Deep Unsupervised Learning for Network Resource Allocation Problems with Convex and Non-Convex Constraints

Mehrazin Alizadeh

A thesis submitted to the
Department of Electrical Engineering and Computer Science
in conformity with the requirements for
the degree of Master of Applied Science

YORK UNIVERSITY
TORONTO, ONTARIO

November 2022

Abstract

The next generation of wireless networks is anticipated to be more complex and heterogeneous due to higher transmission frequencies, massive antenna deployments, ultra-dense access points, etc. Subsequently, optimized network planning and radio resource management (RRM) solutions are critical for systematic resource utilization, attaining data rate or quality-of-service (QoS) requirements, and minimizing network capital and operational expenditures. Deep learning-enabled RRM is emerging as a potential solution to reduce the time complexity of conventional optimization-based solutions to constrained RRM problems. Since the unsupervised learning solutions do not require high-quality training labels, this thesis will focus on developing a comprehensive deep unsupervised learning framework for QoS-constrained RRM problems (e.g., user assignment, power control, and beamforming).

In the first part of this thesis, I developed a framework to solve the classical power control problem with the QoS constraints using deep unsupervised learning with the objective of maximizing the network sum-rate. Utilizing a differentiable projection function, the proposed neural network outputs solutions that are always feasible. Two approaches are pursued to define the projection function implicitly and explicitly using mathematical optimization and an iterative process, respectively. The proposed methods outperform the solutions of conventional neural network-based benchmarks,

such as PCNet, in terms of the data rate and constraint violation probability.

In the second part of the thesis, the problem of joint power allocation and user association to maximize the instantaneous sum-rate is addressed with users' QoS and base station quota constraints. User association problem is first transformed into an equivalent linear program. The QoS constraint is handled by using a technique called masking, which prevents the neural network to output an infeasible solution. Similar to power control, two projection functions are designed using mathematical optimization and an iterative process to systematically handle the base station quota constraint. A joint neural network-based solver is designed by cascading the user association and power control neural networks. Experimental results showed the superior performance of the proposed methods compared to conventional neural network-based solvers and optimization-based benchmarks.

The source code for this thesis document is available at:

https://github.com/Mehrazin/thesis

Acknowledgments

I'd like to thank all the people that supported me throughout the journey of writing this thesis.

First and foremost I would like to extend my deepest gratitude to my supervisor Professor Hina Tabassum for her enormous support during my study and for enhancing my knowledge in the field of wireless communications. Her reviews, insightful comments, and encouragement aided me in developing the contents of my research work. Furthermore, I would like to thank my thesis committee members, Professor Marcus Brubaker and Professor Varvara Nika, for reviewing my thesis and sharing their valuable feedback.

I would like to extend my sincere thanks to alumni and current members of NWGN lab for their encouragement and support: Mohammed Amin Saeidi, Javad Sayehvand, Jeet Tilara, Zijiang Yan, Sadeq bani Melhem, Haider Shoaib, Arjun Kaushik, Sheyda Zarandi. Their support and feedback during our group meetings helped me to improve the quality of my work.

I'd also like to thank my friends for their genuine help and support throughout my graduate studies: Hamidreza Mohabbat, Amirhossein Elmi, Mohammad Ehsan Abdollahi, Kourosh Heidari Kani, Niloofar Namavari, Amin Fadaeinejad, Farid Ehsani, Komeil Jalali, Mohammadreza Sadeghi, Mohammadreza Nafissi, Tina Behrozi, Parsa

Hazrati, Mohamadreza Moradi, Sina Ahmadi Jazani, Mahdi Mousavi. Without their encouragement and emotional support, it's difficult to imagine how I would have survived the foreign student experience.

But most of all, I want to express my deepest gratitude to my parents, Marzieh and Mahmoud, for raising me to value education and for their heartfelt blessings and support throughout my life. I'd also like to thank my uncles, Hadi and Reza, for their guidance and unconditional help and support. Lastly, I like to thank my grandparents for their blessings and encouragement throughout my graduate studies.

Contents

List of Figures

Chapter 1

Introduction

Optimized network planning and radio resource management (RRM) solutions have remained an integral part of wireless network design since their evolution. The efficiency of RRM solutions is crucial for systematic resource utilization, attaining quality-of-service (QoS) requirements, and minimizing network capital and operational expenditures. Nevertheless, the upcoming generation of wireless networks is anticipated to be more complex than ever due to extremely high-frequency (EHF) communication, massive antenna deployments, ultra-dense access points, etc. [3]. On the one hand, the increasing heterogeneity of wireless networks produces higher degrees of freedom (e.g., different types of spectrum, massive antennas, beamforming and power allocation, computational offloading decisions, etc.) which significantly increases the size of RRM problems. On the other hand, the short channel coherence time (the time during which the transmission channel stays the same) of EHF links call for speedy and efficient RRM solutions. Thus, the immediate redesign of conventional RRM solutions is warranted.

In what follows, the fundamental RRM problems in wireless communications are explained in detail.

1.1 Resource Management in Wireless Networks

Resource management can be described as a process in which the amount of resources allocated to each wireless user will be determined [4]. There are various types of network resources (e.g., spectrum, power, antennas, etc.) to support a large number of users while satisfying their QoS requirements. Efficient and optimal resource management can play a huge role in the quality of communications. To date, the most commonly used technique for formalizing RRM problems is mathematical optimization [4] where a predefined objective function is generally maximized or minimized over the feasible space of the resource variables characterized by various constraints [5]. Common objectives include minimizing the energy consumption [6], maximizing the aggregate network data rate [6], etc.

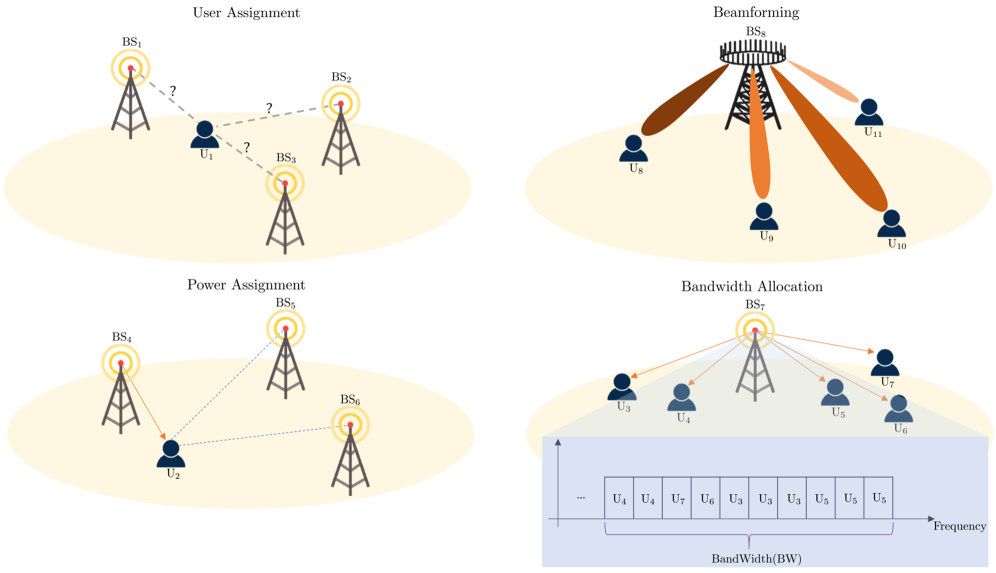

Figure 1.1: An illustration of four fundamental radio resource management problems in wireless networks.

In most wireless communication models, communication happens in a coherence time-slot during which the channel conditions remain steady. The coherence interval

varies based on the users' mobility and environment characteristics [3, 4]. In each time-slot, channel estimation happens at first and resource allocation is conducted afterward by solving a predefined RRM problem given the estimated channel state information (CSI). Once the resources are allocated, data transmission starts to take place [4]. The objective and constraints in RRM problems can be classified as *instantaneous* and *statistical* [7]. Instantaneous constraints refer to the conditions that need to be satisfied in the current time slot, like the power budget constraints or instantaneous rate requirements, while statistical constraints capture the conditions that have to be met over time, e.g., fairness of connectivity in user scheduling, where each user has to receive a fair amount of spectrum resources on average [7].

In the following, the four fundamental RRM problems are described.

1.1.1 User Assignment

In every wireless communication scenario, there are transmitters and receivers. Before any communication happens, it should be determined which transmitter is going to be connected to which receiver. This task is called user assignment and needs to happen less frequently than other resource allocations. It basically happens when the relative position of the transmitter and its receiver changes over time due to mobility, or if the channel quality is no longer appropriate given the users' data rate requirements. For example, in the cellular network, there are users and base stations (BSs), and the task of user assignment is to determine the set of users assigned to each BS for communication purposes. Fig. 1.1 (top-left) shows a graphical example of user assignment where user U_1 is going to be assigned to one of the available base stations in his vicinity. Once the assignment happened, in each time slot, each BS only considers

its preassigned users for resource allocation and in turn data communications.

1.1.2 Power Assignment

Transmit power is one of the most important aspects of the transmitted signal. Transmit power is the amount of power (in Watts) that a transmitter allocates to the communication link [4]. In a wireless medium, signal propagates in many directions and as the distance from the transmitter increases the received signal power decreases, due to free space path-loss, obstacles, etc. Thus, the higher the transmit power the further a signal can propagate into the space [4]. A rule of thumb is that the more transmit power is allocated to a wireless link, the more powerful and reliable it becomes [4]. However, each wireless transmitter has a maximum power budget constraint which is determined by its configuration and how it was manufactured [8].

In addition, in the shared wireless setup, where the medium is shared, the high transmit powers result in interference. This is because of the limitation of resources in code, time, and frequency, which makes a group of transceivers use the same wireless resource to communicate over. As a result, every receiver will experience interference from the other transmitters, which is going to be worse if they decide to transmit with the highest power without any consideration [4,8]. Fig. 1.1 (bottom-left) shows an example of data transmission, where user U_2 gets the desired signal from BS_4 and interfering signals from BS_5 and BS_6. Lastly, every receiver has its own expectation from the communication service in terms of the transmission rate and since there is a direct connection between the assigned transmit powers in a communication session and the achievable rate [4, 8], the transmission power should be allocated efficiently for QoS provisioning.

1.1.3 Bandwidth (or Sub-Channel) Allocation

Frequency spectrum or bandwidth is one of the fundamental resources in wireless communications. The bandwidth of a transmission signal is the difference between the highest and lowest frequency component of that signal. The amount of allocated bandwidth to a user will put a fundamental limit on the potential data rate it can realize. Moreover, the amount of available bandwidth in each BS is limited, which makes it hard to satisfy all associated users' demands without considering optimized bandwidth allocation. Another important aspect of bandwidth is its relation to interference. If two transmitted signals occupy a disjoint range of frequencies, they will not interfere with each other and will be perfectly distinguishable at the receiver side. On the contrary, if they share a range of frequencies they will interfere if the same time slot and code are used [4]. Thus, the problem of bandwidth management plays an important role in the users' communication experience.

Fig. 1.1 (bottom-right) shows an example of bandwidth allocation to users of BS_7, where disjoint blocks of frequencies are distributed among the users. It should be noted that since in the wireless communication environment, the same spectrum is reused among multiple BSs, thus, in each transmission time slot, the transmitter has to optimize bandwidth allocation to minimize interference [4].

1.1.4 Beamforming

The usage of multiple antennas on both the transmitter and receiver sides has become attractive, as it increases the wireless medium's capacity without additional transmit power or bandwidth consumption [4]. The use of massive antennas is also identified as a central physical layer technology for 5G and beyond. Based on the situation of

the communication channel, the radiation beams from the antenna have to point to certain directions, so that at the receiver side, the signal quality remains detectable. The process of assigning the antennas' directions is called beamforming. Beamforming is done by determining beamforming vectors which are multiplied in transmitted or received signals. These vectors will change the direction of the antenna lobes without rotating the antenna mechanically [4]. Fig. 1.1 (top-right) shows an example of beamforming from a base station with multiple antennas to single antenna users, where the intensity of the color of each beam is relative to the amount of allocated power to that beam.

1.2 Radio Resource Management (RRM) Challenges

As mentioned before, a formal way of looking at RRM problems is through mathematical optimization where every RRM problem can be formulated to find the optimal allocation of the resources, while maximizing a predefined objective given the complete or partial CSI [4]. Thus, an obvious way of solving such problems is to develop algorithms using mathematical optimization principles. Although this approach has worked well so far, it's high computational and time complexity is a bottleneck for the future generation of wireless systems [3]. The specific reasons are listed in the following.

1.2.1 Increased Dimensionality

Due to the dense deployment of terrestrial and aerial BSs, massive devices, and massive antenna deployments, the number of variables over which the optimization has

to occur in future networks will increase drastically [3]. This will increase the computational complexity of the existing optimization-based algorithms that are already computationally demanding. Since most of the RRM problems are non-convex and NP-hard in their nature [8], it is typically hard to find globally optimal solutions while escaping the local optimums in a reasonable amount of time.

1.2.2 Need of Extremely Fast RRM Solutions

The coherence time of wireless channels is going to become shorter in 6G. This is mainly because of the use of extremely high-frequency spectrum, (i.e. millimeter wave, THz, and visible light), where the frequencies are vulnerable to static and dynamic blockages, scattering, molecular absorption, diffraction, etc. [9]. Moreover, for some applications such as vehicular networks [5], due to the high mobility of the users, the communication channel is going to change faster; thus resulting in a shorter coherence time. Also, some particular applications like ultra-reliable low-latency communication (URLLC) necessitate extremely fast decision-making. The shorter coherence time makes the duration of the time-slots shorter, which requires faster resource allocation.

1.2.3 Lack of Robustness

Another systematic deficiency of the conventional RRM algorithms is their inability to utilize history to reduce computational complexity. In other words, if a given RRM problem was solved in the past for a given CSI and the current CSI is very similar to the previous one, there is no means to use the previous solution again. An alternative approach is to extract patterns from the historical data and use them in computing

the current solution. Moreover, most of these algorithms work very well under the assumption of perfect CSI, which is not always the case. Therefore, a new method that is more robust to CSI imperfection needs to be designed [9].

1.2.4 Joint Optimization of Multiple Resources

Another difficulty of wireless RRM problems is that certain resource allocation decisions are not independent and affect each other's optimality. For instance, power control and user-BS assignment are perfect examples of this dependency. Solving the two problems separately will only lead to sub-optimal solutions [8]. A popular approach in dealing with this problem is to use alternating optimization, where an iterative algorithm is designed in which all variables are optimized separately in each iteration [8, 10]. During each step, a single variable is optimized given the values of the other variables. After a number of iterations, the algorithm generally converges and outputs a stationary point, which in most cases is not guaranteed to be globally optimal [8,10]. Note that solving the problem via optimization in a truly joint manner is typically exhaustive, thus not computationally feasible [8, 10].

1.3 Deep Learning for RRM in Wireless Networks

Traditionally, mathematical optimization is used to formulate and solve problems in machine learning. Recently, 'Learning to Optimize (L2O)' approach seeks to use machine learning, in particular deep learning (DL), to solve optimization problems.

Recently, DL is emerging as a promising solution to design RRM solutions [11,12]. The convention in the wireless community is to formulate an RRM problem using variable optimization, where for every data instance an optimization variable needs to be

determined by solving an optimization problem. [13] proposed an equivalent formulation where instead of finding the variables, the function that accurately describes the mapping from the problem data to decision variables is determined. Thus, instead of searching on the variable domain, one can search for every possible mapping that exhibits the properties of the underlying implicit map. This approach is called *functional optimization*.

Subsequently, artificial neural networks (ANN) based model is typically applied to approximate the mapping between the environment and optimal resource allocations via the training process. Once trained, the time complexity to get the results from the ANN is much lower than the traditional optimization-based approaches. This is because the main computation of the current neural networks involves matrix multiplication and point-wise non-linearity like `tanh` and Rectified Linear Units (`ReLU`) [5], which enjoy efficient hardware implementations; thus resulting in lower online computational complexity. In summary, DL offloads the computational complexity from online to offline by training an ANN model offline and using it online to optimize resources [5].

1.4 Scope of the Thesis

While DL can minimize the time complexity of the RRM algorithms, the structure of the neural network hinders incorporating complex constraints. It is noteworthy that most of the RRM problems come with various types of constraints, e.g., power budget, QoS, etc, and efficient systematic incorporation of them in the architecture of neural networks is a fundamental challenge. In this context, the current state-of-the-art classifies constraints into two groups and applies different solution approaches

for them, i.e.,

- The first group is the one that can be expressed analytically, e.g. ReLU and Sigmoid functions impose non-negativity and membership in the interval ranging between 0 and 1, respectively. Such constraints can be incorporated at the final layer of the ANN in a straightforward manner. An example of this is the power budget constraint [12]. Despite the guaranteed feasibility of the output, this approach cannot be applied to complex constraints.

- The second group of constraints, which do not provide a closed-form functionality description, e.g. minimum-rate (or QoS) constraint, are incorporated in the loss function in one of the following two ways. Either they are treated as a penalty term [12] or customize the loss function as the Lagrangian of the original problem, where the learnable dual variables penalize the violation of constraints [7,13]. Although this approach produces reasonable results in terms of loss minimization, it has its own downsides. First, resorting to the dual domain introduces the duality gap, meaning that even a globally optimal solution to the dual problem might not be a global solution for the primal problem. This introduces a systematic sub-optimality, especially when the RRM problem is non-convex which is typical in practice [8]. Second, it does not provide a guaranteed way of incorporating constraints and making sure that the results are always feasible.

Also, in practice, RRM problems are a function of multiple inter-related resource variables, i.e., there is more than one variable to optimize in RRM problems, e.g. power control and user scheduling, beamforming design, spectrum assignment, etc.,

which are best to be optimized jointly as they are not purely independent. This is different from the traditional DL applications.

Due to the infancy of this line of research, most of the existing research works consider simplified constraints belonging to the former group and single variable optimization problems. Thus, several questions need to be addressed before DL-based methods can completely replace traditional optimization-based RRM algorithms, such as (i) How to systematically incorporate convex/non-convex RRM constraints into neural network architecture?, (ii) How to ensure a zero constraint violation probability?, (iii) What architectures are suitable for joint-optimization problems? In this thesis, I aim to address the aforementioned questions.

1.5 Contributions

To this end, the main contributions of this thesis are as follows:

- I propose two novel DL-based solvers for the classical power control and user association problems in a multi-user interference channel with QoS constraints. The first is called Deep Implicit Projection Network (DIPNet), and the second is called Deep Explicit Projection Network (DEPNet). The former utilizes convex optimization to project the neural network's output to the feasible set defined by the QoS constraints and the latter uses an iterative process to achieve the same.

- The proposed models were trained in an unsupervised manner and compared with FCNNs like PCNet [12] as DNN-based benchmarks, and Geometric Program (GP) [14] as the optimization-based benchmark. The network sum-rate,

constraint violation probability, and online test time are used as the performance evaluation metrics.

- Numerical results demonstrate that the proposed DIPNet and DEPNet achieve zero constraint violation probability while outperforming PCNet in terms of network sum-rate at the expense of slightly increased computational complexity.

- The user association networks are trained in an unsupervised manner and compared with both DNN-based and optimization-based benchmarks. For the former, an FCNN with a softmax layer is used as in [15], and for the latter, a mixed-integer programming solver from MOSEK package [16] is utilized. The sum-rate, constraint violation probability, and online testing time are used as the evaluation metrics.

- The user association DNNs are combined with the power control DNNs (chapter 3) to provide an alternating optimization-based solution for the joint power control and user association problem, called JUPNet. For the comparison, the mixed-integer solver from MOSEK package and Geometric Program (GP) is used to solve user association and power control sub-problems, respectively, in an alternating manner. The resulting optimization-based solver is used as the benchmark.

1.6 Research Outcome

- M. Alizadeh and H. Tabassum, "Power Control with QoS Guarantees: A Differentiable Projection-based Unsupervised Learning Framework," (submitted)

Chapter 2

Mathematical Preliminaries and Literature Review

In this section, a brief description of the relevant mathematical preliminaries is provided to lay the foundation for the subsequent sections. This is followed by a literature review of the current state of the DL-enabled RRM techniques as well as the open questions that are still unanswered.

2.1 Mathematical Preliminaries

Given the vivid success of deep neural networks (DNN) in finding high-quality near-optimal solutions for complex tasks like image recognition, machine translation, etc. [17] [18], many scholars have become interested in solving non-convex NP-hard optimization problems using DNNs. The optimization problems are formulated typically as follows:

$$\begin{aligned} \mathbf{y} = \underset{\mathbf{y}}{\text{argmax}} \quad & f(\mathbf{y}, \mathbf{x}) \\ \text{subject to} \quad & g(\mathbf{y}, \mathbf{x}) \leq 0 \\ & h(\mathbf{y}, \mathbf{x}) = 0 \end{aligned} \tag{2.1}$$

where $\mathbf{y} \in \mathbb{R}^n$ is the vector of optimization variables, $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ is the problem data, $f : \mathbb{R}^n \times \mathbb{R}^d \longrightarrow \mathbb{R}$ is the objective function, $g : \mathbb{R}^n \times \mathbb{R}^d \longrightarrow \mathbb{R}^q$, and $h : \mathbb{R}^n \times \mathbb{R}^d \longrightarrow \mathbb{R}^p$ represent the q inequality and p equality constraints of the problem, respectively. As the problem data varies frequently, (2.1) needs to be solved at every data point to acquire the optimal value of $\mathbf{y}$.

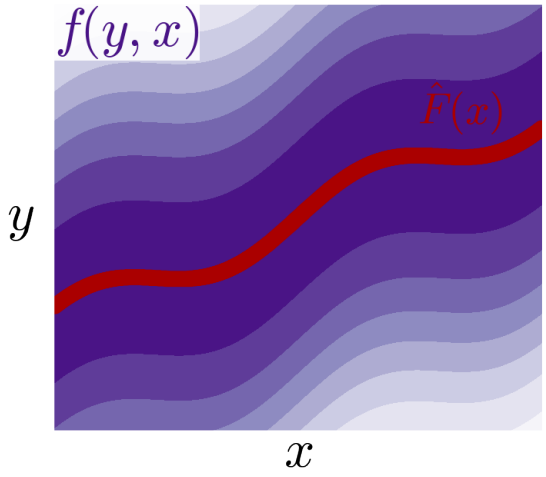

Figure 2.1: An illustrative example of the unconstrained version of problem (2.1) [1]

Although effective, this approach yields high computational complexity [2, 3]. As mentioned in [13], I refer to such optimization problems as '*variable optimization*' problems. Another way to look at problem (2.1) as a mapping from the domain of $\mathbf{x}$ to the domain of $\mathbf{y}$, where for each $\mathbf{x}$, I find the corresponding $\mathbf{y}$ by solving (2.1). Fig. 2.1 shows an illustrative example of this outlook, where $y^*(\mathbf{x})$ is the underlying function between $\mathbf{x}$ and $\mathbf{y}$, and the maximum of $f(\mathbf{y}, \mathbf{x})$ lies on $y^*(\mathbf{x})$. Inspired by this, researchers attempt to learn the implicit mapping from the problem data $\mathbf{x}$ to the solutions $\mathbf{y}$. This can be done by transforming (2.1) to its *equivalent functional optimization form*, whose goal is to find an efficient function to map problem data to the solutions given a class of functions.

The functional optimization form of (2.1) can thus be given as [7]:

$$\begin{aligned} \underset{F(\mathbf{x})}{\text{maximize}} \quad & \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[f(F(\mathbf{x}),\mathbf{x})] \\ \text{subject to} \quad & g(F(\mathbf{x}),\mathbf{x}) \leq 0, \quad \forall \mathbf{x} \in \mathcal{X} \\ & h(F(\mathbf{x}),\mathbf{x}) = 0, \quad \forall \mathbf{x} \in \mathcal{X} \end{aligned} \tag{2.2}$$

where F is a function that maps the problem data to the solution space, i.e., $\mathbf{y} = F(\mathbf{x})$. $p(\mathbf{x})$ is a probability distribution over the possible values of $\mathbf{x}$. As shown in [13], (2.1) and (2.2) are equivalent, in a way that if $\hat{F}(\mathbf{x})$ is a solution to (2.2), then for every $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} = \hat{F}(\mathbf{x})$ is a solution to (2.1).

Since DNNs are shown to be a very rich family of parametric functions, in the sense that even an FCNN with one hidden layer has universal function approximators property [19], I consider them for approximating F, i.e. $F(\mathbf{x}) = \mathcal{N}_y(\mathbf{x};\mathbf{w}_y)$ where $\mathcal{N}_y$ is a DNN, whose output is $\mathbf{y}$, and $\mathbf{w}_y$ is its parameters. Using the statistical learning theory, the problem of finding $\mathbf{w}_y$ via learning can thus be formulated as:

$$\begin{aligned} \underset{\mathbf{w}_y}{\text{minimize}} \quad & \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[l(\mathcal{N}_y(\mathbf{x};\mathbf{w}_y),\mathbf{x})] \\ \text{subject to} \quad & \mathbf{w}_y \in \mathbb{R}^d \end{aligned} \tag{2.3}$$

where d is the dimension of $\mathbf{w}_y$, and l is the loss function and measures how good the output of the neural network is for a given data point $\mathbf{x}$. *Now, the goal is to design the loss function and DNN architecture so that the solution of* (2.3) *will be a solution for* (2.2) *without violating any constraints of* (2.2).

Remark: Regardless of the DNN's architecture choice, the training can be done in a supervised or unsupervised fashion. In the supervised training, the choice of loss

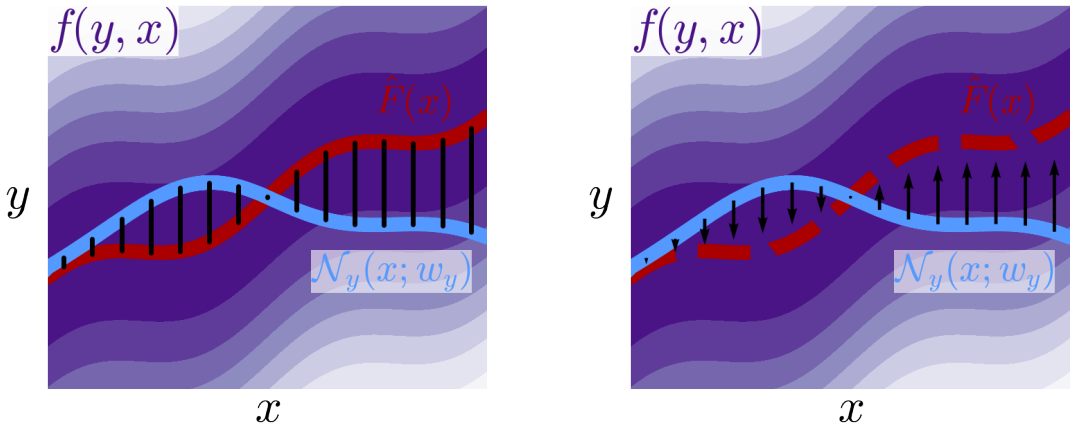

Figure 2.2: An illustration of supervised (left) and unsupervised (right) learning for solving the unconstrained version of (2.3) [1]

function becomes trivial. For instance, l_2-norm of the difference between the solution of (2.1) and the output of the DNNs is the widely-adopted loss function [2, 11]. Fig 2.2 (left figure) illustrates the dynamics of the supervised method, where the straight red line indicates the values of the underlying function ($\hat{F}(\mathbf{x})$), i.e., the labels, and the blue line shows the output of the DNN ($\mathcal{N}_y(\mathbf{x};\mathbf{w}_y)$), i.e., the predictions. The straight black lines are the difference between the labels and the predictions that are quantified by the loss function and will be minimized during the training. Despite its effectiveness, the main drawback of the supervised method is its dependence on the availability of the optimal solutions of (2.1), which in most cases are hard to obtain due to the potential non-convexity of (2.1) [2, 12].

An alternative is unsupervised training which relaxes the need for labels. Fig 2.2 (right figure) illustrates this approach, where the DNN is provided with some guidance (black arrows) about the underlying function ($\hat{F}(\mathbf{x})$). This guidance signal is given to the DNN via the loss function and informs the neural network about the quality of the predictions. The widely-used candidate for the unsupervised loss function is

as follows:

$$l(\mathbf{y}, \mathbf{x}) = -f(\mathbf{y}, \mathbf{x}) \tag{2.4}$$

where $\mathbf{y} = \mathcal{N}_y(\mathbf{x}; \mathbf{w}_y)$ is the output of the DNN, and $l(\mathbf{y}, \mathbf{x})$ is the loss function. By minimizing this loss during the training, the output of the DNN gets closer to the points that maximize the objective function of (2.2). In other words, in this method, the DNN learns to adjust its output so that it resembles the characteristics of the underlying function ($\hat{F}(\mathbf{x})$). This choice is also motivated by the numerical results of the previous works [2, 12, 13].

Remark: It might be better to refer to the unsupervised method as self-supervised. The reason is that there is a supervision signal that comes from the problem data itself. However, since the literature of wireless communication uses the 'Unsupervised' to refer to this method, in this work, I used the term unsupervised to avoid inconsistency with the current literature.

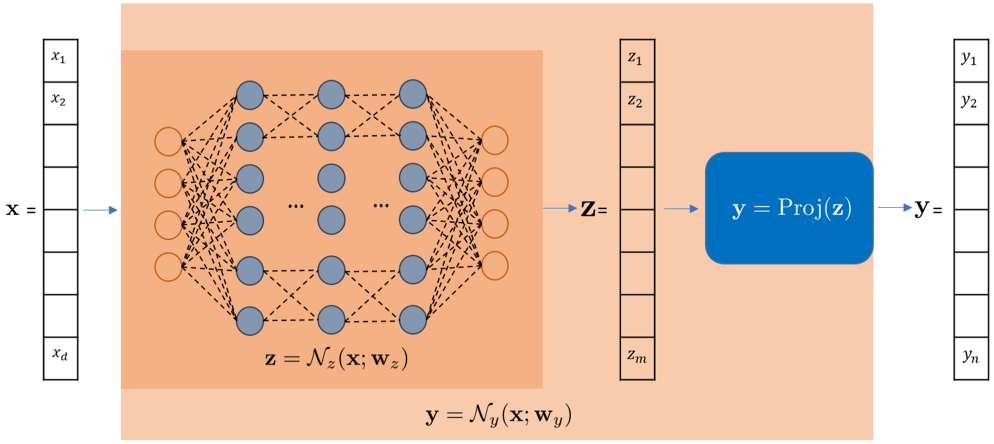

Figure 2.3: Combination of a deep neural network and a differentiable projection function.

Regardless of the aforementioned supervised or unsupervised training methods,

the feasibility of $\mathbf{y}$ cannot be guaranteed in the test time. A remedy for this is to design a transformation that takes the output of the DNN and makes it closer to the feasible set of (2.1). I refer to this transformation as *Projection*, denoted by Proj. In this way, $\mathbf{y}$ becomes the output of the projection function and the output of the DNN is denoted by $\mathbf{z}$, i.e. $\mathbf{y} = \text{Proj}(\mathbf{z}), \mathbf{z} = \mathcal{N}_z(\mathbf{x}; \mathbf{w}_z)$ (Fig. 2.3). Although there are different definitions for a projection function, I use the following definition throughout this work:

Definition 1: A function $\text{Proj} : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ is called a projection function w.r.t (2.1) if the following holds:

- The Jacobian of Proj can be evaluated, i.e., $\frac{\partial \mathbf{y}}{\partial \mathbf{z}}$. This is to make the end-to-end training of the DNN possible using gradient-based methods [17].
- $\mathbf{y}$ meets the constraints of (2.1), i.e., $h(\mathbf{y}, \mathbf{x}) = 0, g(\mathbf{y}, \mathbf{x}) \leq 0$.

The projection function can be defined implicitly or explicitly. I consider mathematical optimization as a language to describe the projection function implicitly. On the other hand, to define the explicit projection, I provide an overview of the analytic and iterative approaches.

2.1.1 Implicit Projection via Mathematical Optimization

In the following, I implicitly define a projection function w.r.t. (2.1), i.e.,

$$\begin{aligned} \mathbf{y} = \underset{\mathbf{y}}{\text{argmax}} \quad & k(\mathbf{y}, \mathbf{z}; \boldsymbol{\theta}) \\ \text{subject to} \quad & g(\mathbf{y}, \mathbf{x}) \leq 0 \\ & h(\mathbf{y}, \mathbf{x}) = 0 \end{aligned} \tag{2.5}$$

where $k : \mathbb{R}^n \times \mathbb{R}^m \longrightarrow \mathbb{R}$ is a parametric objective function, $\boldsymbol{\theta} \in \mathbb{R}^t$ represents the parameters of k, and g, h are the constraints of (2.1). Note that (2.5) describes a parametric mapping from $\mathbf{z}$ to $\mathbf{y}$, where $\boldsymbol{\theta}$ denotes the parameters of this functionality. The functionality described in (2.5) is a *Projection function*, as defined in **Definition 1**. This is because, given the implicit function theorem [20–22], the Jacobian of the output w.r.t. the input, i.e., $\frac{\partial \mathbf{y}}{\partial \mathbf{z}}$, and the parameters, i.e., $\frac{\partial \mathbf{y}}{\partial \boldsymbol{\theta}}$ can be computed, regardless of how the solution is derived. Moreover, since the constraints of (2.5) and (2.1) are the same, they have the same feasible set, meaning that the solution of (2.5) lies in the feasible set and satisfies the constraints of (2.1).

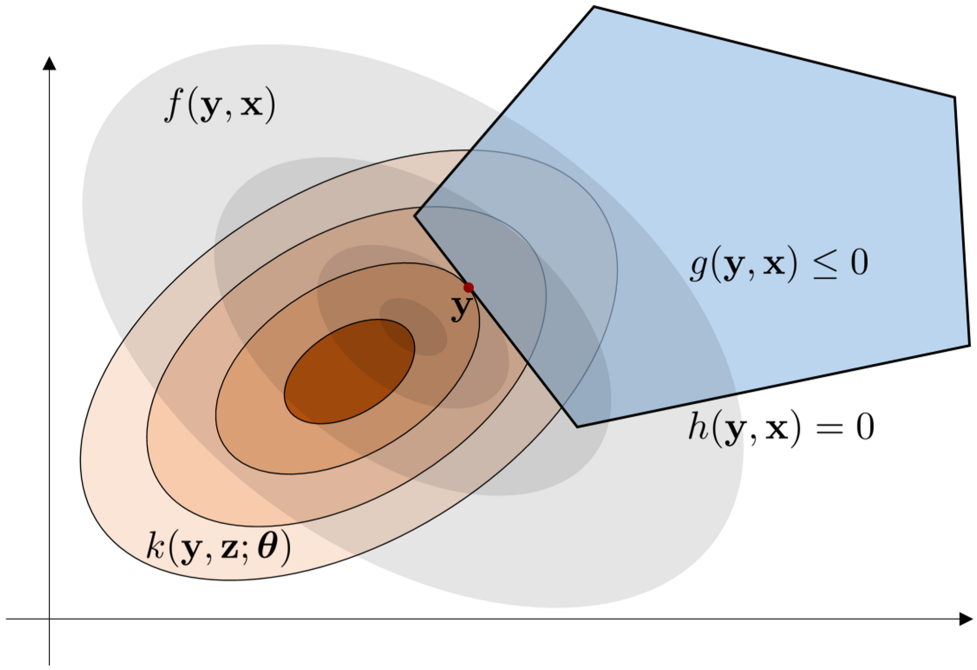

Figure 2.4: An illustration of using mathematical optimization to define a projection function implicitly

The structure of (2.5) enables us to achieve the feasible output of the DNN by designing k. To put it differently, k can be thought of as a score function, which gives

different scores to the points in the feasible set, and the point with the highest score will be chosen eventually. Fig. 2.4 portrays a graphical example of this method where the underlying gray contour and blue polytope show the behavior of the objective and constraints of the main problem (2.1) and the brown contour shows the behavior of the objective function of (2.5). Following are the two examples of such projection functions, i.e.,

$$k = -||\mathbf{y} - \mathbf{z}||_2^2, \quad k = \mathbf{y}^{\mathrm{T}}\mathbf{z} \tag{2.6}$$

This approach requires an algorithm that can solve (2.5) in a computationally efficient way. For example, if (2.5) is a non-convex problem, there is no general algorithm that can provide polynomial time complexity. To make (2.5) convex, k should be convex, like (2.6). Moreover, the constraints of (2.1) should be convex as well. In this way, the projection function (2.5) becomes an instance of the differentiable convex optimization layer introduced in [21]. The implementation details of this function and its integration with automatic-differentiation frameworks like PyTorch are available in [21, 23].

2.1.2 Explicit Projection via an Analytic Expression

There is no doubt about the computational benefits of expressing a projection function with analytical and closed-form expressions. Once the closed-form expression of functionality is available, it can be implemented and evaluated easily. Fig 2.5 shows an example of having access to a closed-form projection function that can take the output of the DNN ($\mathbf{z}$) and map it to the constraint set (blue polytope). Although tempting, in many cases, this computational benefit comes with a cost, which is the

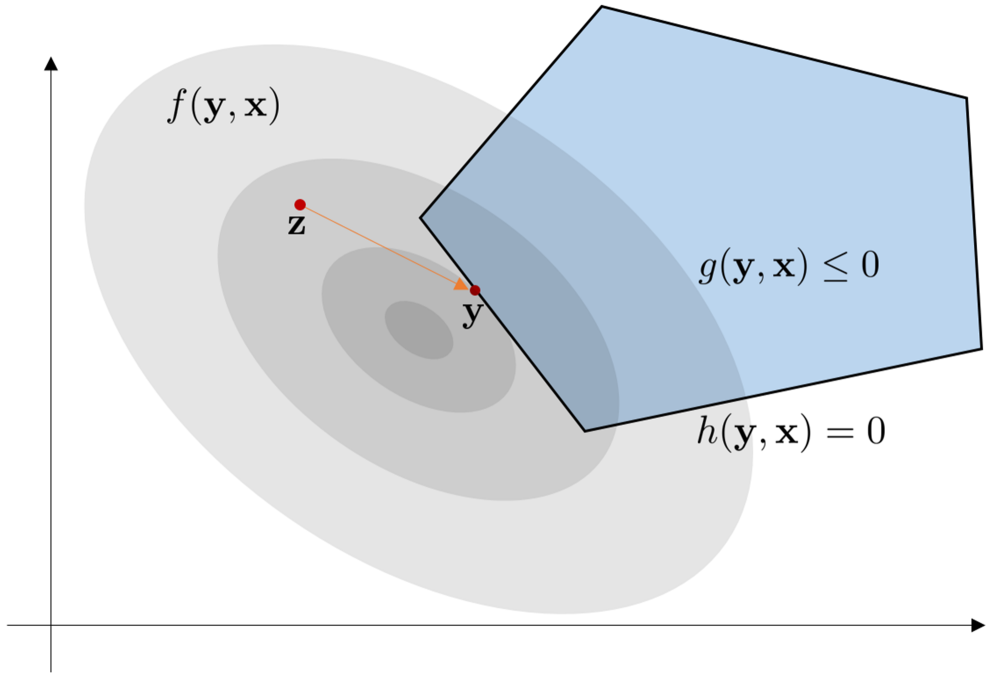

Figure 2.5: An illustration of an explicitly defined projection function

restriction over the class of projection functions. Without an intention to be exhaustive, in this section, I review some projection functions that their closed-form expression is available. In the following, in each line, I first write the constraints, and then the corresponding projection function.

- Projection onto the non-negative orthant ($\mathbf{y} \geq \mathbf{0}$): In this case, one can use ReLU, i.e., $\mathbf{y} = \text{Relu}(\mathbf{z}) = \max(\mathbf{z}, 0)$, as the projection function. As shown in [24], ReLU is the answer to the euclidean projection problem, i.e., $\mathbf{y} = \text{argmin} \frac{1}{2}||\mathbf{y} - \mathbf{z}||_2^2 \quad s.t. \quad \mathbf{y} \geq \mathbf{0}$. The other choice is the exponential function, i.e., $\mathbf{y} = e^{\mathbf{z}}$. This choice always results in an interior-point of the non-negative orthant, meaning that $\mathbf{y}$ is going to be positive and only becomes zero when $\mathbf{x}$ approaches negative infinity.

- Projection onto the unit hypercube ($\mathbf{y} \geq \mathbf{0}$, $\mathbf{y} \leq \mathbf{1}$): In this case, one can use $textbf{y} = max(min(\mathbf{x}, 1), 0)$, which is the realization of the euclidean projection. The other choice, which outputs an interior-point, is Sigmoid, i.e., $\mathbf{y} = \text{Sigmoid}(\mathbf{z})$.

- Projection onto the $(n-1)$ simplex ($\mathbf{1}^{\mathrm{T}}\mathbf{y} = 1$): In this case, Sparsemax [25] realizes the euclidean projection, and Softmax, i.e., $\mathbf{y} = \text{softmax}(\mathbf{z})$, outputs an interior-point of the probability simplex.

One may find other closed-form projections that meet different types of constraints. The benefit of the closed-form projection is that the output of the projection function always lies in the feasible set, and enables us to find the optimal solution of (2.1). The downside is that it does not provide a general way of tackling a broad range of constraints.

2.1.3 Explicit projection via an Iterative Process

As explained in section 2.1.1, a parametric optimization problem, e.g. (2.5), can be used to formalize a projection function. Despite its wide applicability and guaranteed feasibility in case of convex constraints, its dependence on the efficient optimization solver, and challenges in fully leveraging the parallel computational power of GPUs, motivate the search for other alternatives.

Thus, I describe another way of incorporating constraints at the output of neural networks. This method uses a differentiable iterative process to realize a projection function. Fig. 2.6 demonstrates an example of this approach. One example of this method is Sinkhorn normalization [26], which takes a positive-valued matrix and after some iterations outputs a doubly stochastic matrix, i.e. a positive-valued

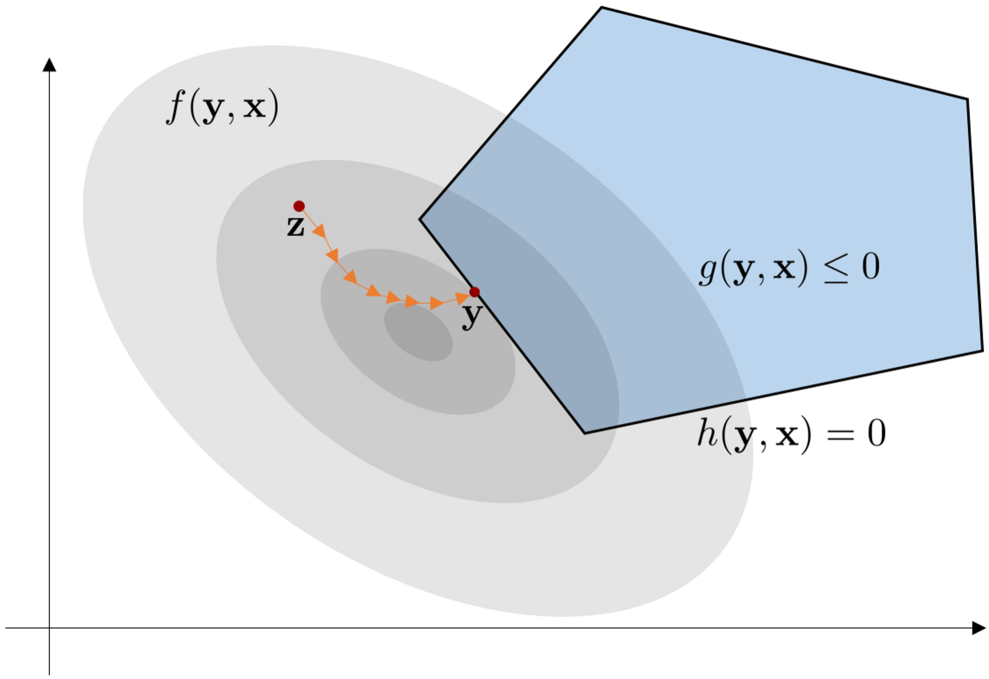

Figure 2.6: An illustration of an iterative process as the projection function

matrix whose rows and columns sum to one. A detailed explanation of this process is provided in chapter 4.

Inspired by the idea of an iterative process, recently, [2] proposed a new and general method, called Deep Constraint Completion and Correction (DC3), which is compatible with GPU-based training and doesn't require the convexity of the constraint set. In this method, the projection function is broken down into two processes called *completion* and *correction*. The pipeline of this method takes the following steps. First, the DNN outputs a partial subset $\mathbf{z}$ of the solution variables of (2.1). The completion process then completes the remaining variables such that the resulting solution $\mathbf{y}'$ meets the equality constraints of (2.1). The correction process, afterward, corrects the solution by making it closer to the feasible set of (2.1). The resulting solution $\mathbf{y}$ doesn't violate both equality and inequality constraints of (2.1).

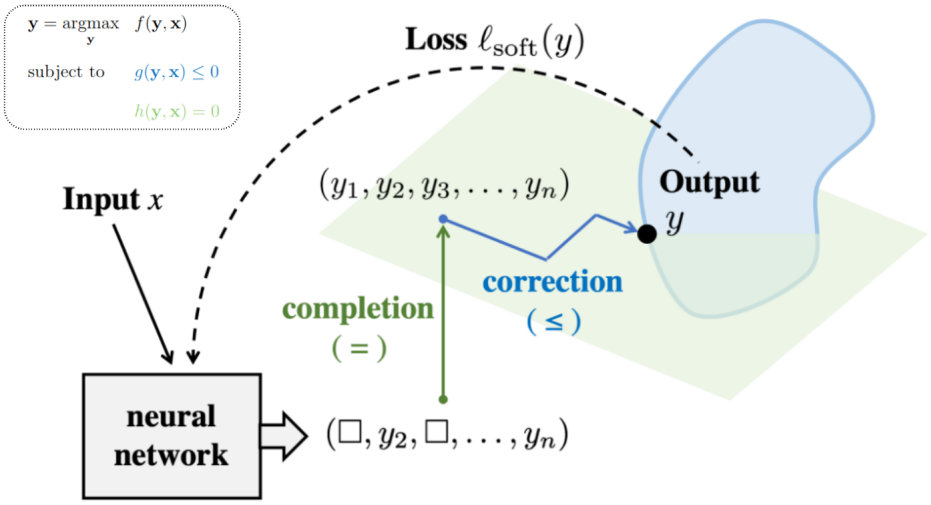

Figure 2.7: An illustration of Completion and Correction process [2]

Following [2], first and second-order based methods are used to realize the correction process. It is not guaranteed that these methods always converge to a global or local optimum. However, once initialized with a point close to the optimum solution, these methods are expected to converge in a finite number of iterations. Thus, once the training is done, the output of the backbone DNN should be close to the feasible set, and applying the correction process afterward will result in a point that meets the constraints. The effectiveness of this approach is supported by the numerical results of the original paper [2], and the numerical results of this work, presented in chapter 3. Aside from gradient-based methods, investigated in [2], I used the newton method at test time to accelerate the convergence. To address the instability of the hessian matrix, I used regularization by adding the identity matrix. This is a widely-used technique [27,28] that makes sure about the existence of the inverse of hessian, which is required for calculating the updates for the correction process. Fig. 2.7 illustrates the functionality of both completion and correction processes. In the following, the

mathematical descriptions of the completion and correction processes are provided.

- **Completion Process:** The goal of this process is to take the output of the neural network ($\mathbf{z}$) and output a variable $\hat{\mathbf{y}}$ that meets the equality constraints. I adopt the variable-elimination approach proposed in [2]. In this way, the optimization variable $\mathbf{y}$ is divided into two variables. The first one, the output of the DNN, is an independent variable that has a lower dimension. Given the independent variable, the other is calculated such that the resulting optimization variable will meet the equality constraints. As pointed out in [2], it is more efficient to have an independent variable, that has a lower dimension than the original variable, than to output something with the same or higher dimension and adjust its entries to satisfy the equality constraints.

  Let $z \in \mathbb{R}^m$ denotes the output of the DNN, i.e., $\mathbf{z} = \mathcal{N}_z(\mathbf{x}; \mathbf{w}_z)$, and let $m \leq n$, where m is the dimension of $\mathbf{z}$ and n is the dimension of $\mathbf{y}$, the solution of (2.1). The completion process $\phi(\mathbf{z}) : \mathbb{R}^m \longrightarrow \mathbb{R}^{n-m}$ takes the output of the DNN, denoted by $\mathbf{z}$, and using that output completes the $n - m$ missing variables so that the resulting n-dimensional vector satisfies the equality constraints, i.e.,

$$\mathbf{y}^{'} = \begin{bmatrix} \mathbf{z} \\ \phi(\mathbf{z}) \end{bmatrix} \longrightarrow h(\mathbf{y}^{'}, \mathbf{x}) = 0 \tag{2.7}$$

  Doing this results in a point ($\hat{\mathbf{y}}$) that always lives on the manifold of points that satisfy the equality constraints. An explanation of the choice of m (dimension of the $\mathbf{z}$) and the correction process $\phi(.)$ is provided later. Now, the output of the DNN can change independently in $\mathbb{R}^m$, and the resulting $\hat{\mathbf{y}}$ will be adjusted accordingly. In the case of linear equality constraints, i.e., $h(\mathbf{y}, \mathbf{x}) = \mathbf{A}(\mathbf{x})\mathbf{y} -$

$\mathbf{b}(\mathbf{x})$, other choices than (2.7) are available as well [29]. For example, if $\mathbf{A}(\mathbf{x})$ has a full row rank and the number of rows is smaller than the number of its columns (under-determined equality constraint), one can find a projection to the null space of $\mathbf{A}(\mathbf{x})$ like $\mathbf{F}(\mathbf{x})$, i.e., $\mathbf{A}(\mathbf{x})\mathbf{F}(\mathbf{x}) = 0$. By applying that projection to the output of the DNN, the resulting solution will satisfy the equality constraints [30]. Since the approach proposed in [2] (dc3) provides a general framework for dealing with constraints and is used in this work, the rest of this section is focused on a detailed explanation of dc3 methodology.

The $\mathbf{y}^{'}$, acquired form (2.7), can be passed to the correction process, or be fed directly to the loss function, in case there are no inequality constraints in (2.1). Either way, for realizing end-to-end training of the DNN, i.e. $\mathcal{N}_z$, the Jacobian of $\mathbf{y}^{'}$ w.r.t. $\mathbf{z}$ should be calculable. Using (2.7), I can write:

$$J^{\mathbf{y}^{'}}(\mathbf{z}) = \frac{\partial \mathbf{y}^{'}}{\partial \mathbf{z}} = \begin{bmatrix} \mathbf{I} \\ \frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}} \end{bmatrix} \tag{2.8}$$

where $J^{\mathbf{y}^{'}}(\mathbf{z}) : \mathbb{R}^{n \times m}$ denotes the Jacobian of $\mathbf{y}^{'}$ w.r.t. $\mathbf{z}$, and $\mathbf{I} \in \mathbb{R}^{m \times m}$. As denoted in [2], the Jacobian of ϕ w.r.t. $\mathbf{z}$ can be calculated as:

$$J^{\phi}(\mathbf{z}) = \frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}} = -J^{h}_{:,m:n}(\mathbf{y}^{'})^{-1} J^{h}_{:,0:m}(\mathbf{y}^{'}) \tag{2.9}$$

$J^{h}(\mathbf{y}^{'}) : \mathbb{R}^{p} \longrightarrow \mathbb{R}^{n}$ denotes the Jacobian of h w.r.t. $\mathbf{y}^{'}$. If $\mathbf{y}^{'}$ is then fed to a loss function like $l : \mathbb{R}^{n} \longrightarrow \mathbb{R}$ the derivative of loss w.r.t. $\mathbf{z}$ can be calculated as follows:

$$\frac{dl}{d\mathbf{z}} = \frac{\partial l}{\partial \mathbf{z}} - \frac{\partial l}{\partial \phi(\mathbf{z})} J^{h}_{:,m:n}(\mathbf{y}^{'})^{-1} J^{h}_{:,0:m}(\mathbf{y}^{'}) \tag{2.10}$$

This derivative is necessary for using backpropagation for calculating the derivative of l w.r.t. DNN's parameters, i.e. $\frac{\partial l}{\partial \mathbf{w}_z}$.

- **Correction Process:** Now that I have a point $\mathbf{y}^{'}$ that satisfies equality constraints ((2.7)), I aim to satisfy inequality constraints as well. This is done via the correction process. The correction process basically takes $\mathbf{y}^{'}$ and outputs $\mathbf{y}^{"}$ after one iteration. $\mathbf{y}^{"}$ is closer to the feasible set of inequality constraints and lies in the manifold of points in $\mathbb{R}^n$ than satisfies equality constraints, i.e. $h(\mathbf{y}^{"}, \mathbf{x}) = 0$. By applying the correction process multiple times, a point $\mathbf{y}$ that meets both equality and inequality constraints is achieved. The correction process can be described mathematically as follows [2]:

$$\rho : \mathbb{R}^n \longrightarrow \mathbb{R}^n, \quad \rho(\mathbf{y}^{'}) = \begin{bmatrix} \mathbf{z} + \Delta\mathbf{z} \\ \phi(\mathbf{z}) + \frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}} \Delta\mathbf{z} \end{bmatrix} \tag{2.11}$$

In other words, I slightly move $\mathbf{z}$ by $\Delta\mathbf{z}$ and at the same time update the rest of the variables in a way that the resulting point also meets the equality constraints. Later, different candidates for realizing $\Delta\mathbf{z}$ are discussed. The update is basically a move towards the direction of the first-order approximation of ϕ, i.e.,

$$\hat{\mathbf{z}} = \mathbf{z} + \Delta\mathbf{z} \longrightarrow \phi(\hat{\mathbf{z}}) \approx \phi(\mathbf{z}) + \frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}} \Delta\mathbf{z} \tag{2.12}$$

where $\frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}}$ is derived from (2.9). It should be noted that $\Delta\mathbf{z}$ should be small enough to make this approximation work. As shown in [2], choosing gradient-based updates for $\Delta\mathbf{z}$ is a good match with this approximation and works even in the case of non-linear equality constraints. One can use higher-order

approximations if this does not result in feasible points for equality constraints.

Choice of m: The dimension of $\mathbf{z}$, given by m, can be determined based on the chosen solver for the completion process (ϕ). In [2], they assumed that the main problem (2.1) is not over-determined, meaning that the number of variables (n) is greater than or equal to the number of equality constraints, i.e., $n \geq p$. Thus, they chose $m = n - p$ for linear equality constraints. This is a wise choice from a practical perspective, which will be explained later.

Choice of $\Delta\mathbf{z}$: Let us denote $||\max(g(\mathbf{y}, \mathbf{x}), 0)||_2^2$ as $\mathrm{V}_x(\mathbf{y}) : \mathbb{R}^n \longrightarrow \mathbb{R}$, which measures violation of inequality constraints for a datapoint $\mathbf{x}$. Since $\mathbf{y}^{'}$ is used as the input to $\mathrm{V_x}$, $\mathrm{V_x}$ becomes a function of $\mathbf{z}$. let $\nabla^{V_x}(\mathbf{z}) \in \mathbb{R}^m$ and $\mathbf{H}^{V_x}(\mathbf{z}) \in \mathbb{R}^{m \times m}$ denote the gradient and Hessian of the violation of inequality constraints w.r.t. $\mathbf{z}$ and calculated, respectively, as follows:

$$\nabla^{V_x}(\mathbf{z}) = \nabla_{\mathbf{z}} ||\max(g(\mathbf{y}^{'}, \mathbf{x}), 0)||_2^2 = 2J^g(\mathbf{z}, \mathbf{x})^{\mathrm{T}} \max(g(\mathbf{y}^{'}, \mathbf{x}), 0), \tag{2.13}$$

$$\mathbf{H}^{V_x}(\mathbf{z}) = 2\mathcal{I}(\max(g(\mathbf{y}^{'}, \mathbf{x}),), 0)^{\mathrm{T}} \mathbf{K}^g(\mathbf{z}, \mathbf{x}) + 2\max(g(\mathbf{y}^{'}, \mathbf{x}), 0)^{\mathrm{T}} \mathbf{T}^g(\mathbf{z}, \mathbf{x}) \tag{2.14}$$

where

$$\mathbf{y}^{'} = \begin{bmatrix} \mathbf{z} \\ \\ \phi(\mathbf{z}) \end{bmatrix}$$

$$J^g(\mathbf{z}, \mathbf{x}) = J^g_{:,:m}(\mathbf{y}^{'}, \mathbf{x}) + J^g_{:,m:n}(\mathbf{y}^{'}, \mathbf{x}) J^{\phi}(\mathbf{z}, \mathbf{x}) \tag{2.15}$$

where $\mathcal{I} : \mathbb{R} \longrightarrow \mathbb{R}$ is an indicator function that is applied element-wise to $\max(g(\mathbf{y}^{'}, \mathbf{x}), 0)$,

i.e.,

$$\mathcal{I}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}, \tag{2.16}$$

$\mathbf{K} \in \mathbb{R}^{q \times m \times m}$ and $\mathbf{T} \in \mathbb{R}^{q \times m \times m}$ are defined as follows:

$$K^g(\mathbf{z}, \mathbf{x})_{k,i,j} = \frac{\partial g_k}{\partial z_i} \frac{\partial g_k}{\partial z_j}, \quad T^g(\mathbf{z}, \mathbf{x})_{k,i,j} = \frac{\partial^2 g_k}{\partial z_i \partial z_j} \tag{2.17}$$

$\mathbf{K}$ and $\mathbf{T}$ are rank 3 tensors and both terms in (2.14) contain vector to tensor dot product which result in a $m \times m$ matrix. The product is calculated as:

$$\mathbf{C} = \mathbf{a}^{\mathrm{T}} \mathbf{B} \longrightarrow C_{i,j} = \sum_{k=1}^{q} a_k B_{k,i,j} \tag{2.18}$$

where $\mathbf{B} \in \mathbb{R}^{q \times m \times m}$, $\mathbf{a} \in \mathbb{R}^{q \times 1}$, and $\mathbf{C} \in \mathbb{R}^{m \times m}$. Now that I have the first and second order information of V_x w.r.t. $\mathbf{z}$, $\Delta\mathbf{z}$ can vary for different descent methods [31], such as for vanilla gradient descent ($\Delta\mathbf{z} = -\mu \nabla^{V_x}(\mathbf{z})$), where μ is the step size, or Newton method ($\Delta\mathbf{z} = -\mathbf{H}^{V_x}(\mathbf{z})^{-1} \nabla^{V_x}(\mathbf{z})$) [31].

It should be noted that due to the usage of $\max(-, 0)$, a function with zero second derivative, in the definition of $V_x(.)$, the hessian includes lots of zeros, which makes it non-invertible. To solve this issue, I used the regularized version of the newton method, called Levenberg method [28]. In this way, the identity matrix is added to the hessian, i.e., $\Delta\mathbf{z} = -(\mathbf{H}^{V_x}(\mathbf{z}) + \alpha\mathbf{I})^{-1} \nabla^{V_x}(\mathbf{z})$, where $\mathbf{I} \in \mathbb{R}^{m \times m}$ is the identity matrix and α is a hyperparameter.

2.2 Literature Review

The initial use cases of DL in wireless communications include channel estimation, information decoding, localization, end-to-end learning-based transmitter, receiver design, etc. [32]. However, in this section, a review of the existing DL solutions to RRM problems, i.e., power control and user scheduling/association is provided.

2.2.1 DL-enabled Power Control

To date, most of the research works considered sum-rate or spectral efficiency maximization via power control, while considering simple BS power budget constraints. For instance, [11, 33] used fully connected neural networks (FCNNs) and recurrent neural networks (RNNs), respectively, with a supervised approach. The labels were obtained from the famous weighted-minimum-mean-square-error (WMMSE) algorithm [34]. The authors in [35] proposed the use of self-supervised learning with an FCNN to improve the network sum-rate and sample efficiency compared to supervised learning. On the other hand, [12, 36] applied unsupervised training of FCNNs. The results showed improvements over WMMSE while relaxing the need for high-quality labels. Leveraging the unsupervised approach, [6] and [37] studied the applications of convolutional neural networks (CNNs) and graph neural networks (GNNs) for power control, respectively. While the above-mentioned works considered the complete CSI as the input of the DNN, which is composed of path-loss, shadowing, and fading components, [36] demonstrated improved robustness of the solution by considering only the path-loss component.

The aforementioned works handled the power budget constraint by either using a sigmoid (in case of a single channel) or softmax (in case of multiple channels) at

the output layer of the DNN. However, none of them considered incorporating more complex constraints such as QoS constraints.

Recently, a handful of research works considered the problem of network sum-rate maximization via power control with power budget constraint and minimum rate or QoS constraint, indicating the required QoS of the users. The main theme of these works is to handle the QoS constraint either by adding a penalty term to the loss function [12], indicating the violation of these constraints, or by transforming the problem into its Lagrangian and performing the learning in the dual domain [38, 39]. To be more precise, [38] tried to develop a generic unsupervised learning framework using random edge graph neural networks to address a wide range of RRM problems with constraints. In their framework, the main problem was first transformed to its functional form which then transformed into its Lagrangian. A gradient-descent approach is used to iteratively update the primal variables, i.e. DNN parameters, and the dual variables. A subsequent work [39] used the same neural network model on the same problem, but introduced a slack variable that relaxes the minimum rate constraints. The objective changed to maximizing the network sum-rate while minimizing the slack variable. This new method, called counterfactual optimization, was an attempt to provide a fair power allocation that sacrifices the network sum-rate to provide QoS of the cell-edge users. Although the results of dual domain learning are satisfactory in terms of performance, the satisfaction of constraints cannot be guaranteed at inference. Moreover, resorting to Lagrangian comes with the duality gap which is not zero in non-convex problems [8].

Very recently, [40] studied the sum-rate maximization via power control with minimum rate and power budget constraints for an ad-hoc setup. To address the

constraints, they tried to find a differentiable closed-form projection that takes the output of the DNN and projects it into the feasible set. Although effective, this approach cannot be generalized and be applied easily to other constraints. Thus, a new generic framework that can systematically guarantee constraint satisfaction is needed.

2.2.2 Deep Learning and User Association

User or cell association is one of the core problems in wireless resource allocation which enables assigning BSs to users [32] in order to maximize a system-level performance metric, e.g., network sum-rate. The typical constraints, considered for this problem are BS quota and minimum rate (or QoS) constraints. BS quota determines the number of users allowed to be associated with a BS, and QoS constraint determines the minimum required rate that each user should experience after association.

One main difference between user association and power control is the discrete nature of decision variables. Thus, the problem is combinatorial in nature [41], however, under some conditions the problem can be relaxed to its continuous counterpart, which is a linear program, without the loss of optimality [42]. This transition is possible when the constraints have uni-modular properties. This is the case when there are only user-quota and BS-quota constraints [42]. However, this might not be the case when dealing with user-imposed or soft constraints, e.g. minimum-rate constraint. [41].

Similar to power control, the mapping between the problem data, i.e., potential users' achievable rate or the CSI, and user association decision can be learned by a DNN [15, 43]. In this regard, using a fully-connected neural network and mean

square error (MSE) loss, [44] applied supervised learning to solve the user association problem in a MIMO setup to maximize the network sum-rate. The authors did not consider the BS quota and QoS constraints.

Convolutional neural networks (CNNs) are used in [43] to solve the same problem in a heterogeneous ultra-dense network while considering BS quota, fairness, and load balancing constraints. Supervised training is used with a customized loss function consisting of MSE and penalty terms for the violation of the constraints. At the inference, however, their method failed to directly meet the BS quota constraints; thus an iterative re-association took place to handle them. Following the same paradigm, [45] tried to look at user association as a special case of a linear sum assignment problem and used supervised learning to tackle this problem. They further divided the original problem into sub-problems, solving the association for each user separately by considering it as a classification problem. Although they made sure that each user is only associated with one BS, the BS-quota constraints were left to be tackled via a heuristic. Furthermore, [15] solved the same problem via unsupervised learning. By tensor splitting at the final layer of the DNN and applying a softmax afterward, they made sure that each user will be associated with only one BS. The BS-quota constraints violation was added as a penalty term to the loss function. Although this penalizing method will reduce the constraint violation probability in general, it does not guarantee zero violation at the inference.

2.2.3 Joint optimization

As implied from the previous sections, power control and user association are two dependent problems, where the solution of one can directly impact the solution of the

other. This motivates the search for solvers that tackle these two problems simultaneously in a joint manner.

Another problem with future RRM problems is the existence of lots of co-dependent variables, e.g. power allocation and user scheduling, which require jointly optimal solutions [3]. Most of the well-known traditional works resort to solving a version of coordinate descent that guarantees the convergence to a stationary point, which might not have a good quality [10, 46]. To efficiently apply the L2O paradigm, a guiding framework has to be in place to capture the essence of joint optimization. There are very few L2O works that considered joint optimization of the resources [47, 48].

Chapter 3

Optimized Power Control with QoS Guarantees: A Deep Unsupervised Approach

Due to the practical relevance and non-convex nature of the network sum-rate maximization in wireless networks, in this chapter, I designed a DL-enabled power control solution. To resemble a realistic scenario, minimum rate constraints, as an indication of users' required QoS, are considered alongside the BSs' power budget constraint. To effectively tackle this problem with guaranteed constraint satisfaction, I need to make sure that the output of the DNN always lies in the feasible set of this problem. The common practice in this situation is to project the output of the DNN onto the feasible set. When the feasible set has a simple geometry, closed-form projection functions can be utilized. Examples of which include, ReLU, sigmoid, and softmax, which project the input onto the non-negative orthant, the interior of the unit hypercube, and the interior of the (n-1))-simplex, respectively [24]. Although the power budget constraint can be handled via these projection functions, the QoS constraint makes the feasible set of the problem have more complex geometry, requiring more complex projections.

Recently, in the deep learning community, two approaches are introduced as means of incorporating hard constraints at the DNN's output. The first utilizes the language of mathematical optimization to implicitly define the functionality of a layer [22, 49]. Using the implicit function theorem, the layer alongside the rest of the network can enjoy efficient end-to-end learning via gradient-based methods. In the same context, [21] proposed the use of an implicit layer to realize convex optimization problems and even developed a Python library for the so-called CVXPYLayer.

The second approach, called deep Constraint Completion and Correction (dc3), has an iterative nature and treats equality and inequality constraints in a different manner [2]. The DNN outputs a partial subset of the optimization variables. The remaining variables are then completed to the full-dimensional variables by the completion process to satisfy the equality constraints. The resulting solution is then corrected to satisfy the inequality constraints in an iterative manner. The direction of the updates is designed to avoid the violation of the equality constraints while getting closer to the feasible set. Another interpretation of this method is that the DNN outputs a statistically good initial point which then is moved to reach the feasible set. In this chapter, my contributions are as follows:

- I propose two novel DL-based solvers for the classical power control problem in a multi-user interference channel with QoS constraints. The first is called Deep Implicit Projection Network (DIPNet), and the second is called Deep Explicit Projection Network (DEPNet). The former utilizes convex optimization to project the neural network's output to the feasible set defined by the QoS constraints and the latter uses an iterative process to achieve the same.

- The proposed models were trained in an unsupervised manner and compared

with FCNNs like PCNet and ePCNet [12] as DNN-based benchmarks, and Geometric Program (GP) [14] as the optimization-based benchmark. The sum-rate, constraint violation probability, and online test time are used as the performance evaluation metrics.

- Numerical results demonstrate that the proposed DIPNet and DEPNet guarantee zero constraint violation probability while outperforming PCNet and ePCNet in terms of sum-rate at the expense of slightly increased computational complexity.

3.1 System Model and Problem Statement

I consider a downlink wireless network composed of B single-antenna BSs where each BS can serve Q users at maximum in Q orthogonal frequency channels. Due to orthogonal channel allocation at each BS, the users that are being served by the same BS will not interfere with each other, i.e., no intra-cell interference exists. However, the BSs share the same frequency spectrum and they equally distribute the bandwidth among their users, denoted by W. Thus, the inter-cell interference on each channel of bandwidth W exists from the neighboring BSs. Without the loss of generality, I consider a total of U single-antenna users in the system, where $U = BQ$. The achievable rate of the user associated to channel q of BS b can thus be modeled as follows:

$$R_{b,q}(\mathbf{P}, \mathbf{H}) = W \log_2 \left(1 + \gamma_{b,q}(\mathbf{P}, \mathbf{H})\right),$$

where

$$\gamma_{b,q}(\mathbf{P}, \mathbf{H}) = \frac{|H_{b,q,b}|^2 P_{b,q}}{\sum_{\hat{b}=1,\hat{b}\neq b}^{B} |H_{b,q,\hat{b}}|^2 P_{\hat{b},q} + \sigma^2},$$

where $H_{b,q,\hat{b}}$ denotes the interfering channel between the BS $\hat{b}$ and the user who is assigned to channel q of BS b, $P_{b,q}$ denotes the transmit power allocated to the user scheduled on channel q of BS b, and $\gamma_{b,q}$ denotes the received Signal-to-Interference-to-Noise ratio (SINR) of the user scheduled on channel q of BS b. Note that $\mathbf{P} \in \mathbb{R}^{B \times Q}$ and $\mathbf{H} \in \mathbb{C}^{B \times Q \times B}$ denote the matrix and tensor containing all values of the transmit powers and complex channel gains composed of distance-based path-loss, shadowing, and fading, respectively. I assume that the perfect channel state information (CSI) is available at the BS side. Also, σ^2 refers to the thermal noise power at the users' receivers, which is the same for all the users. I denote the set of all BSs and channels as $\mathcal{B} = \{1, \cdots, B\}$ and $\mathcal{Q} = \{1, \cdots, Q\}$, respectively. The sum-rate maximization problem with QoS constraints can then be formulated as follows:

$$\begin{aligned} \underset{\mathbf{P}}{\text{maximize}} \quad & R(\mathbf{P}, \mathbf{H}) = \sum_{b=1}^{B} \sum_{q=1}^{Q} R_{b,q}(\mathbf{P}, \mathbf{H}) \\ \text{subject to} \quad & P_{b,q} \geq 0, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q} \\ & \sum_{q=1}^{Q} P_{b,q} \leq P_{\max}, \quad \forall b \in \mathcal{B} \\ & R_{b,q}(\mathbf{P}, \mathbf{H}) \geq \alpha_{b,q}, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q} \end{aligned} \tag{3.1}$$

where the first constraint ensures non-negative power allocations and the second constraint refers to the transmit power budget of each BS. Without loss of generality, I assume the same maximum power budget $P_{\max}$ for all the BSs. The third constraint refers to the minimum rate requirement of the user scheduled on the channel q of BS b ($\alpha_{b,q}$). The problem in (3.1) is known to be NP-hard and non-convex both in its objective and the constraints set [8]; thus, finding an optimal solution is challenging.

3.2 Problem Transformation and Differentiable Projection Framework

Since the implicit projection method, introduced in Section 3.3, requires the convexity of the constraints and the explicit projection, introduced in Section 3.4, provides improved results when the feasible set is convex, I first transform the problem to its equivalent form with convex constraints. However, the non-convexity still arises from the non-convex objective function.

3.2.1 Problem Transformation

The considered power control problem in (3.1) can be reformulated in two different ways, i.e., either using the matrix version of power or using the vector form of the power allocations. The matrix form of (3.1) is shown below:

$$\begin{aligned} \underset{\mathbf{P}}{\text{maximize}} \quad & R(\mathbf{P}, \mathbf{H}) \\ \text{subject to} \quad & \mathbf{P} \geq \mathbf{0} \\ & \mathbf{P}.\mathbf{1} \leq P_{\max}\mathbf{1} \\ & \gamma_{b,q}(\mathbf{P}, \mathbf{H}) \geq \beta_{b,q} \end{aligned} \tag{3.2}$$

where $\mathbf{1} = [1, 1, \cdots, 1]^T$ is a B-dimensional vector and $\beta_{b,q} = 2^{\frac{\alpha_{b,q}}{W}} - 1$ is the minimum SINR to get the minimum rate requirement. To reformulate the problem in the vector form, I convert $\mathbf{P}$ in a vector form using the following **Definition-1**.

Definition 1: For a given $m \times n$ matrix $\mathbf{A}$ on a given field $\mathbb{F}$, i.e., $\mathbf{A} \in \mathbb{F}^{m \times n}$, the vectorization operation, $\mathbf{a} = \text{Vec}(\mathbf{A}) : \mathbf{F}^{m \times n} \longrightarrow \mathbf{F}^{mn \times 1}$ is defined as $A_{i,j} = a_{(j-1)m+i}$. The resulting vector $\mathbf{a} \in \mathbb{F}^{mn \times 1}$ is a column vector obtained by stacking the columns of $\mathbf{A}$ on top of each other.

Subsequently, $\mathbf{p}$ can be derived by stacking the Q columns of matrix $\mathbf{P}$ on top of each other using **Definition 1**, i.e.,

$$P_{b,q} = p_{(q-1)B+b}, \tag{3.3}$$

The vector form of (3.2) is shown below:

$$\begin{aligned} \underset{\mathbf{p}}{\text{maximize}} \quad & R(\mathbf{p}, \mathbf{H}) \\ \text{subject to} \quad & \mathbf{p} \geq \mathbf{0} \\ & \mathbf{A}\mathbf{p} \leq P_{\max}\mathbf{1} \\ & \mathbf{C}\mathbf{p} \geq \mathbf{d} \end{aligned} \tag{3.4}$$

where $\mathbf{A} \in \mathbb{R}^{B \times BQ}$ is defined as follows:

$$A_{i,j} = \begin{cases} 1 & \text{if } i \equiv j \pmod{B} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

The problem in (3.4) offers a linear (thus convex) formulation of the third constraint. It is noteworthy that in the case of a single channel available per BS, i.e., $Q = 1$, the formulation of $\mathbf{C}$ and $\mathbf{d}$ is straightforward and presented in traditional power control works [8]. Nevertheless, when considering the case where each BS can serve multiple users at a time and has a certain predefined users' quota, the matrix $\mathbf{C} \in R^{U \times U}$ becomes a block-diagonal matrix, where each block is related to one of the Q channels.

The q-th block denoted by $\mathbf{M}^q \in \mathbb{R}^{B\times B}$ is defined as follows:

$$M^q_{b,\hat{b}} = \begin{cases} |h_{b,q,b}|^2 & \text{if } b = \hat{b} \\ -\beta_{b,q}|h_{b,q,\hat{b}}|^2 & \text{if } b \neq \hat{b} \end{cases}, \tag{3.6}$$

where $\mathbf{C} = \text{diag}(\mathbf{M}^1, ..., \mathbf{M}^Q)$, and $\mathbf{d} \in \mathbb{R}^{U\times 1}$ is also derived by stacking the columns of matrix $\mathbf{D} \in \mathbb{R}^{B\times Q}$, defined below, on top of each other, i.e.,

$$D_{b,q} = \beta_{b,q}\sigma^2, \quad \mathbf{d} = [\mathbf{D}^T_{:,1}, ..., \mathbf{D}^T_{:,Q}]^T, \tag{3.7}$$

The aforementioned transformations also show that the third constraint can be expressed as either a non-convex, non-linear, or linear constraint in (3.1), (3.2), and (3.4), respectively [8]. In the following, I will leverage the aforementioned transformations in Section 3.3 and Section 3.4.

3.2.2 Functional Optimization Form

Despite the linearity of the constraints in (3.4), this problem is NP-hard due to the non-convex objective function [8]. Thus, finding an optimal solution is challenging. Traditionally, (3.4) is solved for each channel realization, i.e., for each realization of $\mathbf{H}$, I solve (3.4) to get $\mathbf{p}$ which dictates an implicit mapping between $\mathbf{H}$ and $\mathbf{p}$. Although effective, this variable optimization approach yields high computational complexity. To overcome this problem, I can approximate the implicit mapping between $\mathbf{H}$ and $\mathbf{p}$ with an explicit function. This will significantly improve the computational complexity as long as the explicit function has an efficient implementation, e.g., using neural

networks [5]. The equivalent functional optimization form of (3.4) is [7]:

$$\begin{aligned} \underset{F(\mathbf{H})}{\text{maximize}} \quad & \mathbb{E}_{\mathbf{H}\sim p(\mathbf{H})}[R(F(\mathbf{H}),\mathbf{H})] \\ \text{subject to} \quad & F(\mathbf{H}) \geq \mathbf{0}, \quad \forall \mathbf{H} \in \mathbb{H} \\ & \mathbf{A}F(\mathbf{H}) \leq P_{\max}\mathbf{1}, \quad \forall \mathbf{H} \in \mathbb{H} \\ & \mathbf{C}F(\mathbf{H}) \geq \mathbf{d}, \quad \forall \mathbf{H} \in \mathbb{H} \end{aligned} \tag{3.8}$$

where $F(\cdot)$ represents the functionality that maps CSI to a power allocation. It has been proven in [7] that the solution of (3.8) is also the optimal solution of (3.4). The same transition can be written for (3.2). The output of $F(\cdot)$ is a matrix for (3.2) and a vector for (3.4).

DNNs have been shown to be a very rich family of parametric functions, in a sense that even an FCNN has universal function approximation property [19], and has shown success in approximating the aforementioned mapping in supervised and unsupervised ways. Thus, I consider them for approximating F, i.e. $F(\mathbf{H}) = \mathcal{N}_p(\mathbf{H};\mathbf{w}_p)$ where $\mathcal{N}_p$ is a DNN, and $\mathbf{w}_p$ is a vector containing all trainable parameters, i.e., weights and biases, of the DNN. The output of the DNN appears as a subscript to the DNN and its parameters. For example, if the output of the DNN is variable $\mathbf{y}$, the DNN and its parameters are denoted by $\mathcal{N}_y$ and $\mathbf{w}_y$, respectively.

As a result, using the statistical learning theory, the problem of finding $\mathbf{w}_p$ via learning can thus be formulated as:

$$\begin{aligned} \underset{\mathbf{w}_p}{\text{minimize}} \quad & \mathbb{E}_{\mathbf{H}\sim p(\mathbf{H})}[l(\mathcal{N}_p(\mathbf{H};\mathbf{w}_p),\mathbf{H})] \\ \text{subject to} \quad & \mathbf{w}_p \in \mathbb{W} \end{aligned} \tag{3.9}$$

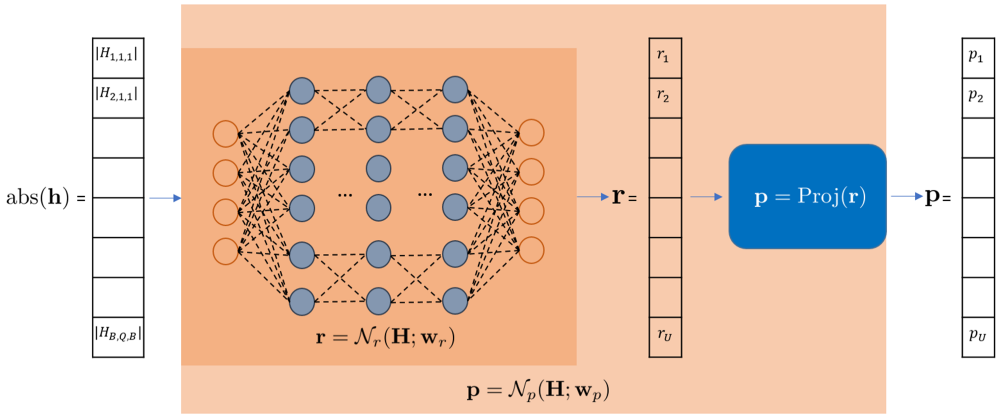

Figure 3.1: A graphical illustration of the proposed differentiable projection framework.

where l is the loss function and measures how good the output of the neural network is for a given data $\mathbf{H}$. Importantly, the design of the loss function is critical to solving the constrained optimization problem. Most of the current research typically incorporates the power budget constraint into the output of the DNN by using bounded activation functions like Sigmoid [12]. Other constraints are generally incorporated by customizing the loss function using the dual problem formulation. The downside of this choice is that there is no easy way to make sure the output of the neural network always meets the constraints and lies in the feasible set of problem (3.4). To overcome this issue, in what follows, I present a differentiable projection-based framework that projects the DNN's output into the feasible set of (3.8).

3.2.3 Differentiable Projection Framework

In this paper, I focus on designing a projection framework where I can add a special layer to the DNN, which projects the output of the DNN to the feasible set of (3.4)

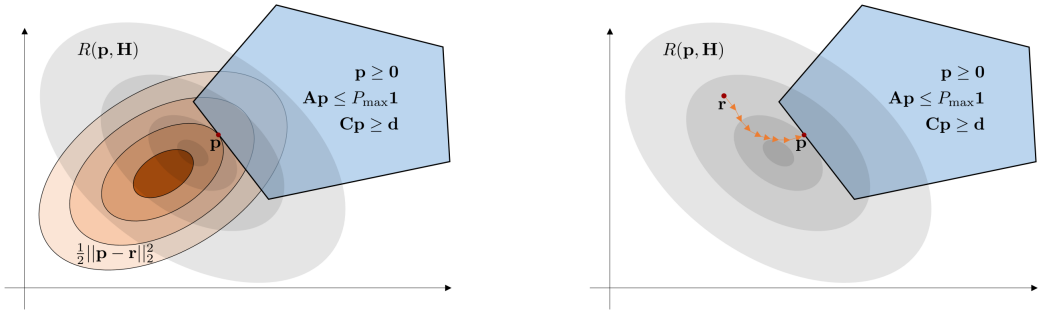

Figure 3.2: An illustration of the proposed projection methods: Implicit projection via mathematical optimization (left) - Explicit projection via an iterative process (right).

or (3.8). I refer to this transformation as *Projection*, i.e.,

$$\mathbf{p} = \mathcal{N}_p(\mathbf{H}; \mathbf{w}_p) = \text{Proj}(\mathbf{r}); \quad \mathbf{r} = \mathcal{N}_r(\mathbf{H}; \mathbf{w}_r), \tag{3.10}$$

where $\mathbf{r}$ is the output of the backbone DNN without the projection layer, and Proj : $\mathbb{R}^U \longrightarrow \mathbb{R}^U$ is a projection function unto the feasible set of (3.8). Fig. 3.1 shows the graphical illustration of this framework. Since I do not consider parametric projection functions, the parameters of $\mathcal{N}_p$ and $\mathcal{N}_r$ are considered as the same, i.e., $\mathbf{w}_p = \mathbf{w}_r$.

Remark: A function Proj : $\mathbb{R}^U \longrightarrow \mathbb{R}^U$ is a differentiable projection function w.r.t. (3.8), if its Jacobian can be evaluated, i.e., $\frac{\partial \mathbf{p}}{\partial \mathbf{r}}$ and $\mathbf{p}$ meets the constraints of (3.8). Being differentiable is critical to make the end-to-end training of the DNN possible using gradient-based methods [17].

The projection function can be defined implicitly or explicitly as in the following, respectively.

- **Differentiable Implicit Projection:** in which a differentiable convex optimization (DCO) layer, a type of implicit layer in DNN, is applied to project the output of the $\mathcal{N}_r$ to the feasible set. By doing this, I make sure that the

output of $\mathcal{N}_p$ always satisfies the constraints. A detailed explanation of DNN architecture is in Section IV. Now that the output of DNN always satisfies the constraints, $l(\mathcal{N}_p(\mathbf{H}; \mathbf{w}_p), \mathbf{H})$ can take the form of $-R(\mathcal{N}_p(\mathbf{H}; \mathbf{w}_p), \mathbf{H})$ to directly optimize (3.8).

- **Differentiable Explicit Projection:** uses a differentiable iterative process to realize the projection function (Proj) and moves the output of the $\mathcal{N}_r$ closer to the feasible set. Each iteration uses a process, called correction process [2], which *corrects* the previous output towards lesser violation of the constraints. This approach uses soft-loss during training and shows faster performance relative to the first approach at the expense of the lack of the provable feasibility of the results. Experimental evaluations, however, confirm the zero constraint violation probability, as detailed in Section VI.

Remark: Let $f : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ be a function such that $\mathbf{y} = f(\mathbf{x})$. If the process of evaluating the output $\mathbf{y}$ from an input $\mathbf{x}$ is known and the derivative of the output w.r.t. the input is calculated explicitly, I refer to the function as **explicit**, whereas the **implicit** function means that the output evaluation process from the input is unknown to the user and the derivatives are calculated implicitly at the solution point. The term Explictly-defined is mostly used when a closed-form analytic expression of a function is available, which might contradict its usage for an iterative process. However, in this work, I use it to highlight the use of explicit and implicit differentiation.

3.3 Differentiable Implicit Projection Framework

In this section, I describe the systematic incorporation of the QoS constraint into the DNN architecture. To be more specific, I utilize the newly introduced DCO layer [21] to project the output of the neural network to the feasible set defined by the QoS or minimum rate constraints. I call this layer a projection layer. DCO and other types of layers that describe an implicit functionality between input and output space lie under the umbrella of implicit layers, as explained in chapter 2. In the following, a brief description of the projection layer is provided followed by the proposed projection function and the neural network architecture.

3.3.1 Projection Layer

I define the projection function implicitly using the concept of DCO which requires reformulating the constraints of the original problem as convex constraints and choosing a convex objective function for this layer. Using the affine formulation of the constraints of (3.1), as defined in (3.4) or (3.8), the optimization problem that characterizes the projection layer is formulated as:

$$\begin{aligned} \mathbf{p} = \underset{\hat{\mathbf{p}}}{\operatorname{argmin}} \quad & \frac{1}{2}||\hat{\mathbf{p}} - \mathbf{r}||_2^2 \\ \text{subject to} \quad & \hat{\mathbf{p}} \geq \mathbf{0} \\ & \mathbf{A}\hat{\mathbf{p}} \leq P_{\max}\mathbf{1} \\ & \mathbf{C}\hat{\mathbf{p}} \geq \mathbf{d} \end{aligned} \tag{3.11}$$

where (3.11) implicitly defines the projection function ($\mathbf{p} = \text{Proj}(\mathbf{r})$). This projection takes the form of the euclidean projection [31] unto a set, defined by the constraints

of (3.8). Given the implicit function, theorem [20–22], the Jacobian of the output w.r.t. the input, i.e., $\frac{\partial \mathbf{p}}{\partial \mathbf{r}}$, can be computed, regardless of how the solution is derived. Moreover, since the constraints of (3.11) and (3.8) are the same, they have the same feasible set, i.e., the solution of (3.11) satisfies the constraints of (3.8). Note that, as long as the convexity is preserved, one can choose other objective functions for (3.11) as well. The main role of this objective is to formalize the similarity between $\mathbf{r}$ and the points in the feasible set of (3.8). Once the formalization is there, the output of the projection function will be the point with the highest similarity. In (3.11), the distance, measured by the euclidean norm, is chosen to measure the similarity, i.e., the lower the distance, the higher the similarity. The upside of this choice is that (3.11) becomes a quadratic program, which can be solved efficiently [31]; thus offering a reasonable evaluation complexity once composed with $\mathcal{N}_r$. Fig. 3.2 (left one) illustrates how (3.11) works. The projection function (3.11) is an instance of the DCO layer. The implementation details of this function and its integration with automatic-differentiation frameworks like PyTorch are available in [21, 23].

3.3.2 Neural Network Architecture

As depicted in (3.10), the overall neural network ($\mathcal{N}_p$) is the composition of the projection function (Proj) and a backbone neural network ($\mathcal{N}_r$). Since the focus of this work is on the design of the projection function, I consider a neural network with fully connected layers as the backbone. The overall architecture is presented in Fig. 3.1. The architecture is composed of fully connected layers with ReLU activation function at the hidden layers. The input to the neural network is the vector form of tensor $\mathbf{H}$ which is obtained using Definition 2 as given below:

Definition 2: For a given tensor $\mathbf{T} \in \mathbb{F}^{m\times n\times p}$, the vectorization operation, $\mathbf{t} = \text{Vec}(\mathbf{T}) : \mathbf{F}^{m\times n\times p} \longrightarrow \mathbf{F}^{mnp\times 1}$, is defined as $T_{i,j,k} = t_{(j-1)mp+(k-1)m+i}$. The vector $\mathbf{t} \in \mathbb{F}^{mnp\times 1}$ is a column vector obtained by first gathering a $mp \times n$ matrix whose column j is obtained by vectorizing the matrix $\mathbf{T}_{:,j,:}$. The resulting $mp \times n$ matrix is then vectorized to get $\mathbf{t}$.

Subsequently, the input dimension is BU. The output will have the same dimension as the power vector, i.e., U. The final layer's activation function is sigmoid to bound the output of $\mathcal{N}_r$ between zero and one. Experiments showed that doing this improves the computation time of the optimization problem of the projection layer (3.11). Since the final neural network ($\mathcal{N}_p$) uses an implicit projection, I refer to it as **D**ifferentiable **I**mplicit **P**rojection **NET**work, or for short **DIPNet**. The proposed framework is agnostic to the DNN architecture; thus, one can extend to other DNN architectures to enhance the performance even further.

3.4 Differentiable Explicit Projection Framework

In this section, I describe another way of incorporating constraints at the DNN's output. Specifically, I design a projection function via an iterative process. This idea is inspired by [2], where a process called correction is applied iteratively to the output of the DNN to make it fall onto the feasible set of inequality constraints. By choosing a differentiable and explicitly-defined correction process, the resulting projection function will be differentiable and explicitly-defined as well, which projects the input to the feasible set of the problem.

3.4.1 Differentiable Iterative Projection

Consider the following formulation of (3.4) or (3.8) , where all the constraints are concatenated to form a single vector and are denoted as $g(\mathbf{p}, \mathbf{H}) \leq \mathbf{0}$, i.e.,

$$\begin{aligned} \underset{\mathbf{p}}{\text{maximize}} \quad & R(\mathbf{p}, \mathbf{H}) \\ \text{subject to} \quad & g(\mathbf{p}, \mathbf{H}) \leq \mathbf{0} \end{aligned} \tag{3.12}$$

where $g : \mathbb{R}^U \times \mathbb{R}^{BU} \longrightarrow \mathbb{R}^G$ contains all the inequality constraints of the main problem ($G = UBU$). Based on (3.4), g is an affine transformation w.r.t. $\mathbf{p}$. Let us define $V_H : \mathbb{R}^U \longrightarrow \mathbb{R}$ as a measure of the constraint violation, i.e.,

$$V_H(\mathbf{p}) = ||\text{ReLU}(\text{g}(\mathbf{p}, \mathbf{H}))||_2^2 \tag{3.13}$$

This means that given two arbitrary points $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^U$ if $V_H(\mathbf{x}_1) \leq V_H(\mathbf{x}_2)$, $\mathbf{x}_2$ violates the constraints of (3.12) more than $\mathbf{x}_1$. In what follows, I define the correction process.

Definition 3: An explicitly defined function $\rho : \mathbb{R}^U \longrightarrow \mathbb{R}^U$ that has the following properties is called correction process: if $\mathbf{y} = \rho(\mathbf{x})$, then $V_H(\mathbf{y}) < V_H(\mathbf{x})$, and $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is calculable. The former condition makes sure that the output of the correction process is closer to the feasible set than its input, and the latter guarantees the differentiability of the correction process.

I also denote ρ^t as applying ρ for t times. Based on the first condition in Definition 3, by applying ρ iteratively, I will end up with a point that meets the constraints i.e.,

$$\mathbf{p} = \text{Proj}(\mathbf{r}) = \lim_{t \longrightarrow \infty} \rho^t(\mathbf{r}). \tag{3.14}$$

Since ρ is explicitly defined and differentiable, ρ^t and Proj are also explicitly defined and differentiable, and their Jacobian can be derived using the chain rule. Thus, the resulting projection function in (3.14) is explicitly defined and follows the definition of the differentiable projection function. In other words, given an input $\mathbf{r} \in \mathbb{R}^U$ and output $\mathbf{p} \in \mathbb{R}^U$ of the projection function, i.e., $\mathbf{p} = \text{Proj}(\mathbf{r})$, $V_H(\mathbf{p}) = 0$ and the Jacobian of $\mathbf{p}$ w.r.t $\mathbf{r}$, i.e. $\frac{\partial \mathbf{p}}{\partial \mathbf{r}}$ is derivable. The former condition enables end-to-end training of the whole system, i.e. letting the gradients of the loss function w.r.t. the DNN's parameters be calculated via backpropagation [17].

Due to the infinite limit, the projection function as defined in (3.14) cannot be realized in practice. Hence, similar to [2], I use the truncated version of it, i.e. $\text{Proj}(.) = \rho^t(.)$ for some finite t. Due to truncation, the output of the projection function may not lie on the feasible set. However, I can speed up the convergence rate of ρ^t by carefully designing ρ, discussed later, and choosing the initial point $\mathbf{r}$. The latter can be handled by making the output of the backbone neural network $\mathcal{N}_r$ closer to the feasible set. To do this, I use the following loss function, called soft-loss, during training, i.e.,

$$l_{\text{soft}}(\mathbf{p}, \mathbf{H}) = -R(\mathbf{p}, \mathbf{H}) + \lambda V_H(\mathbf{p}), \tag{3.15}$$

where $\mathbf{p} = \rho^t(\mathcal{N}_r(\mathbf{H}; \mathbf{w}_r))$, t is a finite number, and λ is a hyperparameter controlling the constraint satisfaction relative to objective optimization. The second term is added to the loss function to penalize the points that violate the constraints, which will make $\mathcal{N}_r$ to output a good initial point to ρ^t. Moreover, since it is not necessary to fully satisfy the constraints during training, fewer iterations can be used to speed up the training process. During the test time, however, the number of iterations is increased to output a feasible point [2].

3.4.2 Design of the Correction Process (ρ)

Following [2], I use gradient-descent-based methods to realize the correction process ρ. Let $\nabla^{V_H}(\mathbf{x}) \in \mathbb{R}^U$ and $\mathcal{H}^{V_H}(\mathbf{x}) \in \mathbb{R}^{U \times U}$ denote the gradient and Hessian of V_H w.r.t. $\mathbf{x}$ and defined, respectively, as follows:

$$\nabla^{V_H}(\mathbf{x}) = \nabla_{\mathbf{x}} ||\max(g(\mathbf{x}, \mathbf{H}), 0)||_2^2 = 2J^g(\mathbf{x}, \mathbf{H})^{\mathrm{T}} \max(g(\mathbf{x}, \mathbf{H}), 0). \tag{3.16}$$

$$\mathcal{H}^{V_H}(\mathbf{x}) = 2I(\max(\mathrm{g}(\mathbf{x}, \mathbf{H}), 0))^{\mathrm{T}} \mathbf{K}^g(\mathbf{x}, \mathbf{H}) + 2\max(g(\mathbf{x}, \mathbf{H}), 0)^{\mathrm{T}} \mathbf{T}^g(\mathbf{x}, \mathbf{H}). \tag{3.17}$$

where $J^g(\mathbf{x}, \mathbf{H}) \in \mathbb{R}^{G \times U}$ is the Jacobian of g w.r.t. $\mathbf{x}$. Moreover, $I : \mathbb{R} \longrightarrow \mathbb{R}$, $\mathbf{K} \in \mathbb{R}^{G \times U \times U}$, and $\mathbf{T} \in \mathbb{R}^{G \times U \times U}$ are defined as shown in the following.

$$I(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}, \quad K^g(\mathbf{x}, \mathbf{H})_{k,i,j} = \frac{\partial g_k}{\partial x_i} \frac{\partial g_k}{\partial x_j}, \quad T^g(\mathbf{x}, \mathbf{H})_{k,i,j} = \frac{\partial^2 g_k}{\partial x_i \partial x_j}. \tag{3.18}$$

where I is an indicator function that is applied element-wise to $\max(\mathrm{g}(\mathbf{x}, \mathbf{H}), 0)$, $\mathbf{K}$, and $\mathbf{T}$ are rank-3 tensors. The definition of Hessian in (3.17) contains vector to tensor dot product, resulting in a $U \times U$ matrix. The product is computed as follows:

$$\mathbf{C} = \mathbf{a}^{\mathrm{T}} \mathbf{Z} \longrightarrow C_{i,j} = \sum_{k=1}^{G} a_k Z_{k,i,j}, \tag{3.19}$$

where $\mathbf{Z} \in \mathbb{R}^{G \times U \times U}$, $\mathbf{a} \in \mathbb{R}^{G \times 1}$, and $\mathbf{C} \in \mathbb{R}^{U \times U}$.

The feasible set of (3.1) can be presented with linear constraints, i.e., a polyhedron, as can be seen in (3.4). Thus, in the following, I derive the above-mentioned formulas in case that g is an affine function, i.e. $g(\mathbf{x}, \mathbf{H}) = \mathbf{M}\mathbf{x} + \mathbf{n}$, where $\mathbf{M} : \mathbb{R}^{G \times U}$ and

$\mathbf{n} \in \mathbb{R}^{G\times 1}$ are functions of $\mathbf{H}$.

$$g(\mathbf{x},\mathbf{H}) = \mathbf{M}\mathbf{x} + \mathbf{n} \longrightarrow \begin{cases} J^g(\mathbf{x},\mathbf{H}) = \mathbf{M} \\ K^g(\mathbf{x},\mathbf{H})_{k,i,j} = M_{k,i}M_{k,j} \\ T^g(\mathbf{x},\mathbf{H})_{k,i,j} = 0 \end{cases} \tag{3.20}$$

Remark: Unlike DIPNet, DEPNet does not have a convexity requirement.

Now that I have access to the first and second-order information of V_H w.r.t. $\mathbf{x}$, the correction process can be formulated as follows:

$$\rho(\mathbf{x}) = \mathbf{x} + \Delta\mathbf{x}, \tag{3.21}$$

where $\Delta\mathbf{x}$ can take the form of variations of descent methods [31]. Examples of which include vanilla gradient descent ($\Delta\mathbf{x} = -\gamma\nabla^{V_H}(\mathbf{x})$) or Newton method ($\Delta\mathbf{x} = -\mathcal{H}^{V_H}(\mathbf{x})^{-1}\nabla^{V_H}(\mathbf{x})$) [31]. Here, I used gradient-descent with momentum [27] during training and the Newton method during test for faster training and lower violation probability at test. In the following, the correction process formulation for these choices are provided.

Let $\Delta\mathbf{x}^t$ be the step of the correction process that is applied at the iteration t of the projection function. The update rule for gradient descent with momentum can be given as follows:

$$\Delta\mathbf{x}^t = -\gamma\nabla^{V_H}(\mathbf{x}) - \mu\Delta\mathbf{x}^{t-1}, \tag{3.22}$$

where γ is called step-size or learning rate and μ is called the momentum. To avoid confusion with the learning rate involved in training DNNs, I refer to γ as the step-size. It should be noted that γ is chosen prior to training and is fixed during the training.

This is to make the execution time of the correction process faster and the calculation of its derivatives easier during the training. In this case, having momentum will help the convergence of gradient descent by making it less vulnerable to the oscillations of noisy gradients [27].

One can use other alternatives like exact or backtracking line-search [31] for choosing γ in each iteration. Although it helps with convergence, it introduces more computational complexity during the training, and the calculation of the derivative of the correction process will not be as straightforward as using a fixed value for the step-size. At the test time, however, I can use these techniques to fasten the convergence rate of the correction process and get feasible solutions from the projection function. Inspired by this idea, instead of using gradient descent at the test time, I use Newton method, which has a faster convergence rate [31]. Since $\max(-, 0)$ is used in the definition of $V_H(.)$ and its second derivative is zero, the hessian matrix is poorly conditioned and has lots of zeros, making it not invertible. To deal with this issue, I regularize the hessian matrix by adding a small value α to its diagonal [28]. Thus, the update rule becomes:

$$\Delta \mathbf{x}^t = -(\mathcal{H}^{V_H}(\mathbf{x}) + \alpha \mathbf{I})^{-1} \nabla^{V_H}(\mathbf{x}) \tag{3.23}$$

where $\mathbf{I} \in \mathbb{R}^{U \times U}$ is the identity matrix. In the following, for simplicity, I refer to (3.23) as the Newton method update rule. (3.23) uses the second-order information of the objective at hand. As proven in [31] and shown in my experiments, Newton method has a faster convergence rate than gradient-descent. The only downside of it is that the calculation of the derivative of its steps is not straightforward due to matrix inversion in (3.23). Thus, I only used it at test time, and utilized gradient

descent during training.

3.4.3 Neural Network Architecture

For consistency and the sake of fair comparison, I use the same DNN architecture, as described in Section IV, with minor modifications. As depicted in Fig. 3.1, after the last affine transformation, there is a sigmoid non-linearity followed by a projection function. Using sigmoid at the final layer of $\mathcal{N}_r$ showed faster convergence of this projection method in the experiments. Since this model uses an explicitly defined differentiable transformation to realize the projection, I call it **D**ifferentiable **E**xplicit **P**rojection **NET**work (DEPNet).

3.5 An enhancement: Frank-Wolfe Algorithm

The optimality of the output of the projection function is dependent on the quality of the initial point ($\mathbf{r}$) and the projection method. Although the quality of the initial point will improve during the training, it might show sub-optimality. This means that the final power profile is feasible, but might not be optimal. A remedy is to apply constrained optimization algorithms, where the output of the projection function is passed as the initial point of the chosen algorithm ($\mathbf{p}^0$). The algorithm, then, outputs an enhanced power profile that achieves a higher sum-rate than $\mathbf{p}^0$.

Among several variants of general constraint optimization algorithms, I select Frank-Wolfe [50], a.k.a conditional gradient descent due to its low computational cost in each iteration of the algorithm. Compared to other alternatives like projected gradient descent which requires solving a quadratic program in each iteration, Frank-Wolfe only solves a linear program in each iteration [50]. In the following, the

description of Frank-Wolfe algorithm is provided.

As mentioned earlier, the initial point of this algorithm will be the output of the projection function, i.e., $\mathbf{p}^0 = \text{Proj}(\mathbf{r})$. The algorithm is agnostic to the projection method and only requires a feasible initial point. Thus, it can be used with both DIPNet and DEPNet. The final power profile will be the output of this algorithm.

Let us denote the output of the Frank-Wolfe algorithm after t and $t+1$ iterations as $\mathbf{p}^t$ and $\mathbf{p}^{t+1} \in \mathbb{R}^U$, respectively. They belong to the feasible set of (3.4) and the relation between them is:

$$\mathbf{p}^{t+1} = \mathbf{p}^t + \lambda \mathbf{p}^{t+\frac{1}{2}}, \quad \lambda \in [0, 1] \tag{3.24}$$

where $\mathbf{p}^{t+\frac{1}{2}}$ is an intermediate variable derived as follows:

$$\begin{aligned} \mathbf{p}^{t+\frac{1}{2}} = \underset{\hat{\mathbf{p}}}{\text{argmin}} \quad & < \hat{\mathbf{p}}, \nabla_p R(\mathbf{p}^t, \mathbf{H}) > \\ \text{subject to} \quad & \hat{\mathbf{p}} \geq \mathbf{0} \\ & \mathbf{A}\hat{\mathbf{p}} \leq P_{\max}\mathbf{1} \\ & \mathbf{C}\hat{\mathbf{p}} \geq \mathbf{d} \end{aligned} \tag{3.25}$$

where the objective is the euclidean dot product of the optimization variable and the gradient of sum-rate ($R(\mathbf{p}, \mathbf{H})$) w.r.t $\mathbf{p}^t$. Since (3.25) is solved over the feasible set of (3.4), $\mathbf{p}^{t+\frac{1}{2}}$ belongs to the feasible set of (3.4) as well. Following (3.24), a convex combination of $\mathbf{p}^{t+\frac{1}{2}}$ and $\mathbf{p}^t$ is chosen as the output of the $t+1$ iteration. Since the feasible set of (3.4) is convex, $\mathbf{p}^{t+1}$ will be a feasible power profile. λ is the step size that can be chosen via line search or adaptive methods [50].

3.6 Experimental Set-up and Benchmarks

In this section, I describe the dataset generation procedure, system set-up, and relevant benchmarks to evaluate the performance of the proposed DIPNet and DEPNet.

3.6.1 DataSet Generation

For the experimental evaluation, I created two datasets. The first one, called *Gaussian Dataset*, follows the symmetric interference channel model with independent and identically distributed (i.i.d.) Rayleigh fading for all channels coefficients, i.e., $h_{b,q,\hat{b}} \sim \mathcal{CN}(0,1)$. The maximum power is set to 1 mW, the transmission bandwidth is 5 MHz, and the thermal noise power is assumed to be $0.01\mu W$ [51]. This model is widely adopted in the literature to evaluate resource allocation algorithms [11,12,51]. The second dataset, called *Path-loss Dataset*, has channel coefficients that are composed of large-scale path-loss, shadowing, and fading. The dataset generation has the following steps. First, the locations of the BSs and users are determined by random sampling from a 500 m $\times$500 m area. I consider that the distance between the two closest BSs, BSs and user, and between closest users, should be at least 100 m, 5 m, and 2 m, respectively. The carrier frequency is 2.4 GHz, the transmission bandwidth is 5 MHz, and the noise spectral density is set to -169 dBm. The remaining details of the channel model can be found in [52].

For both datasets, once a datapoint or channel realization is generated, I associate the best Q users to the first BS in terms of their channels. These users are then excluded from the user set. Then, the users' of the next BS are determined by the same process. This process is continued until all the users are associated with a BS. After that, for each BS, the allocated set of users is sorted out based on their channel

gains, and then gets assigned to the channels such that the first channel of each BS has the strongest user of that BS.

Different datasets are generated under different configurations. For each configuration, 100,000 data samples are generated and divided to train, validation, and test with 0.9, 0.05, 0.05 ratios, respectively. Unless stated otherwise, the number of BSs is set to B =4. Also, I set the minimum users' rate requirement to $\alpha = 2.5$ Mbps unless stated otherwise.

To provide a comprehensive quantitative analysis, additional datasets are generated under different configurations listed in Table II. Dataset ID is used to refer to different configurations, where the quota of each BS is given as $Q = U/B$. All datasets in Table II are following the Path-loss models. The number of BSs, users, and the minimum rate are different among them.

3.6.2 Feasibility Check

In the dataset preparation process, I need to make sure that all data points are feasible, i.e., there exists a power profile that meets the constraints of problem (3.1). For that, I use the approach mentioned in [8] and used in [12, 51]. The approach provides feasible transmit powers, as well as minimum, transmit powers that can fulfill the minimum rate constraints. Originally, this approach is not designed for the case where there is more than one channel, i.e., $Q > 1$. However, since channels on a given BS are orthogonal, I can break (3.1) into Q sub-problems, and apply the feasibility check approach with slight modifications. The description of the approach

is defined next. Let us define matrix $\mathbf{B}^q$ as follows:

$$B^q_{b,\hat{b}} = \begin{cases} 0 & b = \hat{b} \\ \frac{\beta_{b,q}|h_{b,q,\hat{b}}|^2}{|h_{b,q,b}|^2} & b \neq \hat{b} \end{cases}, \tag{3.26}$$

If the maximum eigenvalue of $\mathbf{B}^q$ is larger than 1, there is no feasible solution. That is, I can not find a power vector for channel q that can satisfy the minimum rate requirement of the users associated with this channel. Otherwise, I can find a feasible power allocation profile as:

$$\mathbf{P}_{:,q} = (\mathbf{I} - \mathbf{B}^q)^{-1}\mathbf{u}^q, \tag{3.27}$$

where $\mathbf{P}_{:,q}$ is the transmit power vector over channel q, $\mathbf{I}$ is a $B \times B$ identity matrix, and $\mathbf{u}^q$ is a $B \times 1$ vector with jth element given as $u^q_b = \frac{\beta_{b,q}\sigma^2}{|h_{b,q,b}|^2}$. Once all power vectors are calculated for $q \in \mathcal{Q}$, I create the final power matrix in the following manner. If $\mathbf{P}$ meets the first and second constraints of (3.1), i.e., all elements of $\mathbf{P}$ are greater than zero and the sum of each row is lesser than $P_{\max}$, $\mathbf{P}$ is a feasible solution of (3.1). Otherwise, (3.1) is not feasible.

3.6.3 Benchmarks

I consider two main benchmarks. The details of these benchmarks are given below:

- **PNet:** is a neural network model exactly like DIPNet and DEPNet, but without the projection layer, i.e. FCNN. This model is an extension of PCNet [12] that works for the multi-channel scenario. The power budget constraint was handled by having softmax as the activation function of the final layer. The violation of QoS constraint was added as a penalty term to the loss function, similar to the soft loss as in Section VI. This DNN-based benchmark does not utilize the

proposed projection methods.

- **GP:** is an optimization-based solution that uses the high-SINR assumption to transform the main problem (3.1) to a geometric program [14]. The high-SINR assumption can be relaxed by using GP with successive convex approximation technique which results in solving series of GPs until finding a stationary point of (3.1) [14]. Other alternatives like MAPEL [51] perform an implicit exhaustive search to find the globally optimum point of (3.1). Since these methods result in non-practical computational complexity, which dramatically increases with problem size [12], I chose GP with high-SINR assumption as the optimization-based benchmark.

3.7 Numerical Results and Discussions

In this section, I present the performance of the proposed methods (DIPNet and DEPNet) with the conventional benchmarks. The considered performance metrics include network sum-rate, QoS violation probability, and per sample test time. To get a good estimate of the computation time, the overall computation time of each model is measured over the test set. Per-sample computation time is then calculated by averaging over the data points.

In all the experiments, an FCNN with three 200-dimensional hidden layers is used as the backbone of DIPNet and DEPNet ($\mathcal{N}_r$). Batch normalization [53] and Dropout [54] are used to accelerate the training and prevent over-fitting. The same neural network is used for the PNet as well. For the correction process of DEPNet, I used gradient descent with momentum for the training and Newton method for the testing. The momentum is set to 0.5 for all the datasets, and the step size is

fine-tuned for each dataset individually from the interval of 0.5 to 0.0005. Similarly, the parameter λ in soft-loss (3.15) is chosen from the interval of 10 to 10000 for each dataset individually. The number of iterations in the iterative projection is set to 5 and 100 for the training (with Gradient Descent method) and testing (with Newton method), respectively. α, used for regularizing the hessian in Newton method (3.23), is set to 10^{-8} for all the experiments.

For all DNNs, I used learning rate of 0.001, batch size of 10, and learning rate decay rate of 0.99. ADAM is also used for training the DNNs. All DNNs are trained for 20 epochs, and early stopping is used to pick the model with the best performance. That is, after each epoch I check the performance of the model over the validation set based on a metric like sum-rate. Then, I pick the model that has the best performance among all epochs. The performance metric for DIPNet and DEPNet is the network sum-rate. For the PNet, I pick the model that has the minimum QoS violation probability. This is because choosing network sum-rate as the selecting criteria will result in a model that has a significant QoS violation probability, which will not be comparable with DIPNet and DEPNet. Moreover, based on the experiments, λ in the soft-loss of PNet is set to 1000 for Gaussian datasets and 10000 for path-loss datasets. This results in a model that has comparable performance in terms of network sum-rate and QoS violation with DIPNet and DEPNet.

For the implementation, I used PyTorch [55] as the automatic differentiation engine. For implementing the implicit projection in DIPNet, I used the CVXPYlayer package [21]. ECOS [56] is chosen as the optimization solver of this layer among the available options. To make the implementation consistent, GP is implemented in Python using CVXPY package [57]. MOSEK [16], a commercial solver, is used as the

Table 3.1: A comparative analysis of the proposed DIPNet, DEPNet, and optimization-based benchmark in terms of sum-rate (in Mbps) and computation time (in msec).

Dataset ID	**Configuration**				**DIPNet**		**DEPNet**		**GP**	
	BSs	Users	Quota	Target Rate	Sum-rate	Time	Sum-rate	Time	Sum-rate	Time
1	5	5	1	10	104.15	2.91	107.35	0.69	119.6	478.47
2	2	6	3	2.5	243.35	2.70	244.6	0.46	244	224.22
3	4	20	5	5	510.05	4.06	515.2	3.06	569.6	7852.19
4	4	20	5	2.5	456.1	4.02	458.95	3.05	512.9	7864.58
5	6	24	4	5	441.25	5.85	444.2	4.99	515.4	35607.32
6	6	24	4	2.5	397.7	6.05	399.85	5.81	466.55	36338.41
7	5	5	1	2.5	55.7	2.92	57.4	0.66	68.35	530.52
8	4	12	3	2.5	258.7	3.82	261.9	1.73	297	2244.52

backbone solver of CVXPY for both GP and Frank-Wolfe. The maximum number of iterations for Frank-Wolfe is set to 50 and the threshold is set to 0.001. Since the output of DIPNet is always feasible, I only applied Frank-Wolfe to the output of DIPNet, referred to as DIPNet+FW. The experiments are done on a desktop computer with an Intel Core i7-8700 CPU 3.20GHz and 8GB of RAM.

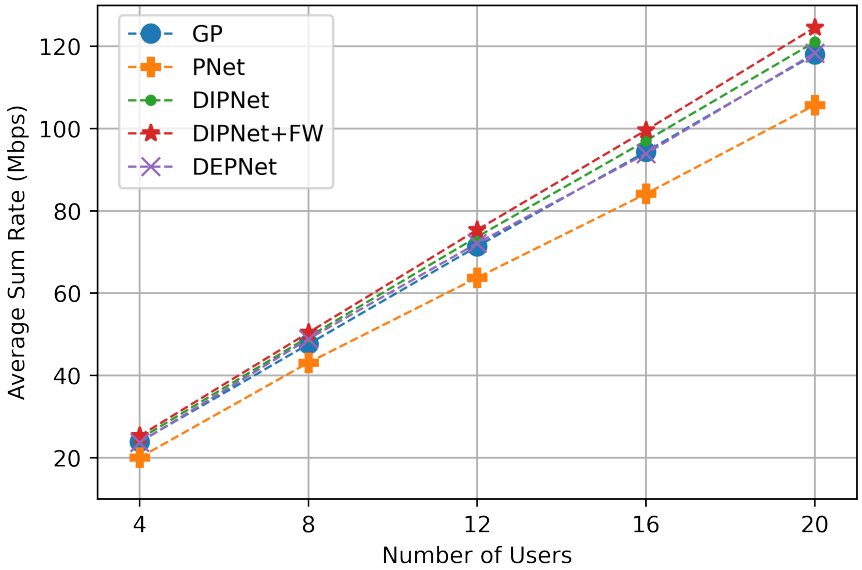

Figure 3.3: A comparison of sum-rate for GP, PNet, DIPNet, DIPNet+FW, and DEPNet considering Gaussian datasets.

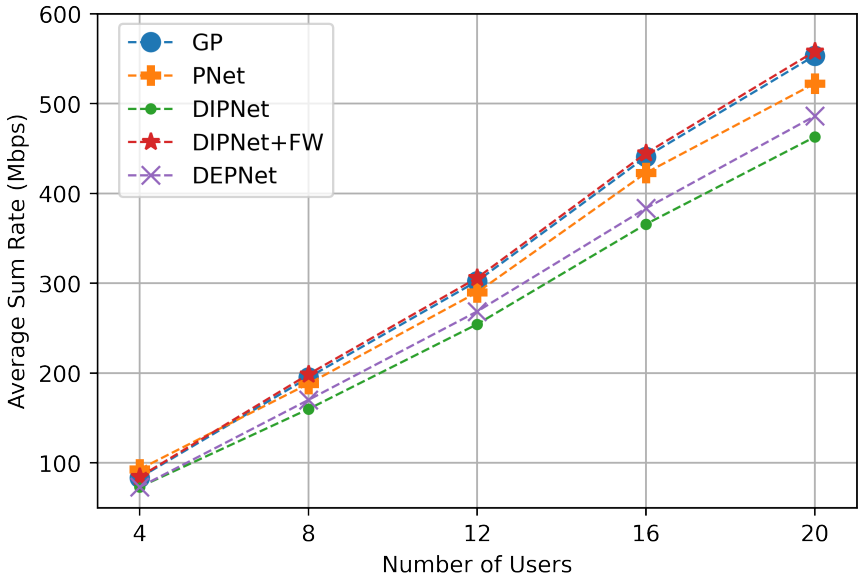

Figure 3.4: A comparison of sum-rate for GP, PNet, DIPNet, DIPNet+FW, and DEPNet considering Path-loss datasets.

3.7.1 A Comparison to Optimization-based Benchmark

Starting with the Gaussian datasets, Fig. 3.3 demonstrates the performance of DIPNet and DEPNet in terms of the achievable aggregate network sum-rate as compared to the conventional GP-based optimization solution. Both DIPNet and DEPNet demonstrate a close sum-rate to GP while showing zero QoS violation probability (as shown in Fig. 3.5 (left)). By increasing the problem size, the required computation time of GP increases drastically (as shown in Fig. 3.6 (left)). DIPNet and DEPNet, on the other hand, have much reduced computational complexity.

Considering the Path-loss datasets, Fig.3.4 shows that GP outperforms both DIPNet and DEPNet in terms of network sum-rate. The difference in network sum-rates

starts to grow as the problem size increases. When it comes to feasibility and constraint satisfaction, the solutions of GP are always feasible, i.e., the constraint violation probability is zero. The same is true for DIPNet and DEPNet (as shown in Fig. 3.5 (right)). The computation time complexity follows the same trend as Gaussian case and the time complexity of GP increases exponentially when the problem size increases (as shown in Fig. 3.6 (right)). Once the Frank-Wolfe algorithm is applied to DIPNet results, I can observe an apparent boost in the network sum-rate for both Gaussian and Path-loss datasets (as can be seen in Fig. 3.3 and 3.4). Compared to GP, it can be seen that the results of DIPNet with Frank-Wolfe enhancement surpasses GP across all the datasets. The cost of this boost is an increase in computation time compared to DIPNet (as can be seen in Fig. 3.6). Although the computation time of Frank-Wolfe-based DIPNet is higher than other DNN-based methods, it is still lower than GP, especially when the number of users increases (as in Fig. 3.6).

Finally, Table 3.1 provides more experimental results of the DIPNet, DEPNet, and GP across various network configurations. Similar to Fig 3.4, GP also, outperforms DEPNet and DIPNet in terms of network sum-rate. The difference becomes less significant when the minimum rate and the problem size are small. The main reason behind the better performance of GP is that the proposed projection methods tend to find points at the boundary of the feasible set. GP, on the other hand, can search within the feasible set to find solutions with a higher network sum-rate.

3.7.2 A Comparison to Conventional PNet (Enhanced PCNet)

As shown in Fig. 3.3, PNet always outputs a solution that achieves a lower sum-rate than DIPNet and DEPNet for Gaussian dataset. On the other hand, for Path-loss

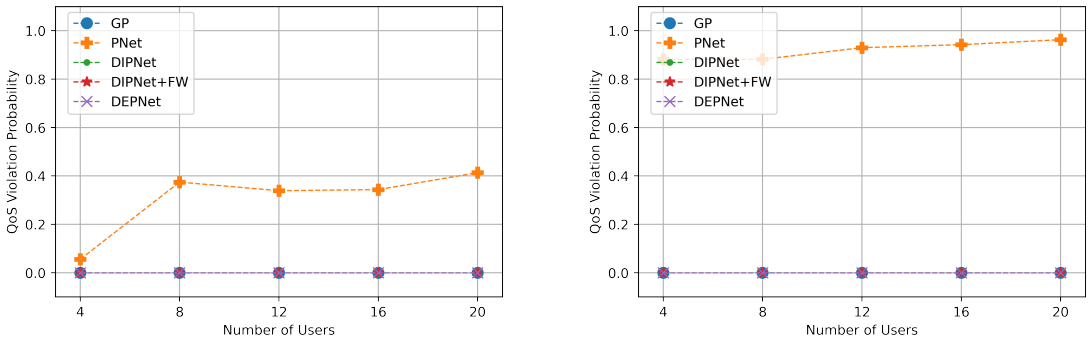

Figure 3.5: QoS violation probability for GP, PNet, DIPNet, DIPNet+FW, and DEPNet (Left: Gaussian Dataset - Right: Pathloss Dataset).

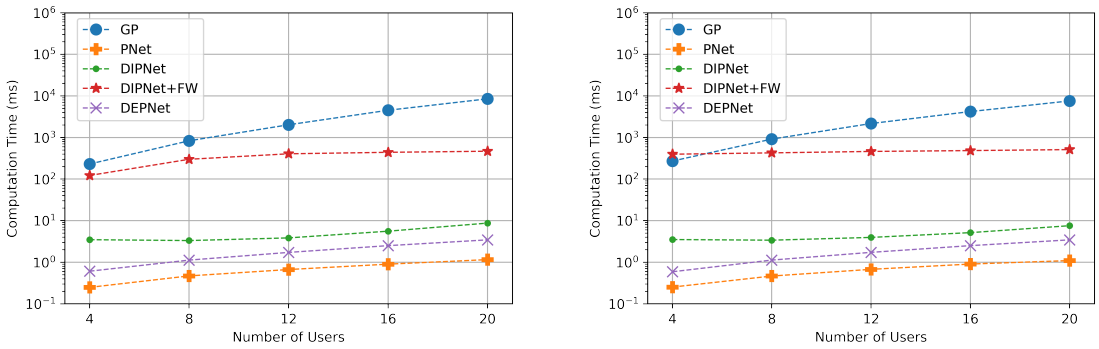

Figure 3.6: Computation time comparison for GP, PNet, DIPNet, DIPNet+FW, and DEPNet (Left: Gaussian Dataset - Right: Pathloss Dataset).

datasets, PNet attains a higher network sum-rate than DEPNet and DIPNet. The sum-rate performance of PNet can be improved by reducing the γ in the soft-loss. However, that will result in an increase in the QoS violation probability. In Fig 3.5, I note that the violation probability of PNet increases with the dimension of the problem (number of users). Comparing Path-loss and Gaussian datasets, I can see that the QoS violation probability of PNet increases dramatically while working on a more realistic dataset, i.e. Path-loss dataset. This is due to the fact that there is no mechanism in the architecture of PNet to satisfy the QoS constraints. Since PNet doesn't utilize any projection functionality, the computation time of PNet is lesser

than DIPNet and DEPNet (as can be seen in Fig. 3.6).

Considering all these factors, I can conclude that PNet is the best option when there is no complex hard constraint like QoS. However, once the QoS constraint is introduced, a projection function has to be utilized to properly handle this constraint and reduce the violation probability.

3.7.3 DIPNet vs DEPNet

The main difference between DIPNet and DEPNet is the projection function used in them to satisfy the QoS constraint. Since the DIPNet uses an optimization solver to incorporate the constraint, the violation probability is always zero. Moreover, it is easier to implement and does not require tuning some hyperparameters for the projection function.

DEPNet, however, uses an iterative process to realize the projection functionality. Thus, the feasibility of the final solutions is a function of the iterative process convergence rate. I used gradient descent with momentum during the training with five iterations. This choice is computationally efficient and leads to fast training. However, the step-size of gradient-descent needs to be tuned for each dataset; thus requiring experimentation. At the test time, I used Newton method, which has a faster convergence rate than gradient descent, with 100 iterations to make sure the output is always feasible. As shown in Fig. 3.5, DEPNet achieves zero violation probability and is more computationally efficient than DIPNet (as can be noted from Fig. 3.6).

When it comes to sum-rate, the performance of DIPNet is better than DEPNet in Gaussian datasets but is worse for the Path-loss dataset. This is because the

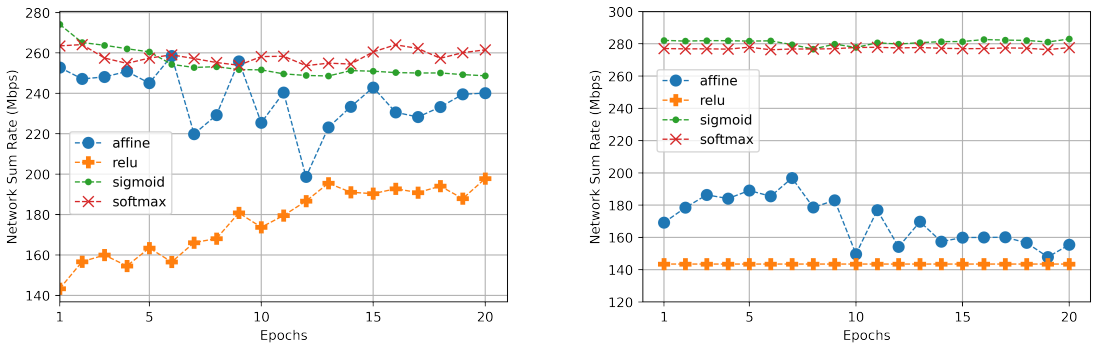

Figure 3.7: The comparison of sum-rate during training using different activation functions for the last layer of (Left: DIPNet - Right: DEPNet).

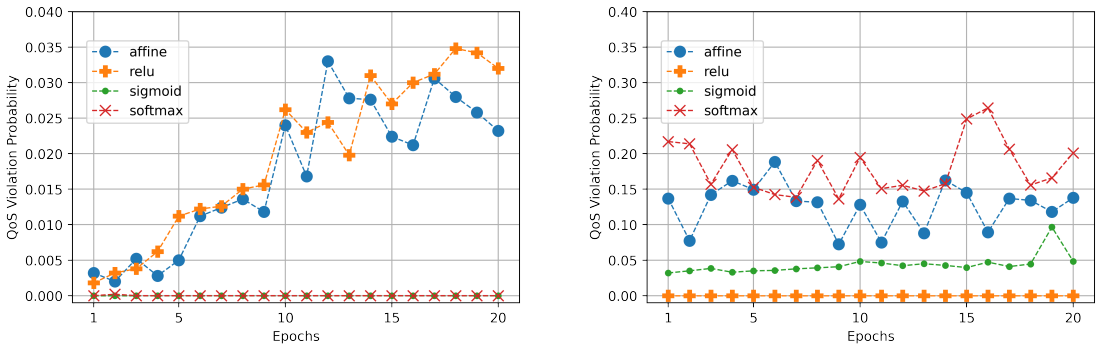

Figure 3.8: The comparison of constraint violation probability during training using different activation functions for the last layer of (Left: DIPNet - Right: DEPNet).

iterative process used to define the projection function in DEPNet provides a better gradient for the backbone neural network. The same trend can be observed in Table 3.1. Although tempting, the better performance of DEPNet comes at the cost of the careful configuration of the parameters of its projection function.

The computation time of DEPNet is faster than DIPNet in all the experiments consistently (Fig. 3.6). This is because DIPNet requires solving a quadratic program in each step. Moreover, since the iterative process used in DEPNet is based on Newton method and only requires gradient, Hessian, matrix inversion, and matrix multiplication, DEPNet can be run on GPU as well, which can significantly improve

the computation time of DEPNet. DIPNet, however, uses an optimization solver for the projection function, which is not GPU friendly. Hence, DEPNet is superior to DIPNet in terms of time complexity, but DIPNet is easier from an implementation perspective.

3.7.4 Activation function of the Final layer of the backbone DNN ($\mathcal{N}_r$)

In this section, I examine the effect of using different activation functions for the last layer of DIPNet and DEPNet $\mathcal{N}_r$. The experiments are conducted on dataset number three (12 users) which follows the Path-loss model. For each activation, I recorded the performance (network sum-rate and QoS violation probability) of the model on the test dataset after each epoch. I tested the following activations: affine, i.e. not using any activation, ReLU, Sigmoid, and Softmax, where it is applied in a way to satisfy the second constraint of (3.4) ($\mathbf{Ar} \leq P_{\max}\mathbf{1}$).

Starting with the network sum-rate, I can observe that Sigmoid and Softmax outperform the other activation functions for both DIPNet and DEPNet (as shown in Fig. 3.7). Comparing these two together, I can see that sigmoid reaches higher network sum-rate for DIPNet and DEPNet. Coming to the QoS violation probability, I can see that ReLU and Sigmoid has the least violation in DEPNet (Fig. 3.8 (Right)), and Softmax and Sigmoid have the least violation in DIPNet (Fig. 3.8 (Left)). Thus, I can conclude that Sigmoid has the best performance overall.

3.7.5 Gradient Descent (GD) vs Newton method

In this section, I show a comparison of the convergence rate of gradient-descent (GD) with different step-sizes and Newton method. The experiments are done on the test

set of dataset number three which follows the Path-loss model. The input to the algorithms ($\mathbf{r}$) is the output of the backbone neural network ($\mathcal{N}_r$) before training and after training. The training is conducted with the step-size of 0.007 as it has the best convergence rate among the others. Moreover, the momentum is set to 0.5 for all the gradient-descent configurations.

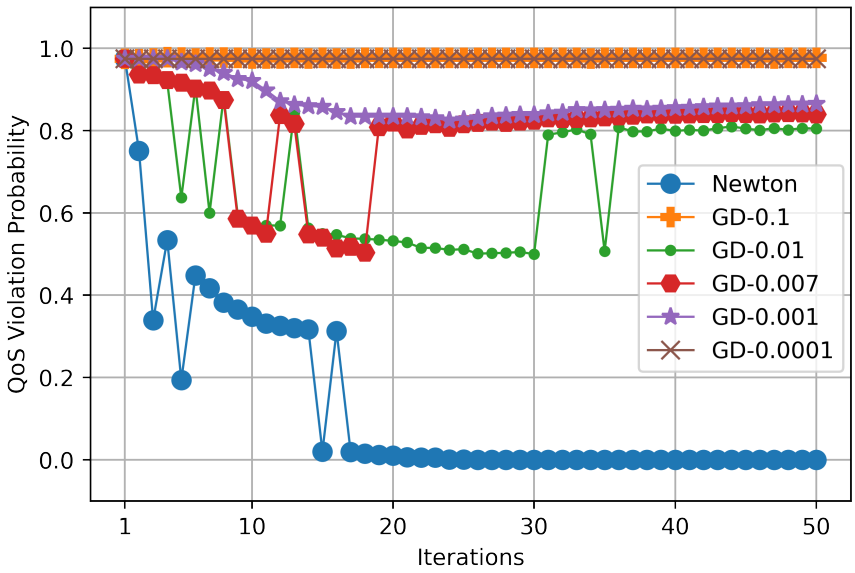

Figure 3.9: The comparison of the convergence rate of different correction processes before training. GD refers to gradient-descent and the number is the step-size.

Fig. 3.9 shows the dynamics of the projection function before the training. As I can see, Newton method is the only correction process that can achieve zero violation probability. Among different step-sizes for GD, I can observe that step size 0.01 and 0.007 achieve lower violation probability than others. I can see that having very large and small step-sizes (0.1 and 0.0001) results in no progress. Thus, the step-size should be chosen with experiments to find the right range that helps the convergence of the projection function. Based on Fig. 3.9, I see that a step-size between 0.001 and 0.01 will work for this dataset. After trying over more datapoints, I chose the step-size of

0.007 to enjoy fast convergence and avoid any diverging issues.

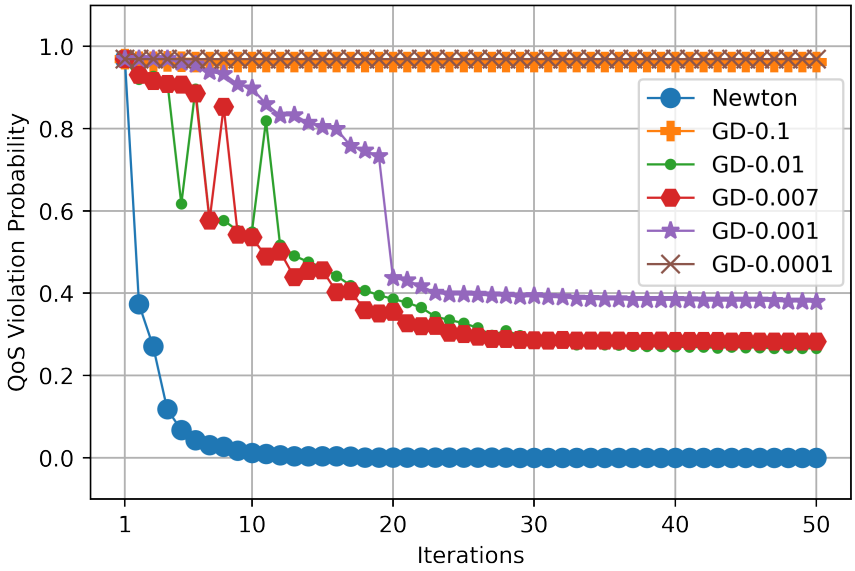

Figure 3.10: The comparison of the convergence rate of different correction processes after training.

Fig. 3.10 shows the dynamics of the projection function of DEPNet with different correction processes after the training is completed. I can see that the gradient-descent with step-size 0.01, 0.007, and 0.001, achieves better results than before (without training). I can see that step-size of 0.01 and 0.007 achieve to near 0.25 violation probability, which is a considerable improvement over the 0.8 violation I observed before training. The Newton method still is the winner of the game by achieving zero violation probability. Compared to Fig. 3.9, I can see that the convergence of Newton method is improved after training and it converges after almost 10 iterations, which took about 20 iterations to happen before the training. Moreover, considering the dynamics of the Newton method, I can observe that after training, I don't have any oscillations before the convergence. This implies that by training, the backbone neural network learns to output points that are already very close to

the feasible set, and just taking a few steps of Newton method will land them on the feasible set.

3.8 Summary

In this chapter, a novel neural network-based solver is proposed to address the network sum-rate maximization using power control with QoS and power budget constraints. To achieve zero constraint violation probability, a differentiable projection framework is utilized, which uses a projection function to project the output of the backbone neural network to the feasible set of the problem. The projection function is defined implicitly using convex optimization and explicitly using an iterative process. The resulting DIPNet and DEPNet are tested against optimization-based and neural-based benchmarks. Numerical experiments confirmed zero violation probability of the output of the proposed models while outperforming the neural-based benchmark in terms of sum-rate and GP in terms of the computation time. In realistic setups, however, GP outperformed the proposed models in terms of network sum-rate. The main reason behind this is the tendency of the proposed projection functions to find solutions at the boundary of the feasible set. The performance can be improved by designing other projection functions that can search within the feasible set to improve the performance.

Chapter 4

Optimized User Association with QoS Guarantees

User association and power control are two co-related resource management problems in wireless communication. Thus, solving one without considering the other will result in a sub-optimal solution. The sole power control was addressed in the previous chapter with the assumption that user association was already performed. In this chapter, I consider the problem of joint user association and power control with QoS constraints which is a non-convex mixed integer non-linear programming (MINLP) problem.

The considered joint problem is decomposed into power control and user association sub-problems. The power control sub-problem is solved using the discussed method in the previous chapter. For the user association sub-problem, a DNN-based solver is developed which uses the discussed projection methods in chapter 3 to handle the constraints. More specifically, two projection methods are designed for this problem. The first one uses convex optimization to define the projection implicitly. The second one uses an iterative algorithm to project the output of the neural network to the Birkhof polytope [58] using the so-called Sinkhorn normalization [26].

To summarize, the contribution of this chapter are as follows:

- I propose two DL-based solvers for the user association sub-problem in a multi-user interference channel with QoS constraints. Two DNNs called Deep Implicit Projection User Network (DIUNet) and Deep Explicit Projection User Network (DEUNet) are designed to provide feasible solutions for user association problem. They use convex optimization and iterative processes, respectively, to realize the projections.

- The user association networks are trained in an unsupervised manner and compared with both DNN-based and optimization-based benchmarks. For the former, an FCNN with a softmax layer is used as in [15], and for the latter, a mixed-integer programming solver from MOSEK package [16] is utilized. The sum-rate, constraint violation probability, and online testing time are used as the evaluation metrics.

- The user association DNNs are combined with the power control DNNs (chapter 3) to provide an alternating optimization-based solution for the joint power control and user association problem, called JUPNet. For the comparison, the mixed-integer solver from MOSEK package and Geometric Program (GP) is used to solve user association and power control sub-problems, respectively, in an alternating manner. The resulting optimization-based solver is used as the benchmark.

4.1 Problem formulation

I consider the problem of joint power control and user association in a downlink wireless network with B single-antenna base stations (BSs) and U users. Each BS can serve Q users at maximum in Q orthogonal frequency channels and BSs are operating

over the same frequency spectrum. The overall bandwidth is equally distributed amongst the users of the same BS. While the orthogonal channel allocation at each BS prevents the intra-cell interference, the inter-cell interference on each channel is experienced from the neighboring BSs. Thus, the achievable rate of the user u if being associated to the channel q of BS b, i.e. $R_{b,q,u}$, can be modeled as follows:

$$R_{b,q,u}(\mathbf{P},\mathbf{H}) = W\log_2\left(1+\gamma_{b,q,u}(\mathbf{P},\mathbf{H})\right), \quad \gamma_{b,q,u}(\mathbf{P},\mathbf{H}) = \frac{|H_{b,q,u}|^2 P_{b,q}}{\sum_{\hat{b}=1,\hat{b}\neq b}^{B}|H_{\hat{b},q,u}|^2 P_{\hat{b},q}+\sigma^2}$$

where $H_{b,q,u}$ denotes the channel from channel q of BS b to user u, $P_{b,q}$ denotes the transmit power of BS b over channel q, W denotes the amount of bandwidth allocated to the user, and $\gamma_{b,q,u}$ denotes the Signal-to-Interference-to-Noise ratio (SINR) experienced by user u on channel q of BS b. $\mathbf{P} \in \mathbb{R}^{B\times Q}$ and $\mathbf{H} \in \mathbb{R}^{B\times Q\times U}$ denote the matrix and tensor containing all values of transmit powers and complex channel gains composed of distance-based path-loss, shadowing, and fading, respectively. I assume that the perfect CSI is available at the BS side. Also, σ refers to the thermal noise power experienced at the users' receivers, which is the same for all the users. I also denote the set of all users, BSs, and channels as $\mathcal{U} = 1,\ldots,U$, $\mathcal{B} = 1,\ldots,B$, and $\mathcal{Q} = 1,\ldots,Q$, respectively. The association decisions are formalized as follows:

$$A_{b,q,u} = \begin{cases} 1 & \text{if user } u \text{ if associated to channel } q \text{ of BS } b \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Moreover, the tensor $\mathbf{A} \in \mathbb{R}^{B \times Q \times U}$ denotes the collection of all association decisions. The network sum-rate maximization problem with QoS constraints can then be formulated as follows:

$$
\begin{aligned}
\underset{\mathbf{P},\mathbf{A}}{\text{maximize}} \quad & R(\mathbf{P},\mathbf{A},\mathbf{H}) = \sum_{u=1}^{U} R_u(\mathbf{P},\mathbf{A},\mathbf{H}) = \sum_{u=1}^{U}\sum_{q=1}^{Q}\sum_{b=1}^{B} A_{b,q,u} R_{b,q,u}(\mathbf{P},\mathbf{H}) \\
\text{subject to} \quad & P_{b,q} \geq 0, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q} \\
& \sum_{q=1}^{Q} P_{b,q} \leq P_{\max}, \quad \forall b \in \mathcal{B} \\
& A_{b,q,u} \in \{0,1\}, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q}, \forall u \in \mathcal{U} \\
& \sum_{u}^{U} A_{b,q,u} = 1, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q} \\
& \sum_{q}^{Q}\sum_{b}^{B} A_{b,q,u} = 1, \quad \forall u \in \mathcal{U} \\
& R_u(\mathbf{P},\mathbf{A},\mathbf{H}) \geq \alpha_u, \quad \forall u \in \mathcal{U}
\end{aligned}
\tag{4.2}
$$

where the first constraint ensures the non-negative power allocations, and the second one refers to the transmit power budget of each BS which is considered to be the same without the loss of generality. The third constraint ensures that the association variables are binary. The fourth and fifth constraints, respectively, ensure that each channel of each BS gets exactly one user, and each user is associated with only one BS and one channel of it. Finally, the sixth constraint ensures the minimum rate requirement of each user, denoted by α_u. This optimization problem has a non-convex objective and constraint set and involves both continuous ($\mathbf{P}$) and discrete ($\mathbf{A}$) variables, which makes it MINLP as mentioned before.

To solve this problem, I first decompose (4.2) into two sub-problems to deal with

power allocation and user association decisions separately in an alternating manner. Given a certain $\hat{\mathbf{A}}$ denote a user association decision, the power control sub-problem can be formalized as follows:

$$\begin{aligned} \underset{\mathbf{P}}{\text{maximize}} \quad & R(\mathbf{P}, \hat{\mathbf{A}}, \mathbf{H}) \\ \text{subject to} \quad & P_{b,q} \geq 0, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q} \\ & \sum_{q=1}^{Q} P_{b,q} \leq P_{\max}, \quad \forall b \in \mathcal{B} \\ & R_u(\mathbf{P}, \hat{\mathbf{A}}, \mathbf{H}) \geq \alpha_u, \quad \forall u \in \mathcal{U} \end{aligned} \tag{4.3}$$

where the objective is to find the power allocation $\mathbf{P}$ that maximizes the network sum-rate and meets the QoS constraints. This problem is derived from (4.2) by assuming that the user association has already happened. The DNN-based solvers for the aforementioned problem are already discussed in chapter 3.

The user association sub-problem, given the power allocations $\hat{\mathbf{P}}$, is given as follows:

$$\begin{aligned} \underset{\mathbf{A}}{\text{maximize}} \quad & R(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) \\ \text{subject to} \quad & A_{b,q,u} \in \{0, 1\}, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q}, \forall u \in \mathcal{U} \\ & \sum_{u}^{U} A_{b,q,u} = 1, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q} \\ & \sum_{q}^{Q} \sum_{b}^{B} A_{b,q,u} = 1, \quad \forall u \in \mathcal{U} \\ & R_u(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) \geq \alpha_u, \quad \forall u \in \mathcal{U} \end{aligned} \tag{4.4}$$

In what follows, I develop a DNN-based solution for (4.4). For that, I first ensure the third constraint through a transformation to an equivalent problem. The resulting

problem is then relaxed to its linear program (LP) counterpart. Due to the special structure of the constraint set of (4.4), this relaxation will not hurt the optimality or feasibility of the solution [59]. The resulting LP is then solved using a DNN that utilizes a projection method for ensuring the feasibility of its output.

4.2 Problem Transformation and its Equivalence

Due to the reliance on gradient-based methods for training DNNs, the continuity of the variables involved in DNN training is a necessity. Thus, (4.4) with discrete variables has to undergo some transformations to become suitable for a DNN-based solver. The first step is to ensure the minimum rate constraint of (4.4). This is done by formulation of an equivalent problem, whose optimal solution is the optimal solution of (4.4). The resulting problem has only the first three constraints of (4.4), and takes the form of the linear sum assignment problem (LSAP) whose feasible set is the set of permutation matrices [59]. This LSAP can then be converted to its linear program counterpart by letting the association variables take continuous values between zero and one. As shown in [59], solving the resulting LP gives a binary-valued solution, which is the solution of (4.4).

4.2.1 Problem Transformation

Let a new variable $U_{b,q,u}$ is defined as follows:

$$U_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) = \begin{cases} R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) & \text{if } R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) \geq \alpha_u \\ -S(\hat{\mathbf{P}}, \mathbf{H}) & \text{otherwise} \end{cases} \tag{4.5}$$

where $S(\hat{\mathbf{P}}, \mathbf{H}) = \sum_{u=1}^{U} \sum_{q=1}^{Q} \sum_{b=1}^{B} R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H})$. The considered problem can then be transformed into an LSAP form as shown below:

$$\begin{aligned} \underset{\mathbf{A}}{\text{maximize}} \quad & U(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) = \sum_{u=1}^{U} U_u(\mathbf{P}, \mathbf{A}, \mathbf{H}) = \sum_{u=1}^{U} \sum_{q=1}^{Q} \sum_{b=1}^{B} A_{b,q,u} U_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) \\ \text{subject to} \quad & A_{b,q,u} \in \{0,1\}, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q}, \forall u \in \mathcal{U} \\ & \sum_{u}^{U} A_{b,q,u} = 1, \quad \forall b \in \mathcal{B}, \forall q \in \mathcal{Q} \\ & \sum_{q}^{Q} \sum_{b}^{B} A_{b,q,u} = 1, \quad \forall u \in \mathcal{U} \end{aligned} \tag{4.6}$$

The main difference between (4.4) and (4.6) is that the minimum rate constraint is removed from the feasible set of (4.6) and incorporated to the objective of (4.6).

4.2.2 Proof of Equivalence

In the following, I prove that (4.4) and (4.6) are equivalent mathematically.

Theorem 1: Let $\hat{\mathbf{A}}$ be the solution of (4.6) for a given power allocation $\hat{\mathbf{P}}$ and channel $\mathbf{H}$. (4.6) and (4.4) are equivalent if and only if:

- $\hat{\mathbf{A}}$ is the optimal solution of (4.4) for $\hat{\mathbf{P}}$ and $\mathbf{H}$.
- $U(\hat{\mathbf{P}}, \hat{\mathbf{A}}, \mathbf{H}) = R(\hat{\mathbf{P}}, \hat{\mathbf{A}}, \mathbf{H})$

Proof: Let $\mathcal{C}^U$ and $\mathcal{C}^R$ denote the feasible set of (4.6) and (4.4), respectively. Since (4.6) has all the constraints of (4.4) except for the minimum rate, I have:

$$\mathcal{C}^R \subseteq \mathcal{C}^U \tag{4.7}$$

Moreover, for an arbitrary $\mathbf{A} \in \mathcal{C}^U$, I have:

$$\sum_{u=1}^{U}\sum_{q=1}^{Q}\sum_{b=1}^{B} A_{b,q,u} = \sum_{u=1}^{U}(\sum_{q=1}^{Q}\sum_{b=1}^{B} A_{b,q,u}) = \sum_{u=1}^{U} 1 = U \tag{4.8}$$

Where the second term is true due to the third constraint of (4.6), i.e. $\sum_{q=1}^{Q}\sum_{b=1}^{B} A_{b,q,u} = 1$. Since $\mathbf{A}$ is a binary-valued matrix, (4.8) implies that $\mathbf{A}$ has U ones and U(BQ - 1) zeros. Since for the case of one BS and one channel, there is no user association problem, BQ is greater than one. This means that $\mathbf{A}$ always has zeros. Therefore:

$$R(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) = \sum_{u=1}^{U}\sum_{q=1}^{Q}\sum_{b=1}^{B} A_{b,q,u} R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) < \sum_{u=1}^{U}\sum_{q=1}^{Q}\sum_{b=1}^{B} R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) = S(\hat{\mathbf{P}}, \mathbf{H}) \tag{4.9}$$

where the inequality is strict since $\mathbf{A}$ contains zeros and rates are all positive.

Now, I prove that the solution of (4.6) belongs to $\mathcal{C}^R$. Consider two arbitrary points $\mathbf{A}^1 \in \mathcal{C}^R$ and $\mathbf{A}^2 \in \mathcal{C}^U/\mathcal{C}^R$. Since $\mathbf{A}^2$ belongs to $\mathcal{C}^U$ but not $\mathcal{C}^R$, there is at least a user who does not receive his minimum rate with $\mathbf{A}^2$ association. If I denote this user by $\hat{u}$ and its associated BS and channel by $\hat{b}$ and $\hat{q}$, I have:

$$R_{\hat{u}}(\hat{\mathbf{P}}, \mathbf{A}^2, \mathbf{H}) = \sum_{q=1}^{Q}\sum_{b=1}^{B} A_{b,q,\hat{u}} R_{b,q,\hat{u}}(\mathbf{P}, \mathbf{H}) = R_{\hat{b},\hat{q},\hat{u}}(\mathbf{P}, \mathbf{H}) < \alpha_{\hat{u}} \tag{4.10}$$

Thus, following (4.5), I have:

$$U_{\hat{u}}(\hat{\mathbf{P}}, \mathbf{A}^2, \mathbf{H}) = U_{\hat{b},\hat{q},\hat{u}}(\mathbf{P}, \mathbf{H}) = -S(\hat{\mathbf{P}}, \mathbf{H}) \tag{4.11}$$

Moreover, based on (4.5), I know:

$$U_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) \leq R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) \longrightarrow U_u(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) \leq R_u(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) \tag{4.12}$$

where the equality occurs when the user association $\mathbf{A}$ satisfies the minimum rate requirement of user u, i.e. α_u. Therefore:

$$\begin{aligned} U(\hat{\mathbf{P}}, \mathbf{A}^2, \mathbf{H}) &= \textstyle\sum_{u=1, u\neq\hat{u}}^{U} U_u(\hat{\mathbf{P}}, \mathbf{A}^2, \mathbf{H}) - S(\hat{\mathbf{P}}, \mathbf{H}) \\ &\leq \textstyle\sum_{u=1, u\neq\hat{u}}^{U} R_u(\hat{\mathbf{P}}, \mathbf{A}^2, \mathbf{H}) - S(\hat{\mathbf{P}}, \mathbf{H}) \\ &< R(\hat{\mathbf{P}}, \mathbf{A}^2, \mathbf{H}) - S(\hat{\mathbf{P}}, \mathbf{H}) < 0 \\ U(\hat{\mathbf{P}}, \mathbf{A}^2, \mathbf{H}) &< 0 \end{aligned} \tag{4.13}$$

This means that for every user association $\mathbf{A}^2$ that doesn't satisfy the minimum rate constraint for all the users, $U(\hat{\mathbf{P}}, \mathbf{A}^2, \mathbf{H})$ becomes negative. On the other hand, based on (4.12), if a user association, like $\mathbf{A}^1$, satisfies the minimum rate requirement of all the users, $U(\hat{\mathbf{P}}, \mathbf{A}^1, \mathbf{H})$ becomes equal to network sum-rate; thus positive. In other words:

$$U(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) \begin{cases} < 0 \text{ if } \mathbf{A} \in \mathcal{C}^U / \mathcal{C}^R \\ = R(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) > 0 \text{ if } \mathbf{A} \in \mathcal{C}^R \end{cases} \tag{4.14}$$

Thus, the optimal solution of (4.6) should belong to $\mathcal{C}^R$, i.e. $\hat{\mathbf{A}} \in \mathcal{C}^R$. Since the objective of (4.6) is equal to (4.4) when $\mathbf{A} \in C^R$, $\hat{\mathbf{A}}$ maximizes (4.4) as well. This means that $\hat{\mathbf{A}}$ is the optimal solution of (4.4) and based on (4.14), $U(\hat{\mathbf{P}}, \hat{\mathbf{A}}, \mathbf{H}) = R(\hat{\mathbf{P}}, \hat{\mathbf{A}}, \mathbf{H})$. End of proof.

Theorem 1 shows that I can solve (4.6) instead of (4.4) and get the same optimal user association.

4.2.3 LP Relaxation and Equivalence

Now, I show that solving the continuous relaxation of (4.6) results in the optimal solution of (4.6). This is an important step since it allows us to use a DNN as the solver. To perform this transition, I first rewrite (4.6) by representing the user association decision as a matrix and vector.

Let us denote the matrix version of user association, decision as $\mathbf{A}$, which can be derived as follows:

$$A_{k,u} = A_{b,q,u} : \begin{cases} b = (\text{reminder of } k \text{ w.r.t. } B) + 1 \\ q = \lfloor \frac{k-1}{B} \rfloor + 1 \end{cases} \tag{4.15}$$

Now, I can see that k goes from 1 to BQ. Since BQ is equal to U, I denote the set of all values of k as $\mathcal{K} = \{1, \ldots, U\}$.

I can apply the same transformation as (4.15) to $U_{b,q,u}$ as well. Thus, I can rewrite (4.6) w.r.t. $\mathbf{A}$ as:

$$\begin{aligned} \underset{\mathbf{A}}{\text{maximize}} \quad & U(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) = \sum_{u=1}^{U} U_u(\mathbf{P}, \mathbf{A}, \mathbf{H}) = \sum_{u=1}^{U} \sum_{k=1}^{U} A_{k,u} U_{k,u}(\hat{\mathbf{P}}, \mathbf{H}) \\ \text{subject to} \quad & A_{k,u} \in \{0, 1\}, \quad \forall k \in \mathcal{K}, \forall u \in \mathcal{U} \\ & \sum_{u=1}^{U} A_{k,u} = 1, \quad \forall k \in \mathcal{K} \\ & \sum_{k=1}^{U} A_{k,u} = 1, \quad \forall u \in \mathcal{U} \end{aligned} \tag{4.16}$$

This equation takes the form of LSAP [59]. I can relax the first constraint, letting $A_{k,u}$ to be continuous and belong to the interval between zero and one, i.e. $A_{k,u} \in [0, 1]$.

By doing this, (4.16) becomes a linear program (LP):

$$\begin{aligned} \underset{\mathbf{A}}{\text{maximize}} \quad & U(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) \\ \text{subject to} \quad & A_{k,u} \in [0,1], \quad \forall k \in \mathcal{K}, \forall u \in \mathcal{U} \\ & \sum_{u=1}^{U} A_{k,u} = 1, \quad \forall k \in \mathcal{K} \\ & \sum_{k=1}^{U} A_{k,u} = 1, \quad \forall u \in \mathcal{U} \end{aligned} \tag{4.17}$$

The feasible set of this LP is the set of all matrices with values between 0 and 1 whose rows and columns sum to 1. This set is denoted as either Birkhoff polytope or the set of doubly stochastic matrices [59,60]. Formally, I can write this set as follows:

$$\mathcal{C}^B = \{\mathbf{A} | \mathbf{A} \in \mathbb{R}^{U \times U}, \quad \mathbf{A}.\mathbf{1} = \mathbf{A}^T.\mathbf{1} = \mathbf{1}, \quad A_{k,u} \in [0,1]\} \tag{4.18}$$

where $\mathbf{1} \in \mathbb{R}^U$ is the vector of all ones. As mentioned in [59], solving the resulting LP (4.17) results in an integral solution, i.e. a matrix with binary values, that is also the optimal solution of (4.16). In other words, (4.17) is equivalent to (4.16). Therefore, I show that instead of solving (4.4), I can solve (4.17).

4.3 Neural network solutions

In this section, I try to develop a DNN-based solution for (4.17). The procedure to transform a variable optimization problem into a learning problem is described in detail in Chapters 2 and 3. Thus, I only demonstrate the conclusion of this process.

The main goal is to use a DNN to approximate the functionality between $\mathbf{H}$ and $\mathbf{A}$ that implicitly defined in (4.17) using unsupervised learning. For ease of notation,

I first write the vectorized form of (4.17). Let $\mathbf{a} \in \mathbb{R}^{U^2}$ be the vector derived from stacking the columns of $\mathbf{A} \in \mathbb{R}^{U \times U}$ on top of each other, i.e.

$$a_{(u-1)U+k} = A_{k,u} \tag{4.19}$$

Using (4.19), I can rewrite (4.17) as follows:

$$\begin{aligned} \underset{\mathbf{a}}{\text{maximize}} \quad & U(\hat{\mathbf{P}}, \mathbf{a}, \mathbf{H}) \\ \text{subject to} \quad & \mathbf{a} \in [0,1]^{U^2} \\ & \mathbf{D}\mathbf{a} = \mathbf{1} \\ & \mathbf{C}\mathbf{a} = \mathbf{1} \end{aligned} \tag{4.20}$$

Where $\mathbf{D}, \mathbf{C} \in \mathbb{R}^{U \times U^2}$ are defined as follows [59]:

$$C_{i,j} = \begin{cases} 1 \text{ if } i = \lfloor \frac{j-1}{U} \rfloor + 1 \\ 0 \text{ otherwise} \end{cases}, \quad D_{i,j} = \begin{cases} 1 \text{ if } i \equiv j \text{ (mod U)} \\ 0 \text{ otherwise} \end{cases} \tag{4.21}$$

Let $\mathbf{U}(\hat{\mathbf{P}}, \mathbf{H}) \in \mathbb{R}^{U \times U}$ be a matrix containing all possible values of $U_{k,u}(\hat{\mathbf{P}}, \mathbf{H})$ for a given datapoint $\mathbf{H}$. Considering (4.20), I have the following:

$$\mathbf{A} = f(\hat{\mathbf{P}}, \mathbf{H}) = g(\mathbf{U}(\hat{\mathbf{P}}, \mathbf{H})) \tag{4.22}$$

Where $f : \mathbb{R}^{B \times Q \times U} \longrightarrow \mathbb{R}^{U \times U}$ is the underlying functionality of (4.20). Since I already know functionality from $\mathbf{H}$ to $\mathbf{U}$ (4.5), I try to approximate $g : \mathbb{R}^{U \times U} \longrightarrow \mathbb{R}^{U \times U}$ using a DNN.

let $\mathcal{N}_A$ denote the neural network who gets $\mathbf{U}$ and outputs the user association

vector $\mathbf{a}$. Similar to chapter 3, I have:

$$\mathbf{a} = \mathcal{N}_a(\mathbf{U}, \mathbf{w}_a) = \text{Proj}(\mathbf{r}), \quad \mathbf{r} = \mathcal{N}_r(\mathbf{H}, \mathbf{w}_r) \tag{4.23}$$

Where $\mathbf{r} \in \mathbb{R}^{U^2}$ is the output of the previous layer and $\mathcal{N}_r$ is called the backbone neural network. Proj : $\mathbb{R}^{U^2} \longrightarrow \mathbb{R}^{U^2}$. Similar to chapter 3, the projection function needs to be differentiable to enable the end-to-end training of $\mathcal{N}_a$.

In the following, two methods to realize the projection function is explored. The first method uses convex optimization to implicitly define the projection function. The second one, uses the Sinkhorn algorithm [26, 61], to explicitly implement the projection function. A detailed description of these methods are provided in the following sections.

4.3.1 Implicit Projection

In this section, I use mathematical optimization to define the projection function. Similar to chapter 3, differentiable convex optimization layer (DCO) [21] is used for the implementation. Since the feasible set of (4.20) is linear; thus convex, I define the projection function as follows:

$$\begin{aligned} \underset{\mathbf{a}}{\text{maximize}} \quad & \mathbf{r}^T\mathbf{a} \\ \text{subject to} \quad & \mathbf{a} \in [0,1]^{U^2} \\ & \mathbf{D}\mathbf{a} = \mathbf{1} \\ & \mathbf{C}\mathbf{a} = \mathbf{1} \end{aligned} \tag{4.24}$$

Where $\mathbf{r} \in \mathbb{R}^{U^2}$ is the output of the previous layer. For this projection, the dimension of $\mathbf{r}$ is equal to the dimension of the user association vector. As can be seen, (4.24) implicitly define the projection of a vector $\mathbf{r}$ to the feasible set of (4.20). Moreover, if I use (4.19) to transform $\mathbf{a}$ to a matrix $\mathbf{A}$, I can see that this matrix belongs to the Birkhoff polytope ($\mathcal{C}^B$). Therefore, (4.24) defines a projection unto the Birkhoff polytope.

Another interesting property of (4.20) is that if I concatenate $\mathbf{C}$ and $\mathbf{D}$ in a matrix called $\mathbf{T}$, $\mathbf{T}$ will be a unimodular matrix [59]. As a result, the solution of (4.24) will be integral, i.e. only containing zeros and ones [59]. Thus, the output of this projection layer will be a valid association matrix ($\mathbf{A}$).

I call the DNN with this layer Deep Implicit projection User association Network (DIUNet). Since the output of DIUNet always belongs to the Birkhoff polytope, the loss function used for training DIUNet takes the following form:

$$l(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) = -U(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) \tag{4.25}$$

4.3.2 Explicit Projection

In this section, I define a projection function explicitly. Considering (4.17), the main goal of the projection function is to project the output of $\mathcal{N}_r$ to the Birkhoff polytope. One way to do this is to use the well-known Sinkhorn normalization. Sinkhorn normalization (operator) [26,61] is an iterative process that starts from a given positive-valued matrix and by iteratively normalizing its rows and columns outputs a matrix that belongs to the Birkhoff polytope. As mentioned in [61], Sinkhorn operator is also

differentiable, making it a good fit for the projection function. The detailed description of this operator can be found in [26]. In the following, I give a brief explanation of this algorithm.

Let $\mathbf{R} \in \mathbb{R}^{U \times U}$ denote a U dimensional square matrix. The Sinkhorn operator, denoted by $S(\mathbf{R})$ is defined as follows:

$$\begin{aligned} S^0(\mathbf{R}) &= \exp(\mathbf{R}), \\ S^l(\mathbf{R}) &= \mathcal{T}^c(\mathcal{T}^r(S^{l-1}(\mathbf{R}))), \\ S(\mathbf{R}) &= \lim_{l \to \infty} S^l(\mathbf{R}). \end{aligned} \tag{4.26}$$

The first line (S^0) is the initialization of this algorithm, proposed in [26]. $\mathcal{T}^c$ and $\mathcal{T}^r$ are operators that normalize the columns and rows of the input matrix, respectively. These operators are defined below:

$$\mathcal{T}^c(\mathbf{R}) = \mathbf{R} \oslash (\mathbf{R}\mathbf{1}\mathbf{1}^T), \quad \mathcal{T}^r(\mathbf{R}) = \mathbf{R} \oslash (\mathbf{1}\mathbf{1}^T\mathbf{R}) \tag{4.27}$$

Where $\mathbf{1} \in \mathbb{R}^{U \times 1}$ is a U dimensional column vector of ones and $\oslash$ is the element-wise division operation. As proved in [26], I have:

$$\mathbf{A} = \lim_{t \to 0} S(\frac{\mathbf{R}}{t}) \tag{4.28}$$

Where $\mathbf{A}$ is an association (permutation) matrix, i.e. a binary-valued doubly-stochastic matrix. In other words, when t gets close to zero, the output of Sinkhorn operator becomes close to a vertex of Birkhoff polytope. This is an interesting theoretical property of this process.

In practice, however, I cannot realize the infinite iterations of (4.26) or zero limits of (4.28). The latter is addressed by using the truncated version of Sinkhorn normalization [61] and the former is realized by choosing a small t. The truncated version performs a finite number of iterations, denoted by S^L. As mentioned in [61], the truncated version of Sinkhorn normalization (S^L) is differentiable; thus can be incorporated with the backbone neural network, and enjoy the end-to-end training. The projection function, therefore, can be defined as follows:

$$\mathbf{A} = \text{Proj}(\mathbf{R}) = S^L(\mathbf{R}) \tag{4.29}$$

Where $\mathbf{R}$ is the matrix version of $\mathbf{r}$, and can be derived via (4.19). $\mathbf{A}$ is a doubly stochastic matrix. Since this DNN uses an explicit projection function, I call it Deep Explicit projection User association Network (DEUNet). The following loss function is used for training:

$$l(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) = -U(\hat{\mathbf{P}}, \mathbf{A}, \mathbf{H}) + \lambda(||\mathbf{1} - \mathbf{A}\mathbf{1}\mathbf{1}^T||_2^2 + ||\mathbf{1} - \mathbf{1}\mathbf{1}^T\mathbf{A}||_2^2) \tag{4.30}$$

Since finite iterations of Sinkhorn operator are used for the projection purpose, there is no guarantee that the output lies inside the Birkhoff polytope. Thus the second term is added to the loss to penalize the output if it does not belong to the Birkhoff polytope. Moreover, λ is a hyperparameter that controls the relative importance of minimizing the second term versus the first term.

4.3.3 Neural Network Architecture

Since the main concern of this work is the design of the projection functions, I use a fully connected neural network (FCNN) as the backbone ($\mathcal{N}_r$). The architecture is composed of fully connected layers with ReLU activation function for all the layers. The input to the neural network is the vectorized version of $\hat{\mathbf{U}}(\hat{\mathbf{P}}, \mathbf{H})$. $\hat{\mathbf{U}}(\hat{\mathbf{P}}, \mathbf{H})$ is a tensor containing all possible rates $R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H})$ in which every entry that is smaller than α_u is set to zero. In other words, the input if similar to $\mathbf{U}(\hat{\mathbf{P}}, \mathbf{H})$, but I replaced the $-S(\hat{\mathbf{P}}, \mathbf{H})$ with zero. The input is further normalized across the datapoints to fasten the training of DNN.

The dimension of the output layer (d) is set to U^2 for both of the projection methods. Moreover, I added a sigmoid non-linearity to $\mathcal{N}_r$ for the implicit projection. The reason behind this is that it helps the optimization problem of the implicit projection to be solved faster.

During the training, the output of the DIUNet and DEUNet ($\mathbf{A}$) is directly fed to the loss function. At the inference, however, I pick the highest value in each column of $\mathbf{A}$ and replace it with one. The other values will be replaced with zero to get a clean binary matrix. This may result in BS's quota violation, i.e. a BS may get more than Q users. This metric is used as a comparison metric in the experiments.

4.3.4 Ensuring the QoS by masking

I use a trick to prevent the DNNs to output an association that violates the QoS constraints in (4.4). The main idea behind this trick is to change the output of the backbone neural network ($\mathbf{r}$) in a way that prevents the projection function to output an association matrix that violates the minimum rate constraint. At the same time,

this change should not affect the acceptable user association decisions. I call this trick *QoS-Masking*.

Given a datapoint $\mathbf{H}$, I can calculate all possible rates ($R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H})$) that users can experience under every possible assignment. As a result, I already know which rates are smaller than the minimum rate requirement of a user u (α_u). Thus, similar to (4.5), I can define the following quantity:

$$I_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) = \begin{cases} 0 & \text{if } R_{b,q,u}(\hat{\mathbf{P}}, \mathbf{H}) \geq \alpha_u \\ -\infty & \text{otherwise} \end{cases} \tag{4.31}$$

Now by applying (4.15) and (4.19), I can reshape $\mathbf{I} \in \mathbb{R}^{B \times Q \times U}$ to $\mathbf{i} \in \mathbb{R}^{U^2}$. In this way, $\mathbf{i}$ has the same dimension as $\mathbf{r}$ and can be added to it:

$$\hat{\mathbf{r}} = \mathbf{r} + \mathbf{i} \tag{4.32}$$

Where $\hat{\mathbf{r}}$ only differs from $\mathbf{r}$ in the elements that correspond to a rate should not be selected. In the following, I show that if I input $\hat{\mathbf{r}}$ to both of the projection functions, introduced before, the resulting association matrix won't violate the QoS constraint.

For the implicit projection, I can see that the goal of the objective function of (4.24) is to maximize $\hat{\mathbf{r}}^T\mathbf{a}$. Consider an element of $\hat{\mathbf{r}}$ like r_j which is $-\infty$. Since all the elements of $\mathbf{a}$ are between zero and one, the corresponding element of $\mathbf{a}$ to r_j (a_j) has to be zero to prevent the objective function of (4.24) to become $-\infty$. Thus, the entries of the resulting association matrix will be zero for wrong decisions.

For the explicit projection, first, I reshape $\hat{\mathbf{r}}$ to a matrix $\hat{\mathbf{R}}$ following (4.19). Considering the initialization of the Sinkhorn normalization the output of $S^0 = exp(\hat{\mathbf{R}})$

will be zero for the elements of $\hat{\mathbf{R}}$ that are $-\infty$. Moreover, since every iteration of Sinkhorn normalization only involves dividing the elements of the input matrix by some values, if an element of the matrix becomes zero it will stay as such after the subsequent iterations. Thus, the resulting association matrix will have zero values for the incorrect decisions. The same is true if I apply a column-wise softmax. Thus, by using this trick, I will prevent the DNNs to output an association that does not guarantee the QoS requirement of the users.

4.4 Experimental Set-up and Benchmarks

In this section, the dataset specifications, system set-up, and relevant benchmarks to evaluate the performance of DIUNet and DEUNet are presented.

4.4.1 DataSet Generation

For the experimental evaluation, I created six datasets following Gaussian and Path-loss models, as described in chapter 3. To be more specific, in Gaussian datasets, the channel coefficients follow the symmetric interference channel model with independent and identically distributed (i.i.d.) Rayleigh fading, i.e. $h_{b,q,u} \sim \mathcal{CN}(0,1)$. The maximum power is set to 1mW and the thermal noise power is assumed to be $0.01\mu W$ [51], and the transmission bandwidth is 5 MHz. For Path-loss, channel coefficients that are composed of large-scale path-loss, shadowing, and fading. The dataset generation has the following steps. First, the locations of the BSs and users are determined by random sampling from a 500 m $\times$500 m area. I consider that the distance between the two closest BSs, BSs and user, and between closest users, should be at least 100 m, 5 m, and 2 m, respectively. The carrier frequency is 2.4 GHz, the

transmission bandwidth is 5 MHz, and the noise spectral density is set to -169 dBm. The remaining details of the channel model can be found in [52].

In total, six datasets were generated. For Gaussian, the number of BSs in all the datasets is set to 2 ($B = 2$) and the minimum rate is set to 2.5Mbps. Different configurations are specified with dataset id. ID 1 - 3 refers to dataset with 2, 4, and 6 users, respectively, where $Q = U/B$. For Path-loss, the number of BSs is set to 4 and the minimum rate is set to 2.5Mbps, and the number of users ranges form 4 to 12. For each configuration, 100,000 data samples are generated and divided to train, validation, and test with 0.9, 0.05, 0.05 ratios, respectively.

For each configuration, I apply the same feasibility check as chapter 3 to filter out the datapoints for which there is no combination of user association and transmit power that can ensure the minimum rate constraint. Moreover, I use the MinPower algorithm introduced in section 3.6.2 to generate the minimum transmit power required to fulfill the QoS constraint. This transmit power is used as $\hat{\mathbf{P}}$ in (4.17).

4.4.2 Benchmarks

Three benchmarks are used for the evaluation purpose: UNet, similar to the model proposed in [15], an Integer linear program solver (ILP) (from MOSEK's package [16]), and a linear program solver (LP) (from ECOS's package [56]) as the neural network and optimization-based benchmarks, respectively. The details of these benchmarks are presented below:

- **UNet:** an FCNN that has the same backbone architecture as DIUNet and DEUNet, which was introduced in [15], is used as the neural network solver. Instead of using the introduced projections, UNet applies a row-wise softmax

on the rows of the output matrix; ensuring that the sum of the rows of the output matrix is one. Similar to DEUNet's loss function, a term is added to its loss function to penalize its output of it in case its columns do not sum to one.

- **ILP** and **LP:** are used to solve the user association problem in its original form (4.4) and transformed version (4.17), respectively.

4.5 Results and Discussion

In this section, the experimental results of the comparison of the proposed methods (DIUNet and DEUNet) with the benchmarks are provided. Average network sum-rate, qos violation probability, quota violation probability, and per sample test time are used as the comparison metrics. Similar to chapter 3, the methods are tested over two thousand datapoints and the computation time was obtained by averaging over ten independent trials over the datapoints.

In all the experiments, an FCNN with three 200-dimensional hidden layers with ReLU activation is used as the backbone DNN ($\mathcal{N}_r$). To accelerate the training and prevent over-fitting, Batch normalization [53] and Dropout [54] are used, respectively. All DNNs are trained for 20 epochs with learning rate of 0.001 and batch size of 10 and learning rate decay of 0.99 using ADAM optimization algorithm [62]. The λ is set to 1 for DEUNet and UNet. Furthermore, PyTorch [55] is used for DNN's implementation. For ILP and LP, I used the CVXPY [57] package with MOSEK's [16] and ECOS's [56] solver's backbone, respectively. The experiments were done on a desktop computer with an Intel Core i7-8700 CPU 3.20GHz and 8GB of RAM. In the following, I compare the performance of all the methods.

4.5.1 ILP and LP vs DIUNet and DEUNet

Besides Theorem 1, the equivalency of (4.4) and (4.17) can be experimentally verified by considering Fig. 4.1 and 4.4.

As expected, solving the equivalent linear program (4.17) instead of the original integer program (4.4) is computationally more efficient (Fig. 4.3).

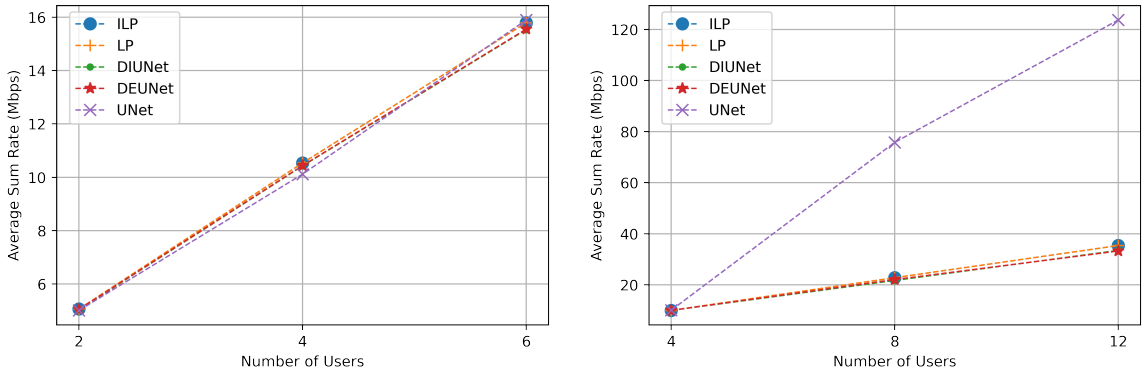

Figure 4.1: Comparison of network sum-rate for ILP, LP, DIUNet, DEUNet, and UNet. (Left: Gaussian Dataset - Right: Pathloss Dataset).

Considering the DIUNet, I can observe that it results in a network sum-rate that is very close to ILP and LP for both Gaussian and Path-loss datasets (Fig. 4.1). Although the computation time of DIUNet is significantly lower than ILP, it is slower than LP. The main reason is that a linear program is used to formulate the projection function of DIUNet (4.24). Both linear programs can be solved in the same time, but DIUNet has a DNN backbone as well. This explains the difference between the computation time of DIUNet and LP.

DEUNet, on the other hand, performs very close to ILP and LP in terms of network sum-rate, but has a lower time complexity than LP. The downside of it is that once the number of users starts to increase the BS quota constraints start to get violated (Fig. 4.5). The violation is negligible comparing to computational benefit of

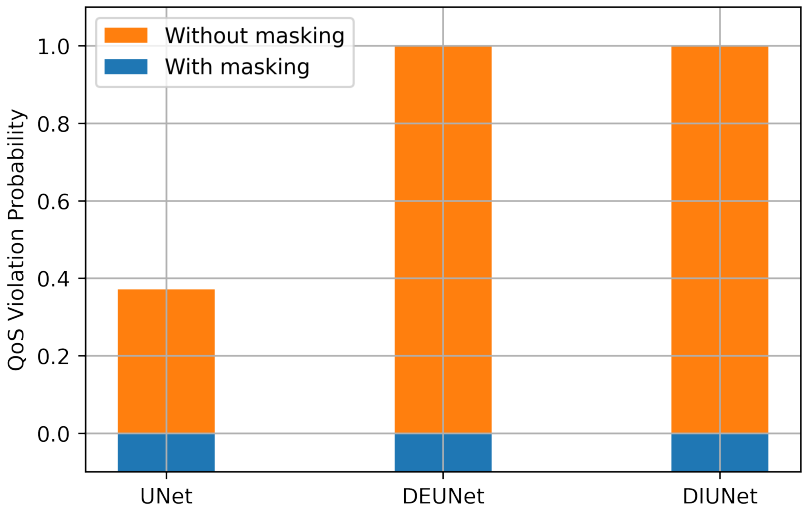

Figure 4.2: Comparison of QoS violation probability for DIUNet, DEUNet, and UNet with and without using the masking technique.

it; thus, in applications where time is a critical factor, DEUNet can be used instead of LP.

4.5.2 UNet vs DIUNet vs DEUNet

Comparing DIUNet and DEUNnet, I can see that they perform very closely in terms of network sum-rate. However, DEUNet is faster than DIUNet since it doesn't require solving an optimization problem for the projection function. When it comes to the BS quota violation, however, I can see that DIUNet never violates this constraint, but DEUNet has a negligible probability of violating it.

Comparing both of the models with UNet, I can see that it always violates the BS quota constraint a probability that grows as the problem size increases. This becomes a significant downside once considering a realistic setup with high number of users like 8 and 12 users of Path-loss datasets (Fig. 4.5 (right figure)). The network sum-rate of UNet gets better than even LP once the problem size increases. This is,

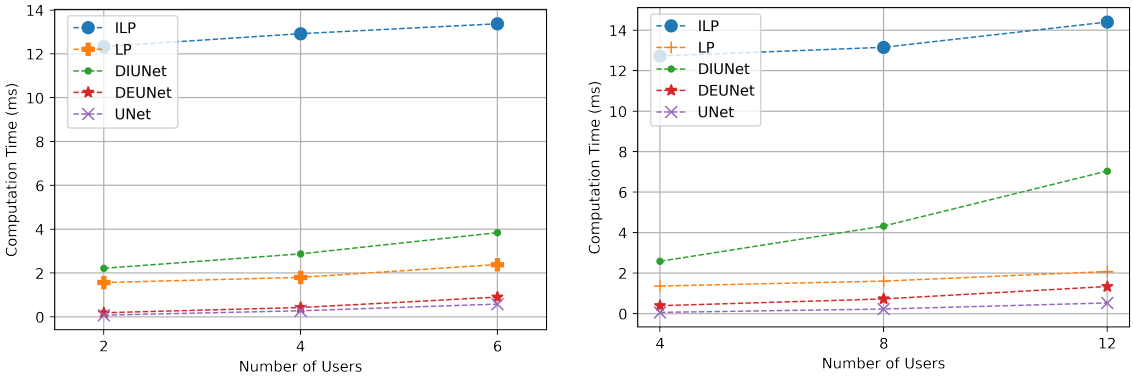

Figure 4.3: Comparison of computation time for ILP, LP, DIUNet, DEUNet, and UNet. (Left: Gaussian Dataset - Right: Pathloss Dataset).

however, at the cost of violating the BS quota constraint. Moreover, I can observe that since UNet doesn't have a complex projection function, it can outperform all of the methods in terms of computation time.

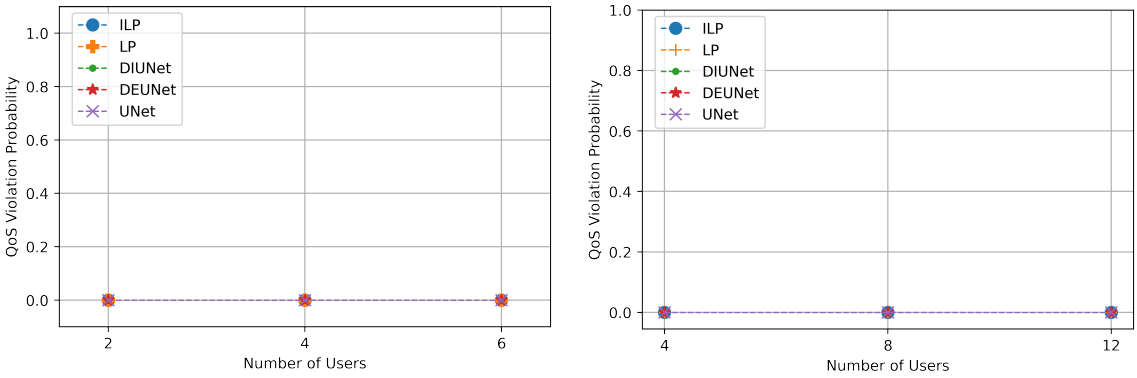

Figure 4.4: Comparison of QoS violation probability for ILP, LP, DIUNet, DEUNet, and UNet. (Left: Gaussian Dataset - Right: Pathloss Dataset).

As a final note, I can conclude that if the BS quota constraint is not strict and time is a critical factor, the best method is UNEt. Otherwise, the best choice is LP. DEUNet, as mentioned before, lies in between. Moreover, by considering all the DNN-based methods, I can see the effect of the masking trick in Fig. 4.2, which results in zero QoS violation probability.

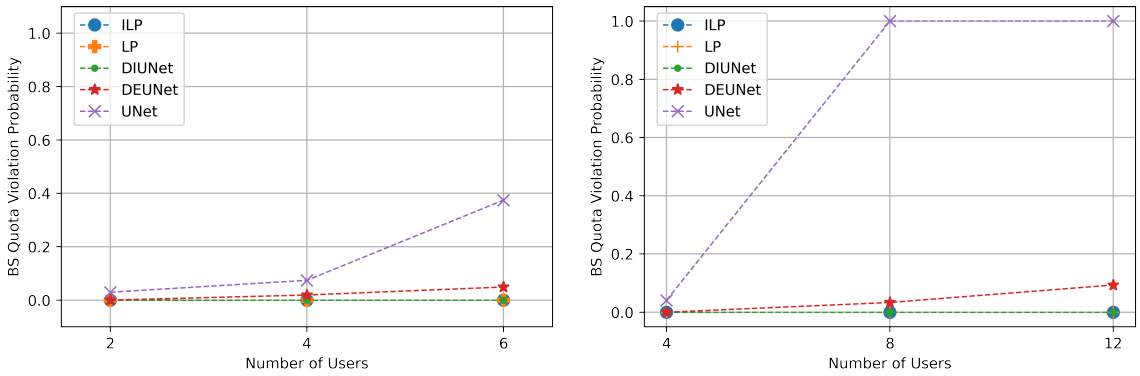

Figure 4.5: Comparison of BS quota violation probability for ILP, LP, DIUNet, DEUNet, and UNet. (Left: Gaussian Dataset - Right: Pathloss Dataset).

4.5.3 Alternating Optimization vs DIPNet + DIUNet

In this experiment, I use DIPNet from the previous chapter and DIUNet to solve power control and user association in a joint manner (problem (4.2)). Starting from DIUNet, I get the user association decision, which is then passed to DIPNet to get the transmit powers. I call this method JUPNet. For the optimization-based solver, called ALT, I used the LP solver from this chapter and GP from the previous chapter. They work in an alternating manner, starting from user association, for five iterations to solve this problem.

Table 4.1: Joint User Association and Power control

Dataset ID	JUPNet			Alternating Optimization (ALT)		
	Rate	Violation	Test time	Rate	Violation	Test time
1	4.44	0	0.91	4.39	0	124.45
2	8.95	0	1.05	8.69	0	481.27
3	13.06	0	1.14	12.87	0	1076.92

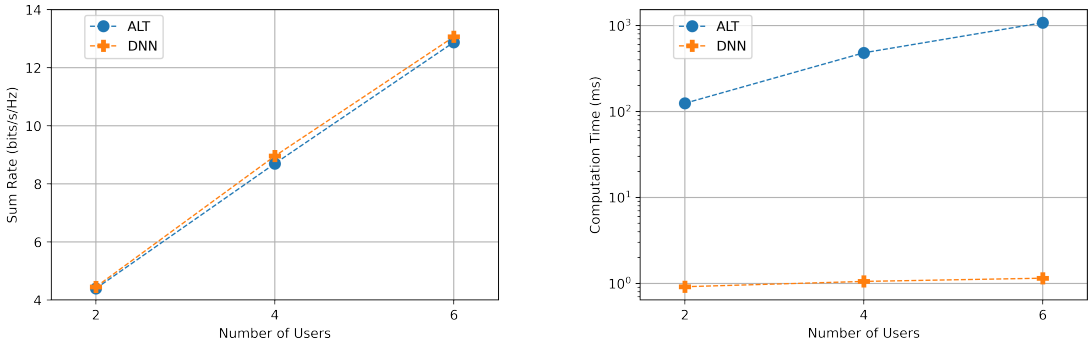

Figure 4.6: A comparison of DNN (JUPNet) and ALT (Left: network sum-rate - Right: Computation Time).

As I can see from Table 4.1, the DNN-based solver outperforms the alternating optimization approach in both network sum-rate and time complexity, while achieving zero QoS violation probability.

4.6 Summary

In this chapter, the user association problem with QoS constraint is addressed. The problem is first transformed to its continuous equivalent to make it a good fit for DNN-based solvers. A trick called masking is developed for DNNs to systematically address the QoS constraints. Two projection function is further designed to project the output of the DNN to Birkhoff polytope, resulting in DIUNet and DEUNet. Experimental results showed superior performance of the proposed methods compared to the DNN-based benchmark in terms of BS quota violation. Cascading DIUNet and DIPNet from the previous chapter resulted in a joint solver of user association and power control, which outperformed the optimization-based benchmark in terms of time complexity and network sum-rate. The solution can be applied to various problems in 6G Terahertz networks [63, 64].

Chapter 5

Conclusions and Future Directions

5.1 Conclusion

In this work, I utilize DNNs to solve power control and user association problems in wireless communication systems. To handle the constraints of the optimization problems, specific projection functions were designed. For both of the problems, I utilized both an optimization problem and an iterative process to define the projection functions implicitly and explicitly, respectively.

For the power control, I observed that using a DNN with implicit or explicit projection can significantly reduce the computation time while ensuring the QoS constraint. Although the implicit projection function always satisfied the constraints, the explicit projection showed some violation probability for more complex channel models (Path-loss). The upside of the explicit projection is its faster speed and GPU-friendly implementation. Moreover, the Frank-Wolfe algorithm was added after the DNNs, and I observed a boost in terms of network sum-rate. The cost was an increase in time complexity, which still was lower than the optimization-based benchmark. The solution can be used for scheduling problems such as [65–67]

Despite the continuous nature of power control, user association deals with discrete variables. To make it a good fit for DNNs, I applied a new transformation, which transformed the original integer program into a linear program (LP). I proved that the optimal solution of LP is the same as the original problem and will satisfy the QoS constraint. Two projection functions were proposed, afterward, to solve this problem with DNNs. The implicit projection uses mathematical optimization for defining the projection function. The explicit one, on the other hand, uses an iterative process.

Furthermore, a new trick called masking was used to make sure that the output of DNN-based solvers won't violate the QoS constraint. In the experiments, both projection methods performed very close to optimization-based solvers in terms of network sum-rate. Moreover, I observed that the implicit one had a higher time complexity than LP, but the explicit one had lower time complexity. The cost of it was that explicit projection violates the BS quota constraint as the problem size grows, which was negligible. Finally, the DNNs with implicit projection were combined to solve the joint power control and user association problem. The results showed zero QoS violation probability and a close network sum-rate to optimization-based approaches while having low computational complexity. The solution can be used for wireless scheduling problems in various context such as [65–68]

5.2 Future Directions

5.2.1 Synthetic vs Real-world data

Although the proper choice of architecture can provide lower sample complexity, better generalization, and scalability, other considerations have to be taken into account before moving toward the practicality of L2O. Most of the existing works made an

assumption of accessibility of the perfect channel state information (CSI) [12,37], and used synthetic data to train and validate their model. This is not true in practice and an estimation error is always at place [37]. Moreover, there is no consensus on what is the appropriate data to use. Although most of the works released their experiments on CSI, consisting of path-loss shadowing and fading effects, some works tried only path-loss or GPS information to come up with RRM solutions [69].

5.2.2 Injection of prior knowledge

It is well-known theoretical evidence that FCNNs with at least one hidden layer are universal function approximators [19]. This means that as long as the mapping that I am trying to approximate shows some qualities, e.g. smoothness, [27], the FCNNs family contains a function that satisfies the chosen approximation error. To find this function via learning, I can either cover most of the input space, which is impossible for higher dimensions due to the curse of dimensionality [27], or reduce the search space of the potential functions [70]. This means that I first identify the more specific conditions that the desired mapping has, and then try to search over the ANNs that systematically satisfy those conditions, this will reduce the sample complexity while providing a better generalization ability [27, 70].

This is known as finding proper inductive biases or priors for the task at hand. One successful example is the use of convolution layers on images since both demonstrate the transition equivariance property [70]. The quest for identifying priors of RRM problems in wireless is still in its infancy but has shown good results over just using FCNN [12] or blindly applying architectures that have worked well on the other domains [6].

Since the data domain in RRM problems are different from image or text in computer vision (CV) and natural language processing (NLP), a skeptical mindset needs to be in place while applying neural network-based models adopted from DL community to RRM problems. This has to be followed with an effort toward identifying underlying prior knowledge of RRM problems and designing models based on these priors. An example of this is the works in [71], [37], where the permutation equivariance of wireless tasks is identified and utilized by designing permutation equivariant linear maps and using graph neural networks, respectively. This resulted in lower training sample complexity and enhanced scalability.

5.2.3 Unified Optimization to DL Conversion

Another major issue is the lack of a unified framework for converting optimization problems in RRM into learning problems with clear guidance on the prepossessing of the data (CSI), levels of supervision, choice of the loss function and evaluation metrics, and the architectural design of the neural network.

5.2.4 Customized Projections

One can design a better objective function by making k to have learnable parameters or to reflect the qualities of a potential maximizer of (2.1). By parametrizing k, I can have parameters in the objective that can potentially be a function of the problem data, i.e. $\boldsymbol{\theta} = a(\mathbf{x})$. I can go one step further and make these parameters learnable, i.e. $\boldsymbol{\theta} = a(\mathbf{x}; \mathbf{w}_\theta)$ where a is a parametric function which can be realized by a DNN. Furthermore, one can redefine (2.5) to have different constraints or reformulation of the main problem's constraints. One example is to use a convex approximation in the

presence of non-convexity in the original problem's feasible set. This can be beneficial from a practical perspective and will expand the applicability of (2.5) as a projection function. These are directions left for future endeavors.

Bibliography

[1] B. Amos, "Tutorial on amortized optimization for learning to optimize over continuous domains," *arXiv preprint arXiv:2202.00665*, 2022.

[2] P. L. Donti, D. Rolnick, and J. Z. Kolter, "Dc3: A learning method for optimization with hard constraints," *arXiv preprint arXiv:2104.12225*, 2021.

[3] M. Rasti, S. Kazemi Taskou, H. Tabassum, and E. Hossain, "Evolution toward 6g wireless networks: A resource management perspective," *arXiv e-prints*, pp. arXiv–2108, 2021.

[4] B. G. Lee, D. Park, and H. Seo, *Wireless communications resource management*. John Wiley & Sons, 2009.

[5] C. She, C. Sun, Z. Gu, Y. Li, C. Yang, H. V. Poor, and B. Vucetic, "A tutorial on ultrareliable and low-latency communications in 6g: Integrating domain knowledge into deep learning," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 204–246, 2021.

[6] W. Lee, M. Kim, and D.-H. Cho, "Deep power control: Transmit power control scheme based on convolutional neural network," *IEEE Communications Letters*, vol. 22, no. 6, pp. 1276–1279, 2018.

[7] C. Sun, C. She, and C. Yang, "Unsupervised deep learning for optimizing wireless systems with instantaneous and statistic constraints," *arXiv preprint arXiv:2006.01641*, 2020.

[8] M. Chiang, P. Hande, and T. Lan, *Power control in wireless cellular networks*. Now Publishers Inc, 2008.

[9] Y. Shen, J. Zhang, S. Song, and K. B. Letaief, "Ai empowered resource management for future wireless networks," in *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, 2021, pp. 252–257.

[10] K. Shen and W. Yu, "Fractional programming for communication systems—part ii: Uplink scheduling via matching," *IEEE Transactions on Signal Processing*, vol. 66, no. 10, pp. 2631–2644, 2018.

[11] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for interference management," *IEEE Transactions on Signal Processing*, vol. 66, no. 20, pp. 5438–5453, 2018.

[12] F. Liang, C. Shen, W. Yu, and F. Wu, "Towards optimal power control via ensembling deep neural networks," *IEEE Transactions on Communications*, vol. 68, no. 3, pp. 1760–1776, 2019.

[13] C. Sun and C. Yang, "Learning to optimize with unsupervised learning: Training deep neural networks for urllc," in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2019, pp. 1–7.

[14] M. Chiang, C. W. Tan, D. P. Palomar, D. O'neill, and D. Julian, "Power control by geometric programming," *IEEE transactions on wireless communications*, vol. 6, no. 7, pp. 2640–2651, 2007.

[15] A. Kaushik, M. Alizadeh, O. Waqar, and H. Tabassum, "Deep unsupervised learning for generalized assignment problems: A case-study of user-association in wireless networks," *arXiv preprint arXiv:2103.14548*, 2021.

[16] M. ApS, *MOSEK Optimizer API for Python 9.3.20*, 2019. [Online]. Available: https://docs.mosek.com/9.3/pythonapi/index.html

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[18] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *Advances in neural information processing systems*, vol. 31, 2018.

[19] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005801315

[20] Z. Kolter, D. Duvenaud, and M. Johnson, "Deep implicit layers," http://implicit-layers-tutorial.org.

[21] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in neural information processing systems*, vol. 32, 2019.

[22] S. Gould, R. Hartley, and D. Campbell, "Deep declarative networks: A new hope," *arXiv preprint arXiv:1909.04866*, 2019.

[23] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," https://locuslab.github.io/2019-10-28-cvxpylayers/.

[24] B. Amos, "Differentiable optimization-based modeling for machine learning," *Ph. D. thesis*, 2019.

[25] A. Martins and R. Astudillo, "From softmax to sparsemax: A sparse model of attention and multi-label classification," in *International conference on machine learning*. PMLR, 2016, pp. 1614–1623.

[26] G. Mena, D. Belanger, S. Linderman, and J. Snoek, "Learning latent permutations with gumbel-sinkhorn networks," *arXiv preprint arXiv:1802.08665*, 2018.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[28] J. F. Henriques, S. Ehrhardt, S. Albanie, and A. Vedaldi, "Small steps and giant leaps: Minimal newton solvers for deep learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4763–4772.

[29] J. Hendriks, C. Jidling, A. Wills, and T. Schön, "Linearly constrained neural networks," *arXiv preprint arXiv:2002.01600*, 2020.

[30] J. Gao, C. Zhong, G. Y. Li, and Z. Zhang, "Online deep neural network for optimization in wireless communications," *IEEE Wireless Communications Letters*, vol. 11, no. 5, pp. 933–937, 2022.

[31] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[32] A. Zappone, M. Di Renzo, and M. Debbah, "Wireless networks design in the era of deep learning: Model-based, ai-based, or both?" *IEEE Transactions on Communications*, vol. 67, no. 10, pp. 7331–7376, 2019.

[33] Z. Deng, Q. Sang, Y. Pan, and Y. Xin, "Application of deep learning for power control in the interference channel: a rnn-based approach," in *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, 2019, pp. 96–100.

[34] Q. Shi, M. Razaviyayn, Z.-Q. Luo, and C. He, "An iteratively weighted mmse approach to distributed sum-utility maximization for a mimo interfering broadcast channel," *IEEE Transactions on Signal Processing*, vol. 59, no. 9, pp. 4331–4340, 2011.

[35] N. Naderializadeh, "Contrastive self-supervised learning for wireless power control," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 4965–4969.

[36] W. Cui, K. Shen, and W. Yu, "Deep learning for robust power control for wireless networks," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8554–8558.

[37] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "Graph neural networks for scalable radio resource management: Architecture design and theoretical analysis," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 101–115, 2020.

[38] M. Eisen and A. Ribeiro, "Optimal wireless resource allocation with random edge graph neural networks," *ieee transactions on signal processing*, vol. 68, pp. 2977–2991, 2020.

[39] N. Naderializadeh, M. Eisen, and A. Ribeiro, "Wireless power control via counterfactual optimization of graph neural networks," *arXiv preprint arXiv:2002.07631*, 2020.

[40] Y. Li, S. Han, and C. Yang, "Multicell power control under rate constraints with deep learning," *IEEE Transactions on Wireless Communications*, vol. 20, no. 12, pp. 7813–7825, 2021.

[41] D. Liu, L. Wang, Y. Chen, M. Elkashlan, K.-K. Wong, R. Schober, and L. Hanzo, "User association in 5g networks: A survey and an outlook," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1018–1044, 2016.

[42] Y. Xu and S. Mao, "User association in massive mimo hetnets," *IEEE Systems Journal*, vol. 11, no. 1, pp. 7–19, 2015.

[43] Y. Zhang, L. Xiong, and J. Yu, "Deep learning based user association in heterogeneous wireless networks," *IEEE Access*, vol. 8, pp. 197 439–197 447, 2020.

[44] A. Zappone, L. Sanguinetti, and M. Debbah, "User association and load balancing for massive mimo through deep learning," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2018, pp. 1262–1266.

[45] G. Jia, Z. Yang, H.-K. Lam, J. Shi, and M. Shikh-Bahaei, "Channel assignment in uplink wireless communication using machine learning approach," *IEEE Communications Letters*, vol. 24, no. 4, pp. 787–791, 2020.

[46] K. Shen and W. Yu, "Fractional programming for communication systems—part i: Power control and beamforming," *IEEE Transactions on Signal Processing*, vol. 66, no. 10, pp. 2616–2630, 2018.

[47] H. Zarini, A. Khalili, H. Tabassum, M. Rasti, and W. Saad, "Alexnet classifier and support vector regressor for scheduling and power control in multimedia heterogeneous networks," *IEEE Transactions on Mobile Computing*, 2021.

[48] X. Cao, R. Ma, L. Liu, H. Shi, Y. Cheng, and C. Sun, "A machine learning-based algorithm for joint scheduling and power control in wireless networks," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4308–4318, 2018.

[49] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145.

[50] S. Lacoste-Julien, "Convergence rate of frank-wolfe for non-convex objectives," *arXiv preprint arXiv:1607.00345*, 2016.

[51] L. P. Qian, Y. J. Zhang, and J. Huang, "Mapel: Achieving global optimality for a non-convex wireless power control problem," *IEEE Transactions on Wireless Communications*, vol. 8, no. 3, pp. 1553–1563, 2009.

[52] Y. Shi, J. Zhang, and K. B. Letaief, "Group sparse beamforming for green cloud-ran," *IEEE Transactions on Wireless Communications*, vol. 13, no. 5, pp. 2809–2823, 2014.

[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[54] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[56] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *European Control Conference (ECC)*, 2013, pp. 3071–3076.

[57] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[58] S. Linderman, G. Mena, H. Cooper, L. Paninski, and J. Cunningham, "Reparameterizing the birkhoff polytope for variational permutation inference," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 1618–1627.

[59] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment problems: revised reprint*. SIAM, 2012.

[60] F. Dufossé and B. Uçar, "Notes on birkhoff–von neumann decomposition of doubly stochastic matrices," *Linear Algebra and its Applications*, vol. 497, pp. 108–115, 2016.

[61] R. P. Adams and R. S. Zemel, "Ranking via sinkhorn propagation," *arXiv preprint arXiv:1106.1925*, 2011.

[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[63] N. Hassan, M. T. Hossan, and H. Tabassum, "User association in coexisting rf and terahertz networks in 6g," in *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, pp. 1–5.

[64] H. Ibrahim, H. Tabassum, and U. T. Nguyen, "The meta distributions of the sir/snr and data rate in coexisting sub-6ghz and millimeter-wave cellular networks," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 1213–1229, 2020.

[65] H. Tabassum, E. Hossain, M. J. Hossain, and D. I. Kim, "On the spectral efficiency of multiuser scheduling in rf-powered uplink cellular networks," *IEEE transactions on wireless communications*, vol. 14, no. 7, pp. 3586–3600, 2015.

[66] S. Zarandi, A. Khalili, M. Rasti, and H. Tabassum, "Multi-objective energy efficient resource allocation and user association for in-band full duplex small-cells," *IEEE Transactions on Green Communications and Networking*, vol. 4, no. 4, pp. 1048–1060, 2020.

[67] I. A. Umoren, M. Z. Shakir, and H. Tabassum, "Resource efficient vehicle-to-grid (v2g) communication systems for electric vehicle enabled microgrids," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4171–4180, 2020.

[68] S. B. Melhem and H. Tabassum, "User pairing and outage analysis in multi-carrier noma-thz networks," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 5, pp. 5546–5551, 2022.

[69] W. Cui, K. Shen, and W. Yu, "Spatial deep learning for wireless scheduling," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1248–1261, 2019.

[70] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," 2021. [Online]. Available: https://arxiv.org/abs/2104.13478

[71] J. Guo and C. Yang, "Structure of deep neural networks with a priori information in wireless tasks," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.