Their parameters are updated externally by a classical computer in a closed-loop fashion to steer the quantum state towards a desirable direction. The variational quantum eigensolver (VQE) used to provide high-quality solutions to combinatorial problems is a representative example. The Quantum Approximate Optimization Algorithm (QAOA) [23] is a special instance of VQE. In QAOA, not only the parameters but also the architecture of the quantum circuit become problem-dependent. The quantum circuit trained by QAOA operates as a sampler to efficiently generate near-optimal solutions of binary quadratic problems (e.g., MAXCUT); see [41] for a summary of claims on QAOA.

While most VQE/QAOA schemes target unconstrained problems, dealing with constraints is crucial to several applications in machine learning, wireless communications, and financial (stock trading) optimization. Adding constraints to QAOA or adiabetic quantum computing [67] (the QAOA counterpart for non-gate-based quantum computers) has been pursued in two ways. One approach has been to convert the constrained problem into an unconstrained minimization of a Lagrangian-like function [64, 77]. However, the weights for constraint penalties can be safely selected only if constraints are expressed as Boolean functions or linear equalities. An alternative approach modifies the architecture of the quantum circuit (via the mixer Hamiltonian of QAOA) to confine quantum states on the subspace spanned by constraints [40, 41, 42, 43]. Nonetheless, constructing such 'driver' mixer Hamiltonians is again highly problem-dependent and often limited to equality constraints. Reference [82] develops a quantum adiabetic approach to tackle binary linearly-constrained quadratic programs. It targets the dual problem and interfaces the quantum computer with a branch-and-bound scheme ran classically. Reference [28] treats mixed-binary quadratic-plus-convex problems using the alternating direction method of multipliers (ADMM) to split binary and continuous variables into separate minimizations, solved by QAOA and classical convex optimizers respectively per ADMM iteration.

Relation to prior work. Addressing binary QCQPs by quantum heuristics has been largely unexplored to the authors' knowledge. We put forth a quantum-based heuristic to solve a stochastic binary QCQP. Harnessing the power of quantum circuits to sample from probability mass functions (PMF) that are hard to sample classically, we devise a dual decomposition technique that solves a sequence of standard VQE tasks to systematically adjust Lagrangian multipliers. Numerical tests using quantum computer simulators provided by IBM evaluate this technique on randomly generated stochastic and deterministic binary QCQPs.

7.4 Quantum Computing Preliminaries

A quantum system consisting of n quantum bits (qubits) is described by an exponentially large state vector $|\mathbf{x}\rangle \in \mathbb{C}^N$ with $N = 2^n$ assuming the system is in a *pure state*. The Dirac notation $|\mathbf{x}\rangle$ named *ket* emphasizes that vector $\mathbf{x}$ is unit-norm or $\sum_{k=0}^{N-1} |x_k|^2 = 1$. If $\mathbf{e}_k$ is the k-th canonical vector of length N, we can write $|\mathbf{x}\rangle = \sum_{k=0}^{N-1} x_k |\mathbf{e}_k\rangle$. The vector $\mathbf{e}_k$ is oftentimes alternatively expressed as $|\mathbf{e}_k\rangle = |k\rangle$, where k is the binary representation of index k. For example, a system with $n = 2$ qubits has a state in $\mathbb{C}^4$, which is spanned by canonical vectors $\{\mathbf{e}_k\}_{k=0}^{3}$ and $\mathbf{e}_0 = [1\ 0\ 0\ 0]^\top = |00\rangle$. Vector $|\mathbf{x}\rangle$ provides a statistical characterization for the quantum state: If we measure the quantum system output, its qubits will be in configuration $|k\rangle$ with probability $|x_k|^2$ for all k. Symbol $\langle \mathbf{x}|$ termed *bra* denotes the conjugate transpose of $|\mathbf{x}\rangle$, while the *braket* $\langle \mathbf{x}|\mathbf{y}\rangle$ denotes the inner product between states.

The fundamental operations we can perform on a quantum system is evolution and measurement. The former can be described by the application of a unitary $\mathbf{U}$ on state $|\mathbf{x}\rangle$ to produce state $|\mathbf{y}\rangle = \mathbf{U}|\mathbf{x}\rangle$. Although $\mathbf{U}$ is exponentially large, it is usually implemented efficiently using quantum gates. Among various types of measurements, we focus on *projective measurements*. A projective measurement is associated with a Hermitian matrix (named *observable*) and its eigenvalue decomposition $\mathbf{H} = \sum_{m=1}^{M} \lambda_m \mathbf{v}_m \mathbf{v}_m^H$. If such measurement is performed on $|\mathbf{x}\rangle$, outcome m is observed with probability $p_m := |\langle \mathbf{x}|\mathbf{v}_m\rangle|^2$. Define a random variable taking value λ_m when outcome m is observed. The expected value of this variable is $\langle \mathbf{x}|\mathbf{H}|\mathbf{x}\rangle = \sum_{m=1}^{M} p_m \lambda_m$. If $\mathbf{H}$ is diagonal, the measurement is on the *computational basis.* This is practically important because now $\mathbf{v}_m = \mathbf{e}_m$, outcome m relates to $|m\rangle$, and each qubit can be measured individually.

If quantum system i has been prepared in state $|\mathbf{x}_i\rangle$ for $i = 1, 2$, their joint state would be $|\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle$, where $\otimes$ is the Kronecker product. This is oftentimes represented as $|\mathbf{x}_1\rangle\,|\mathbf{x}_2\rangle$ or $|\mathbf{x}_1, \mathbf{x}_2\rangle$. The Kronecker product rule generalizes to the composition of n systems. For example, $|1\rangle\,|1\rangle\,|0\rangle = \mathbf{e}_1 \otimes \mathbf{e}_1 \otimes \mathbf{e}_0 = \mathbf{e}_6 = |110\rangle$, where the canonical vectors shown in the middle are in $\mathbb{R}^2$ and those at the end are in $\mathbb{R}^8$.

7.5 Variational Quantum Eigensolver (VQE)

VQE is a heuristic approach to find near-optimal solutions for combinatorial problems of the general form

$$\min_{\mathbf{b}\in\{0,1\}^n} f(\mathbf{b}). \tag{7.1}$$

A particular example of interest is the quadratic unconstrained binary optimization (QUBO) problem with

$$f(\mathbf{b}) = \mathbf{b}^\top \mathbf{A}\mathbf{b} + \mathbf{b}^\top \mathbf{c} + d \tag{7.2}$$

which is known to be NP-hard. For later developments, it is convenient to reformulate QUBO in terms of the *spin* $\{\pm 1\}$ variables through the transformation

$$s_i = 1 - 2b_i = (-1)^{b_i} \quad \text{for} \quad i = 0, \ldots, n-1. \tag{7.3}$$

Collecting the spin variables in vector $\mathbf{s} = \mathbf{1} - 2\mathbf{b}$, the quadratic objective can be equivalently expressed as

$$f(\mathbf{b}) = \bar{f}(\mathbf{s}) = \mathbf{s}^\top \bar{\mathbf{A}}\mathbf{s} + \mathbf{s}^\top \bar{\mathbf{c}} + \bar{d} \tag{7.4}$$

where $\bar{\mathbf{A}} := \frac{1}{4}\mathbf{A}$; $\bar{\mathbf{c}} := -\frac{1}{2}(\mathbf{A}\mathbf{1} + \mathbf{c})$; and $\bar{d} := \frac{1}{4}\mathbf{1}^\top\mathbf{A}\mathbf{1} + \frac{1}{2}\mathbf{1}^\top\mathbf{c} + d$. We next explain how VQE samples high-quality solutions of (7.1) by solving an eigenvalue minimization task.

The VQE method falls under the family of *variational* quantum algorithms. The term *variational* pertains to the fact that the quantum circuit is not fixed, but parameterized by relatively few parameters collected in vector $\boldsymbol{\theta} \in \mathbb{R}^P$. These parameters are iteratively adjusted by classical computer in a closed-loop fashion so that the quantum system eventually reaches a desirable state. The process resembles the training of a neural network whose weights are updated by an optimization algorithm. Similarly to neural networks where the learner has to select an architecture (e.g., network depth/width and type of activations), the parameterized form (also termed *ansatz*) of the variational quantum circuit is specified *a priori*. We will be using a 2-local ansatz where single-qubit R_Y gates are applied to all qubits, followed by a full entanglement circuit, all repeated for 3 layers (iterations) [85].

Given $\boldsymbol{\theta}$ and driven by input $|0\rangle^n$, the quantum circuit produces at its output the quantum state $|\mathbf{x}(\boldsymbol{\theta})\rangle = \mathbf{U}(\boldsymbol{\theta})\,|0\rangle^n$ for a unitary $N \times N$ matrix $\mathbf{U}(\boldsymbol{\theta})$. To simplify notation, we will oftentimes write $|\mathbf{x}\rangle$ in lieu of $|\mathbf{x}(\boldsymbol{\theta})\rangle$. Albeit $|\mathbf{x}\rangle \in \mathbb{C}^N$ is exponentially long, it can be easily generated by the quantum circuit though it cannot be read out of the circuit as a vector in a computationally efficient manner. Instead, it is relatively easy to sample from it. Every time we run the quantum circuit driven by $|0\rangle^n$, we will be observing one of the binary outputs $|k\rangle = |\mathbf{e}_k\rangle$ with probability $p_k := |x_k|^2$ for $k = 0, \ldots, N-1$. The quantum circuit thus serves as an efficient sampler from the exponentially large probability mass function (PMF) $\{p_k\}_{k=0}^{N-1}$.

To exploit this sampling property, we next relate the cost $f(\mathbf{b})$ with a so-termed *Hamiltonian* matrix $\mathbf{H}$ so that

$$\mathbf{H}\,|\mathbf{e}_k\rangle = f(|k\rangle)\,|\mathbf{e}_k\rangle \quad \text{for all } k. \tag{7.5}$$

Matrix $\mathbf{H}$ is apparently diagonal and carries all N function evaluations $f(\mathbf{e}_k)$ on its diagonal. Moreover, the canonical vectors $\mathbf{e}_k$ constitute the eigenvectors of $\mathbf{H}$, each with corresponding eigenvalue $f(|k\rangle)$. Therefore, the minimization in (7.1) can be reformulated as the problem of finding the eigenvector corresponding to the minimum eigenvalue of $\mathbf{H}$

$$\min_{|\mathbf{x}\rangle} \langle \mathbf{x}| \, \mathbf{H} \, |\mathbf{x}\rangle \, . \tag{7.6}$$

As long as $|\mathbf{x}\rangle$ is allowed to take any of the values $\{\mathbf{e}_k\}_{k=0}^{N-1}$, the minimizer of (7.6) corresponds to the minimizer of (7.1). For example, if a quantum system has $n = 3$ qubits, its state would be $|\mathbf{x}\rangle \in \mathbb{C}^8$. Here $\mathbf{e}_k$'s are the columns of the identity matrix $\mathbf{I}_8$. If the minimizer of (7.6) is $|\mathbf{e}_5\rangle = |b_1 b_2 b_3\rangle = |101\rangle$, then the minimizer of (7.1) is $\mathbf{b} = [1\ 0\ 1]^\top$; and vice versa.

Although $\mathbf{H}$ is exponentially large, it can be implemented using only $\mathcal{O}(n^2)$ quantum gates since it can be expressed as

$$\mathbf{H} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \bar{A}_{ij} \mathbf{Z}_i \mathbf{Z}_j + \sum_{i=0}^{n-1} \bar{c}_i \mathbf{Z}_i + \bar{d} \mathbf{I}_N \tag{7.7}$$

where the $N \times N$ Hermitian matrix $\mathbf{Z}_i$ is defined as

$$\mathbf{Z}_i = \mathbf{I}_2 \otimes \cdots \otimes \mathbf{Z} \otimes \cdots \otimes \mathbf{I}_2 \quad \text{with} \quad \mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

This is a Kronecker product involving $(n-1)$ identity matrices $\mathbf{I}_2$ and one *Pauli-Z* operator $\mathbf{Z}$ applied to the i-th qubit. Matrix $\mathbf{H}$ as defined in (7.7) is obviously diagonal. To establish (7.5), note first that $\mathbf{Z}\,|0\rangle = |0\rangle$ and $\mathbf{Z}\,|1\rangle = -\,|1\rangle$, or more compactly, $\mathbf{Z}\,|b\rangle = (-1)^b\,|b\rangle$. Consequently, when $\mathbf{Z}_i$ is applied to a state $|\mathbf{b}\rangle = |b_1 b_2 \cdots b_n\rangle$, the effect is $\mathbf{Z}_i\,|\mathbf{b}\rangle = (-1)^{b_i}\,|\mathbf{b}\rangle = s_i\,|\mathbf{b}\rangle$ from (7.3). Similarly, it also holds that $\mathbf{Z}_i\mathbf{Z}_j\,|\mathbf{b}\rangle = s_i s_j\,|\mathbf{b}\rangle$. Property (7.5) now follows immediately by postmultiplying (7.7) by any $|\mathbf{e}_k\rangle$ and using

$f(\mathbf{b}) = \bar{f}(\mathbf{s})$.

If $|\mathbf{x}\rangle$ in (7.6) is restricted to set $\mathcal{E} := \{\mathbf{e}_k\}_{k=0}^{N-1}$, problem (7.6) is as hard as (7.1). VQE relaxes (7.6) to the set of all quantum states $|\mathbf{x}(\boldsymbol{\theta})\rangle$ that can be parameterized by the chosen ansatz and via $\boldsymbol{\theta}$. Problem (7.6) is then solved over $\boldsymbol{\theta}$ rather than $|\mathbf{x}\rangle$

$$\min_{\boldsymbol{\theta}} \; F(\boldsymbol{\theta}) := \langle \mathbf{x}(\boldsymbol{\theta}) | \mathbf{H} | \mathbf{x}(\boldsymbol{\theta}) \rangle \, . \tag{7.8}$$

From the eigenvalue property (7.5), it follows $\langle \mathbf{e}_n | \, \mathbf{H} \, | \mathbf{e}_k \rangle = f(|k\rangle)$ for all k. How about $\langle \mathbf{x} | \, \mathbf{H} \, | \mathbf{x} \rangle$ for a general state $|\mathbf{x}\rangle$? Because $|\mathbf{x}\rangle = \sum_{k=0}^{N-1} x_k \, |\mathbf{e}_k\rangle$, it is easy to show that

$$\langle \mathbf{x} | \mathbf{H} | \mathbf{x} \rangle = \sum_{k=0}^{N-1} |x_k|^2 f(|k\rangle) = \sum_{k=0}^{N-1} p_k f(|k\rangle). \tag{7.9}$$

In other words, function $F(\boldsymbol{\theta})$ is the average of f under the PMF defined by $|\mathbf{x}\rangle$. For instance, the random outcome $|k\rangle = |101\rangle$ occurring with probability $|x_5|^2$ is assigned to the random variable f taking the value $f([1 \; 0 \; 1]^\top)$. Hence, function $F(\boldsymbol{\theta})$ is really an expectation (an *observable* in the quantum computation parlance) of function $f(\mathbf{b})$ when $\mathbf{b}$ is drawn from the PMF $\{|x_k(\theta)|^2\}_{k=0}^{N-1}$. Ideally, the global minimizer $\boldsymbol{\theta}$ of (7.8) defines a PMF via $|\mathbf{x}(\boldsymbol{\theta})\rangle$ that samples with non-zero probability only the canonical vectors $|\mathbf{e}_k\rangle$ associated with the smallest eigenvalue of $\mathbf{H}$.

Problem (7.8) is solved in a hybrid fashion: The quantum computer samples from $|\mathbf{x}(\boldsymbol{\theta})\rangle$ and estimates $F(\boldsymbol{\theta})$ and possibly its gradient $\nabla_{\boldsymbol{\theta}} F$. A classical computer uses the previous information and iteratively updates $\boldsymbol{\theta}$ based on a zero- or first-order optimization algorithm, such as gradient descent or Bayesian optimization. As with training neural networks, $F(\boldsymbol{\theta})$ is nonconvex due to the form of the ansatz. Moreover, the ensemble statistic $F(\boldsymbol{\theta})$ cannot be computed exactly, but estimated as the sample average $\hat{F}(\boldsymbol{\theta}) := \sum_{r=1}^{R} f(\mathbf{b}_r)/R$ over R

runs, where $\mathbf{b}_r$ is the quantum output after run r.

7.6 Constrained VQE

As discussed earlier, VQE provides a successful heuristic for solving QUBO through the variational formulation of (7.8). Can VQE be generalized to deal with a binary QCQP of the ensuing form?

$$\begin{aligned} \min_{\mathbf{b}\in\{0,1\}^n} \quad & f_0(\mathbf{b}) \\ \text{s.to} \quad & f_m(\mathbf{b}) \leq 0, \quad m = 1:M. \end{aligned} \tag{7.10}$$

Here $f_m(\mathbf{b}) := \mathbf{b}^\top \mathbf{A}_m \mathbf{b} + \mathbf{b}^\top \mathbf{c}_m + d_m$ for $m = 0, \ldots, M$. Solving such problems is also known to be NP-hard. Providing a quantum heuristic to directly deal with (7.10) seems to be challenging. To this end, we relax expectations and aim at designing a quantum state $|\mathbf{x}\rangle$ from which we can draw binary-valued $\mathbf{b}$ that solve the *stochastic* binary QCQP:

$$\begin{aligned} \min_{|\mathbf{x}\rangle} \quad & \mathbb{E}_{\mathbf{x}}[f_0(\mathbf{b})] \\ \text{s.to} \quad & \mathbb{E}_{\mathbf{x}}[f_m(\mathbf{b})] \leq 0, \quad m = 1:M. \end{aligned} \tag{7.11}$$

As in the unconstrained setup, rather than minimizing over $|\mathbf{x}\rangle$, we propose optimizing over a PMF parameterized by $\boldsymbol{\theta}$ and captured by quantum state $|\mathbf{x}(\boldsymbol{\theta})\rangle$. Specifically, we suggest solving the constrained minimization

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & F_0(\boldsymbol{\theta}) \\ \text{s.to} \quad & F_m(\boldsymbol{\theta}) \leq 0 : \quad \lambda_m, \quad m = 1:M \end{aligned} \tag{7.12}$$

where each observable $F_m(\boldsymbol{\theta}) := \langle \mathbf{x}(\boldsymbol{\theta})|\mathbf{H}_m|\mathbf{x}(\boldsymbol{\theta})\rangle$ depends on the Hamiltonian $\mathbf{H}_m$ defined similar to $\mathbf{H}$ in (7.7) for all m. Heed that problem (7.12) can be reformulated and solved as a linear program (LP) over the PMF of $\mathbf{b}$. Nonetheless, that requires evaluating $\{f_m(\mathbf{b})\}_{m=0}^{M}$ for all 2^n values of $\mathbf{b}$. Moreover, the optimization variable of this LP is the vector of PMF values that is exponentially large too. That is also the case with standard VQE/QAOA.

Contrary to (7.10), problem (7.12) is over the continuous variable $\boldsymbol{\theta}$, and thus, we can associate a dual variable λ_m for each constraint and define its Lagrangian function

$$L(\boldsymbol{\theta};\boldsymbol{\lambda}) := F_0(\boldsymbol{\theta}) + \sum_{m=1}^{M} \lambda_m F_m(\boldsymbol{\theta}) \tag{7.13}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^M$ collects all dual variables. Problem (7.12) could be solved via *dual decomposition*, according to which $\boldsymbol{\lambda}$ is updated iteratively via a subgradient ascent step on L as

$$\lambda_m^{t+1} := \max\left\{\lambda_m^t + \mu_t F_m(\boldsymbol{\theta}^t), 0\right\}, \quad m = 1:M \tag{7.14}$$

for a positive step size $\mu_t = \mu_0/(t+\alpha)$ with $\alpha > 0$, and $\boldsymbol{\theta}^t$ is a minimizer of the Lagrangian $L(\boldsymbol{\theta};\boldsymbol{\lambda}^t)$ evaluated at $\boldsymbol{\lambda}^t$:

$$\boldsymbol{\theta}^t \in \arg\min_{\boldsymbol{\theta}} \langle \mathbf{x}(\boldsymbol{\theta})|\mathbf{H}_0 + \sum_{m=1}^{M} \lambda_m^t \mathbf{H}_m|\mathbf{x}(\boldsymbol{\theta})\rangle\,. \tag{7.15}$$

Problem (7.15) takes the QUBO form of (7.8), and is therefore amenable to standard VQE or even the celebrated QAOA approach. Under the latter, the ansatz takes a particular form that depends on the problem Hamiltonian $\mathbf{H}_0 + \sum_{m=1}^{M} \lambda_m^t \mathbf{H}_m$. Here, we used a problem-independent ansatz under the general VQE framework and leave QAOA for future work.

Table 7.1: Comparing the exact solution of (7.12) obtained via a linear program and the proposed quantum-based approach.

#	**Found PMF**		**Dual**	
	Quantum	LP	Quantum	LP
1	$[0.44, 0, 0.56, 0]$	$[0.44, 0, 0.56, 0]$	0.854	0.851
2	$[0.71, 0, 0.29, 0]$	$[0.70, 0, 0.30, 0]$	0.337	0.337
3	$[0, 0.80, 0, 0.20]$	$[0, 0.80, 0, 0.20]$	0.459	0.459
4	$[0, 0, 0.61, 0.39]$	$[0, 0, 0.60, 0.40]$	0.566	0.566

7.7 Numerical Tests

The novel solver for (7.12) was implemented in Python using the Qiskit library [5]. The *VQE* class in Qiskit was used to solve the minimization for the primal update (7.15). In addition to providing the ansatz described in Section 7.5, the VQE class was configured with the 'SLSQP' optimizer. The maximum number of iterations was set to $1,000$, and we used the `aer_simulator_statevector` quantum simulation backend. For the dual update in (7.14), constraint violations were measured over the observables $\mathbf{H}_m$ using the minimum eigenstate returned by VQE. The stopping criteria $\|\boldsymbol{\lambda}^t - \boldsymbol{\lambda}^{t-1}\|_2 \leq 1 \cdot 10^{-5}$ was utilized to ascertain the convergence of the dual updates (7.14).

To illustrate the application of the proposed strategy to solving the stochastic binary QCQP in (7.11), several 2-bit problem instances were sampled randomly by drawing the entries of $\{\mathbf{A}_0, \mathbf{c}_0, \mathbf{d}_0\}$ and $\{\mathbf{A}_1, \mathbf{c}_1, \mathbf{d}_1\}$ from the standard normal distribution, while ensuring the resulting problem was feasible. The VQE approach was compared against a linear program that finds a PMF solving (7.12); this was possible due to the small value of 2^n. For the two approaches, the obtained PMFs along with the associated dual variables are reported in Table 1 for 4 randomly sampled problem instances.

To study the scalability of the approach and to verify the compatibility of the solutions with

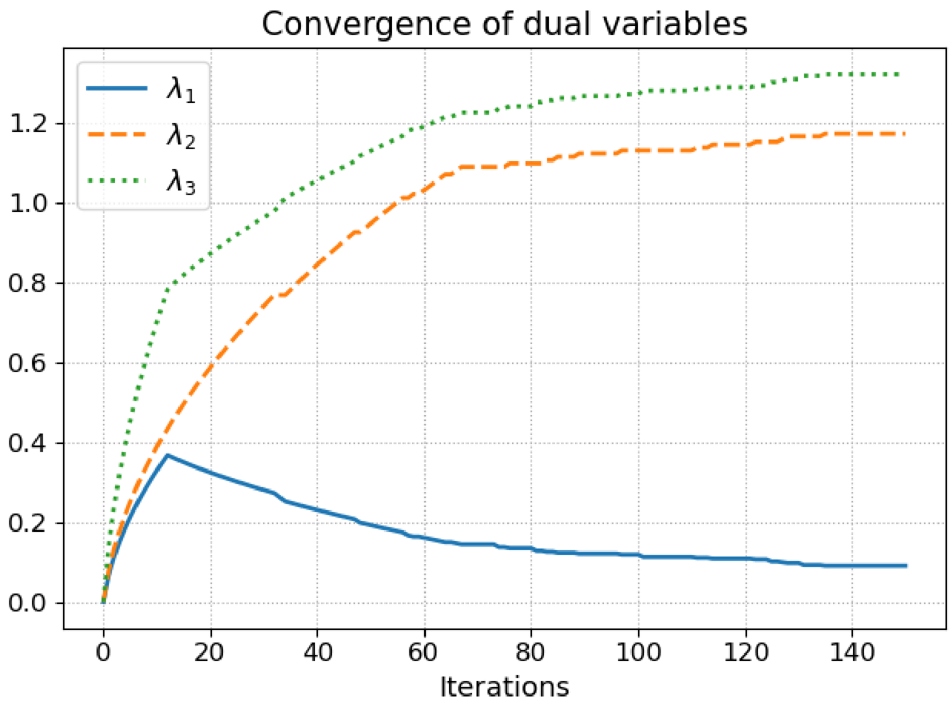


Figure 7.1: Convergence of dual variables under dual updates (7.14) for a stochastic binary QCQP with $M = 3$ constraints.

the deterministic QCQP in (7.10), we also sampled 30 feasible 5-bit problem instances with three constraints each. The quadratic cost and constraint functions were generated as in the previous test. To avoid instances with non-binding constraints, the constants d_m in the constraint functions were manually adjusted so that at least one of the constraints was active and yielded a non-zero dual variable. From the sampled problems, it was found that the dual decomposition involving VQE was able to produce the optimal solutions for 28 out of the 30 problem instances tested, whereas infeasible binary candidates were obtained for the remaining 2 instances. Figure 7.1 illustrates the convergence of the dual variables for one of the problem instances, where all three constraints were found to be active.

7.8 Conclusions

A novel generalization of VQE to address the need for dealing with stochastic binary QCQPs has been developed. Leveraging dual decomposition, the approach entails solving a sequence of judiciously modified VQE tasks. Numerical tests demonstrate that upon convergence of the constrained VQE algorithm, the variational quantum circuit is able to sample from a stochastic policy to generate binary-valued vectors that minimize the binary QCQP and satisfy its constraints in expectation. Some of these samples seem to be feasible for the deterministic binary QCQP too. This novel heuristic sets the foundation for further developments towards constrained discrete optimization. We are currently exploring several exciting directions: *i)* Coupling this approach with QAOA rather than VQE; *ii)* skipping the nested optimization in (7.15) through a primal-dual decomposition alternative as in [33, 34]; and *iii)* dealing with mixed-binary setups.

Bibliography

[1] Pytorch. URL https://pytorch.org/blog/computational-graphs-constructed-in-pytorch/.

[2] Oct2py: Python to gnu octave bridge. URL https://oct2py.readthedocs.io/en/latest/index.html.

[3] Ieee 1547 standard for interconnecting distributed resources with electric power systems, 2014. URL http://grouper.ieee.org/groups/scc21/1547/1547_index.html.

[4] Ieee standard for interconnection and interoperability of distributed energy resources with associated electric power systems interfaces, 2018. URL https://standards.ieee.org/ieee/1547/5915/.

[5] Qiskit: An open-source framework for quantum computing, 2021. URL https://qiskit.org/.

[6] K. Baker, A. Bernstein, E. Dall'Anese, and C. Zhao. Network-cognizant voltage droop control for distribution grids. *IEEE Trans. Power Syst.*, 33(2):2098–2108, March 2018.

[7] Kyri Baker, Andrey Bernstein, Emiliano Dall'Anese, and Changhong Zhao. Network-cognizant voltage droop control for distribution grids. *IEEE Transactions on Power Systems*, 33(2):2098–2108, 2018. doi: 10.1109/TPWRS.2017.2735379.

[8] M.E. Baran and F.F. Wu. Optimal sizing of capacitors placed on a radial distribution system. *IEEE Trans. Power Syst.*, 4(1):735–743, January 1989.

[9] Sean Barker, Aditya Mishra, David Irwin, Emmanuel Cecchet, Prashant Shenoy, and Jeannie Albrecht. Smart*: An open data set and tools for enabling research in sus-

tainable homes. In *Workshop on Data Mining Applications in Sustainability*, pages 1–6, Beijing, China, August 2012.

[10] Daniel Bienstock, Michael Chertkov, and Sean Harnett. Chance-constrained optimal power flow: Risk-aware network control under uncertainty. *Siam Review*, 56(3):461–495, 2014.

[11] Vito Calderaro, Gaspare Conio, Vincenzo Galdi, Giovanni Massa, and Antonio Piccolo. Optimal decentralized voltage control for distribution systems with inverter-based distributed generators. *IEEE Transactions on Power Systems*, 29(1):230–241, Jan 2014.

[12] Guido Cavraro, Saverio Bolognani, Ruggero Carli, and Sandro Zampieri. The value of communication in the voltage regulation problem. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 5781–5786, 2016.

[13] C. Chen and O. L. Mangasarian. Smoothing methods for convex inequalities and linear complementarity problems. *Mathematical Programming*, 71(1):51–69, 1995.

[14] Dong Chen, Srinivasan Iyengar, David Irwin, and Prashant Shenoy. Sunspot: Exposing the location of anonymous solar-powered homes. In *ACM International Conference on Systems for Energy-Efficient Built Environments*, page 85–94, Palo Alto, CA, Nov 2016.

[15] Yize Chen and Baosen Zhang. Learning to solve network flow problems via neural decoding. *arXiv preprint arXiv:2002.04091*, 2020.

[16] Wenqi Cui, Jiayi Li, and Baosen Zhang. Decentralized safe reinforcement learning for inverter-based voltage control. *Electric Power Systems Research*, 211:108609, 2022.

[17] Emiliano Dall'Anese, Kyri Baker, and Tyler Summers. Chance-constrained AC optimal

power flow for distribution systems with renewables. *IEEE Trans. Power Syst.*, 32(5): 3427–3438, 2017.

[18] D. Deka and S. Misra. Learning for DC-OPF: Classifying active sets using neural nets. In *IEEE PowerTech*, pages 1–6, Milan, Italy, June 2019.

[19] R. Dobbe, O. Sondermeijer, D. Fridovich-Keil, D. Arnold, D. Callaway, and C. Tomlin. Towards distributed energy services: Decentralizing optimal power flow with machine learning. *IEEE Trans. Smart Grid*, 11(2):1296–1306, March 2020.

[20] Priya L. Donti, David Rolnick, and J Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, May 2021.

[21] John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. GNU Octave version 6.1.0 manual: a high-level interactive language for numerical computations. 2020. URL https://www.gnu.org/software/octave/doc/v6.1.0/.

[22] M. Eisen, C. Zhang, L. F. O. Chamon, D. D. Lee, and A. Ribeiro. Learning optimal resource allocations in wireless systems. *IEEE Trans. Signal Process.*, 67(10):2775–2790, May 2019.

[23] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem. *arXiv: Quantum Physics*, 2014.

[24] M. Farivar, R. Neal, C. Clarke, and S. Low. Optimal inverter VAR control in distribution systems with high PV penetration. In *Proc. IEEE Power & Energy Society General Meeting*, pages 1–7, San Diego, CA, July 2012.

[25] M. Farivar, X. Zhou, and L. Chen. Local voltage control in distribution systems: An incremental control algorithm. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, Miami, FL, November 2015.

[26] Masoud Farivar, Lijun Chen, and Steven Low. Equilibrium and dynamics of local voltage control in distribution systems. In *52nd IEEE Conference on Decision and Control*, pages 4329–4334, Dec 2013.

[27] Masoud Farivar, Lijun Chen, and Steven Low. Equilibrium and dynamics of local voltage control in distribution systems. In *Proc. IEEE Conf. on Decision and Control*, pages 4329–4334, Florence, Italy, December 2013.

[28] Claudio Gambella and Andrea Simonetto. Multiblock ADMM heuristics for mixed-binary optimization on classical and quantum computers. *IEEE Trans. on Quantum Engineering*, 1:1–22, 10 2020.

[29] Neel Guha, Zhecheng Wang, Matt Wytock, and Arun Majumdar. Machine learning for AC optimal power flow. In *Climate Change Workshop at ICML*, pages 1–4, Long Beach, CA, June 2019.

[30] S. Gupta and V. Kekatos. Real-time operation of heterogeneous energy storage units. In *Proc. IEEE Global Conf. on Signal and Inf. Process.*, Washington, DC, December 2016.

[31] S. Gupta, V. Kekatos, and W. Saad. Optimal real-time coordination of energy storage units as a voltage-constrained game. *IEEE Trans. Smart Grid*, 10(4):3883–3894, July 2019.

[32] S. Gupta, V. Kekatos, and M. Jin. Machine learning for communication-cognizant

smart inverter control. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Tempe, AZ, November 2020.

[33] S. Gupta, S. Misra, D. Deka, and V. Kekatos. DNN-based policies for stochastic AC-OPF. Porto, Portugal, June 2021. URL https://www.faculty.ece.vt.edu/kekatos/papers/PSCC2022a.pdf. (also published in the Elsevier Electric Power Systems Research).

[34] S. Gupta, V. Kekatos, and M. Jin. Controlling smart inverters using proxies: A chance-constrained DNN-based approach. *IEEE Trans. Smart Grid*, 13(2):1310–1321, March 2022.

[35] Sarthak Gupta and Vassilis Kekatos. A quantum approach for stochastic constrained binary optimization. (submitted), 2022.

[36] Sarthak Gupta and Vassilis Kekatos. Scalable optimal design of incremental volt/-var control using deep neural networks. *IEEE Contr. Syst. Lett.*, December 2022. (submitted).

[37] Sarthak Gupta, Vassilis Kekatos, and Ming Jin. Deep learning for reactive power control of smart inverters under communication constraints. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Tempe, AZ, 2020.

[38] Sarthak Gupta, Spyros Chatzivasileiadis, and Vassilis Kekatos. Deep learning for optimal volt/var control using distributed energy resources. (submitted), 2022. URL http://arxiv.org/abs/2211.09557.

[39] Sarthak Gupta, Sidhant Misra, Deepjyoti Deka, and Vassilis Kekatos. Dnn-based policies for stochastic ac opf. *Electric Power Systems Research*, 213:108563, 2022.

[40] S. Hadfield, Z.Wang, E. G. Rieffel, B. O'Gorman, D. Venturelli, and R. Biswas. Quantum approximate optimization with hard and soft constraints. In *ACM Intl. Workshop on Post Moore's Era Supercomputing*, pages 15–21, New York, NY, 2017.

[41] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, 2019.

[42] I. Hen and M. S. Sarandy. Driver Hamiltonians for constrained optimization in quantum annealing. *Phys. Rev. A*, 93(6):062312, 2016.

[43] I. Hen and F. M. Spedalieri. Quantum annealing for constrained optimization. *Phys. Rev. Appl.*, 5(63):034007, 2016.

[44] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. ISSN 0893-6080.

[45] 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*, Jul. 2021.

[46] R. A. Jabr. Linear decision rules for control of reactive power by distributed photovoltaic generators. *IEEE Trans. Power Syst.*, 33(2):2165–2174, March 2019.

[47] Rabih A. Jabr. Segregated linear decision rules for inverter Watt-VAr control. *IEEE Trans. Power Syst.*, 36(3):2702–2708, 2021. doi: 10.1109/TPWRS.2020.3032467.

[48] M. Jalali, V. Kekatos, N. Gatsis, and D. Deka. Designing reactive power control rules for smart inverters using support vector machines. *IEEE Trans. Smart Grid*, 11(2): 1759–1770, March 2020.

[49] M. Jalali, M. K. Singh, V. Kekatos, G. B. Giannakis, and C. C. Liu. Fast inverter control by learning the OPF mapping using sensitivity-informed Gaussian processes. *IEEE Trans. Smart Grid*, 2022. (early access).

[50] S. Karagiannopoulos, P. Aristidou, and G. Hug. Data-driven local control design for active distribution grids using off-line optimal power flow and machine learning techniques. *IEEE Trans. Smart Grid*, 10(6):6461–6471, November 2019.

[51] V. Kekatos, G. Wang, A. J. Conejo, and G. B. Giannakis. Stochastic reactive power management in microgrids with renewables. *IEEE Trans. Power Syst.*, 30(6):3386–3395, November 2015.

[52] V. Kekatos, L. Zhang, G. B. Giannakis, and R. Baldick. Fast localized voltage regulation in single-phase distribution grids. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 725–730, Miami, FL, October 2015.

[53] V. Kekatos, L. Zhang, G. B. Giannakis, and R. Baldick. Voltage regulation algorithms for multiphase power distribution grids. *IEEE Trans. Power Syst.*, 31(5):3913–3923, September 2016.

[54] V. Kekatos, G. Wang, H. Zhu, and G. B. Giannakis. PSSE redux: Convex relaxation, decentralized, robust, and dynamic approaches. In Mo El-Hawary, editor, *Advances in Electric Power and Energy*. Wiley IEEE Press, 2018. URL http://www.faculty.ece.vt.edu/kekatos/papers/PSSE-Redux.pdf.

[55] Vassilis Kekatos, Gang Wang, and Georgios B. Giannakis. Stochastic loss minimization for power distribution networks. In *Proc. North American Power Symposium*, pages 1–6, Pullman, WA, September 2014.

[56] Dongchan Lee, Konstantin Turitsyn, Daniel Kenneth Molzahn, and Line Roald. Robust AC optimal power flow with robust convex restriction. *IEEE Trans. Power Syst.*, 2021.

[57] X. Lei, Z. Yang, J. Yu, J Zhao, Q. Gao, , and H.Yu. Data-driven optimal power flow: A physics-informed machine learning approach. *IEEE Trans. Power Syst.*, 36 (1):346–354, January 2021.

[58] Na Li, Guannan Qu, and Munther Dahleh. Real-time decentralized voltage control in distribution networks. In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 582–588, Allerton, IL, October 2014.

[59] W. Lin and E. Bitar. Decentralized stochastic control of distributed energy resources. *IEEE Trans. Power Syst.*, 33(1):888–900, January 2018.

[60] J. Lofberg. A toolbox for modeling and optimization in MATLAB. In *Proc. of the CACSD Conf.*, 2004. URL http://users.isy.liu.se/johanl/yalmip/.

[61] L. M. Lopez-Ramos, V. Kekatos, A. G. Marques, and G. B. Giannakis. Two-timescale stochastic dispatch of smart distribution grids. *IEEE Trans. Smart Grid*, 9(5):4282–4292, September 2018.

[62] A. Lorca and X.A. Sun. The Adaptive Robust Multi-Period Alternating Current Optimal Power Flow Problem. *IEEE Trans. Power Syst.*, 33(2):1993–2003, 2018.

[63] Steven Low. Convex relaxation of optimal power flow – Part II: Exactness. *IEEE Trans. Control of Network Systems*, 1(2):177–189, June 2014.

[64] A. Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2(5):1–15, 2014.

[65] Sindri Magnússon, Guannan Qu, Carlo Fischione, and Na Li. Voltage control using limited communication. *IEEE Transactions on Control of Network Systems*, 6(3): 993–1003, September 2019.

[66] Fernando Mancilla-David, Alejandro Angulo, and Alexandre Street. Power management in active distribution systems penetrated by photovoltaic inverters: A data-driven robust approach. *IEEE Trans. Smart Grid*, 11(3):2271–2280, May 2020.

[67] C. C. McGeoch. *Adiabatic quantum computation and quantum annealing: Theory and practice*, volume 5. Springer, Switzerland, 2014.

[68] David Métivier, Marc Vuffray, and Sidhant Misra. Efficient polynomial chaos expansion for uncertainty quantification in power systems. *Electric Power Systems Research*, 189: 106791, 2020.

[69] I. Mezghani, S. Misra, and D. Deka. Stochastic AC optimal power flow: A data-driven approach. *Electric Power Systems Research*, 189, Dec 2020.

[70] Tillmann Mühlpfordt, Line Roald, Veit Hagenmeyer, Timm Faulwasser, and Sidhant Misra. Chance-constrained AC optimal power flow: A polynomial chaos approach. *IEEE Trans. Power Syst.*, 34(6), 2019.

[71] I. Murzakhanov, S. Gupta, S. Chatzivasileiadis, and V. Kekatos. Optimal design of Volt/VAR control rules for inverter-interfaced distributed energy resources. *IEEE Trans. Smart Grid*, 2023. URL https://arxiv.org/abs/2210.12805. (submitted).

[72] Ilgiz Murzakhanov, Sarthak Gupta, Spyros Chatzivasileiadis, and Vassilis Kekatos. Optimal design of volt/var control rules for inverter-interfaced distributed energy resources. (submitted), 2022. URL http://arxiv.org/abs/2210.12805.

[73] Arkadi Nemirovski and Alexander Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2007.

[74] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[75] 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.

[76] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[77] M. Ohzeki. Breaking limitation of quantum annealer in solving optimization problems under constraints. *Scientific reports*, 10(1):1–12, 2020.

[78] Damian Owerko, Fernando Gama, and Alejandro Ribeiro. Optimal power flow using graph neural networks. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Process.*, pages 5930–5934, Barcelona, Spain, May 2020.

[79] Xiang Pan, Tianyu Zhao, and Minghua Chen. DeepOPF: Deep neural network for DC optimal power flow. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Beijing, China, October 2019.

[80] Line Roald, Frauke Oldewurtel, Thilo Krause, and Göran Andersson. Analytical reformulation of security constrained optimal power flow with probabilistic constraints. In *2013 IEEE Grenoble Conference*, pages 1–6. IEEE, 2013.

[81] Line Roald, Sidhant Misra, Michael Chertkov, and Göran Andersson. Optimal power flow with weighted chance constraints and general policies for generation control. In *Proc. IEEE Conf. on Decision and Control*, pages 6927–6933. IEEE, 2015.

[82] Pooya Ronagh, Brad Woods, and Ehsan Iranmanesh. Solving constrained quadratic binary problems via quantum adiabatic evolution. *Quantum Info. Comput.*, 16(11–12): 1029–1047, September 2016.

[83] Afshin Samadi, Robert Eriksson, Lennart Söder, Barry G. Rawn, and Jens C. Boemer. Coordinated active power-dependent voltage regulation in distribution grids with pv systems. *IEEE Transactions on Power Delivery*, 29(3):1454–1464, Jun 2014.

[84] Juan Sepulveda, Alejandro Angulo, Fernando Mancilla-David, and Alexandre Street. Robust co-optimization of droop and affine policy parameters in active distribution systems with high penetration of photovoltaic generation. *IEEE Trans. Smart Grid*, pages 1–1, 2022. doi: 10.1109/TSG.2022.3177947.

[85] Osvaldo Simeone. An introduction to quantum machine learning for engineers. *Foundations and Trends in Signal Processing*, 16(1–2):1–223, 2022.

[86] M. K. Singh, S. Gupta, V. Kekatos, G. Cavraro, and A. Bernstein. Learning to optimize power distribution grids using sensitivity-informed deep neural networks. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Tempe, AZ, November 2020.

[87] M. K. Singh, V. Kekatos, and G. B. Giannakis. Learning to solve the AC-OPF using sensitivity-informed deep neural networks. *IEEE Trans. Power Syst.*, 2021. URL https://arxiv.org/abs/2103.14779. submitted.

[88] M. K. Singh, V. Kekatos, and G. B. Giannakis. Learning to solve the AC-OPF using sensitivity-informed deep neural networks. *IEEE Trans. Power Syst.*, 37(4):2833–2846, July 2022.

[89] M. K. Singh, S. Taheri, V. Kekatos, K. P. Schneider, and C.-C. Liu. Joint grid topology

reconfiguration and design of Watt-VAR curves for DERs. In *Proc. IEEE Power & Energy Society General Meeting*, Denver, CO, July 2022.

[90] Ankit Singhal, Venkataramana Ajjarapu, Jason Fuller, and Jacob Hansen. Real-time local volt/var control under external disturbances with high pv penetration. *IEEE Transactions on Smart Grid*, 10(4):3849–3859, Jul 2019.

[91] S. Taheri, M. Jalali, V. Kekatos, and L. Tong. Fast probabilistic hosting capacity analysis for active distribution systems. *IEEE Trans. Smart Grid*, 2020. URL https://arxiv.org/abs/2002.01980. (submitted).

[92] S. Taheri, M. Jalali, V. Kekatos, and L. Tong. Fast probabilistic hosting capacity analysis for active distribution systems. *IEEE Trans. Smart Grid*, 12(3):2000–2012, May 2021.

[93] S. Taheri, M. Jalali, V. Kekatos, and L. Tong. Fast probabilistic hosting capacity analysis for active distribution systems. *IEEE Trans. Smart Grid*, (3):2000–2012, May 2021.

[94] Konstantin Turitsyn, P. Sulc, Scott Backhaus, and Michael Chertkov. Options for control of reactive power by distributed photovoltaic generators. *Proc. IEEE*, 99(6): 1063–1073, June 2011.

[95] A. Velloso and V.H. Pascal. Combining deep learning and optimization for preventive security-constrained DC optimal power flow. *IEEE Trans. Power Syst.*, 36(4):3618–3628, July 2021.

[96] Maria Vrakopoulou, Kostas Margellos, John Lygeros, and Göran Andersson. A probabilistic framework for reserve scheduling and security assessment of systems with high wind power penetration. *IEEE Trans. Power Syst.*, 28(4):3885–3896, 2013.

[97] G. Wang, V. Kekatos, A.-J. Conejo, and G. B. Giannakis. Ergodic energy management leveraging resource variability in distribution grids. *IEEE Trans. Power Syst.*, 31(6), November 2016.

[98] W. Wang, N. Yu, Y. Gao, and J. Shi. Safe off-policy deep reinforcement learning algorithm for Volt-VAR control in power distribution systems. *IEEE Trans. Smart Grid*, 11(4):3008–3018, July 2020.

[99] Qiuling Yang, Gang Wang, Alireza Sadeghi, Gang Wang, Georgios B. Giannakis, and Jian Sun. Real-time voltage control using deep reinforcement learning. *IEEE Trans. Smart Grid*, 11(3):2313–2323, May 2020.

[100] Y. Yang, Z. Yang, J. Yu, B. Zhang, Y. Zhang, and H. Yu. Fast calculation of probabilistic power flow: A model-based deep learning approach. *IEEE Trans. Smart Grid*, 11(3):2235–2244, May 2020.

[101] Ahmed Zamzam and Kyri Baker. Learning optimal solutions for extremely fast AC optimal power flow. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6, Tempe, AZ, November 2020.

[102] Ahmed S Zamzam and Kyri Baker. Learning optimal solutions for extremely fast ac optimal power flow. In *Proc. IEEE Intl. Conf. on Smart Grid Commun.*, pages 1–6. IEEE, 2020.

[103] Baosen Zhang, A.D. Dominguez-Garcia, and D. Tse. A local control approach to voltage regulation in distribution networks. In *Proc. North American Power Symposium*, Manhattan, KS, September 2013.

[104] Qianzhi Zhang, Kaveh Dehghanpour, Zhaoyu Wang, Feng Qiu, and Dongbo Zhao.

Multi-agent safe policy learning for power management of networked microgrids. *IEEE Trans. Smart Grid*, 12(2):1048–1062, March 2021.

[105] Yu Zhang, N. Gatsis, and G. B. Giannakis. Robust energy management for microgrids with high-penetration renewables. *IEEE Trans. Sustain. Energy*, 4(4):944–953, October 2013.

[106] Xinyang Zhou, Masoud Farivar, and Lijun Chen. Pseudo-gradient based local voltage control in distribution networks. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 173–180, 2015.

[107] Xinyang Zhou, Jie Tian, Lijun Chen, and Emiliano Dall'Anese. Local voltage control in distribution networks: A game-theoretic perspective. In *2016 North American Power Symposium (NAPS)*, pages 1–6, Nov 2016.

[108] Xinyang Zhou, Masoud Farivar, Zhiyuan Liu, Lijun Chen, and Steven H. Low. Reverse and forward engineering of local voltage control in distribution networks. *IEEE Trans. Autom. Contr.*, 66(3):1116–1128, 2021. doi: 10.1109/TAC.2020.2994184.

[109] Hao Zhu and Hao Jan Liu. Fast local voltage control under limited reactive power: Optimality and stability analysis. *IEEE Trans. Power Syst.*, 31(5):3794–3803, 2016. doi: 10.1109/TPWRS.2015.2504419.

[110] Ray D Zimmerman and Carlos E Murillo-s. MATPOWER, Version 7.1, Power Systems Engineering Research Center (PSERC). pages 1–250, 2020.