Deep Learning for Resource Allocation in Cellular Wireless Networks

by

Kazi Ishfaq Ahmed

A Thesis submitted to The Faculty of Graduate Studies of

The University of Manitoba

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg

May 2019

Abstract

Optimal resource allocation is a fundamental challenge for dense and heterogeneous wireless networks with massive wireless connections. Traditionally, due to the non-convex nature of the optimization problem, resource allocation is done using some heuristic approaches such as exhaustive search, genetic algorithms, combinatorial and branch and bound techniques. These methods are computationally expensive and therefore not appealing for large-scale heterogeneous cellular networks with ultra-dense base station (BS) deployments, massive connections and diverse QoS requirements for different classes of users. As a result, the next generation of wireless networks will require a paradigm shift from traditional resource allocation mechanisms. Deep learning (DL) is a powerful tool where a multi-layer neural network can be trained to model a resource management algorithm using network data. Therefore, resource allocation decisions can be obtained without intensive online computations which would be required otherwise for the solution of resource allocation problems. In this thesis, I develop a deep learning-based resource allocation framework for multi-cell wireless networks with an objective to maximizing the total network throughput. In addition, I explore the deep reinforcement learning (DRL) approach to perform a near-optimal downlink power allocation for multi-cell wireless networks. Specifically, I use a deep Q-learning (DQL) strategy to achieve near-optimal power allocation policy. For benchmarking the proposed approaches, I use a Genetic Algorithm (GA) to obtain near-optimal resource allocation solution. I compare the proposed power allocation scheme with other traditional power allocation schemes by running numerous simulations.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

5G	Fifth Generation
AE	Auto-encoder
BER	Bit Error Rate
BS	Base Station
CNN	Convolutional Neural Network
CQI	Channel Quality Indicator
CR	Cognitive Radio
D2D	Device-to-Device
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network
DL	Deep Learning
DNN	Deep Neural Network
DQL	Deep Q-Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
GA	Genetic Algorithm
GPU	Graphics Processing Unit
ICI	Inter Cell Interference
LSTM	Long Short-Time Memory

ML	Machine Learning
MSE	Mean Squared Error
PA	Power Allocation
PCA	Principal Component Analysis
QoS	Quality Of Service
RELU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SAE	Stacked Auto-encoder
SBS	Small Base Station
SDN	Software Defined Network
SFP	Sequential Fractional Programming
SINR	Signal-to-interference-plus-noise Ratio
UPN	Unsupervised Pre-trained Network
V2V	Vehicle-to-Vehicle
WMMSE	Weighted Minimum Mean Squared Error

Publications

- Magazine Publications:

    1. **Kazi Ishfaq Ahmed**, Hina Tabassum, and Ekram Hossain, "Deep Learning for Radio Resource Allocation in Multi-cell Networks," *IEEE Network*, 2019

- Conference Publications:

    1. **Kazi Ishfaq Ahmed**, and Ekram Hossain, "A Deep Q-Learning Method for Downlink Power Allocation in Multi-Cell Networks," submitted to the *2019 IEEE Global Communications Conference: Cognitive Radio and AI-Enabled Network Symposium.*

Chapter 1

Introduction

Densification of cellular wireless networks through the deployment of more base stations [BSs] (or cells) has emerged as a possible technique to meet the communication requirements for the ever-increasing number of mobile users as well as their demand for high data rate. This gives rise to the concept of heterogeneous networks (HetNets) in which traditional high-power macro-cells are underlaid with low-power small-cells to increase the network capacity. However, the main challenge in a HetNet (especially in a dense scenario) is efficient interference management among users by prudent use of the scarce radio resources such as frequency bandwidth, power and time. Since the same frequency sub-bands are reused in different cells, users may experience strong inter-cell interference (ICI). In an ultra-dense scenario, the interference becomes much severe as the base stations (e.g. macro-cell and small cell base stations) can be very close to each other. Efficient resource allocation among the cells and users, which is often posed as an optimization problem, can minimize this ICI.

1.1 Challenges in Optimal Resource Allocation

Optimal radio resource allocation is one of the fundamental challenges for the design and operation of cellular wireless networks. The resource allocation problems are often formulated and solved using tools from the optimization theory. These problems need to be solved for sophisticated network scenarios with complex wireless channels, new transmitter and receiver architectures, and diverse quality-of-service (QoS) requirements of the users. When the optimization problems are non-convex (which is often the case in a real-world scenario), the optimal solutions are obtained by applying exhaustive search methods, genetic algorithms, combinatorial, and branch and bound techniques which results in high computational complexities. Therefore, these methods are not appealing for large-scale heterogeneous cellular networks with ultra-dense base station (BS) deployments, massive connections, and different resources (e.g. transmission time, frequency channels, antennas, transmit power) and diverse QoS requirements for different classes of users. Sub-optimal solutions based on techniques such as Lagrangian relaxations, iterative distributed optimization, heuristic algorithms, and auction/game theory may also be computationally intensive and can be far from optimality. Their convergence and optimality gap could be unknown as well.

1.2 Motivation

The main motivation of this thesis is to address the challenge of optimal resource allocation for next-generation wireless networks. Compared to the aforementioned traditional techniques, machine learning (ML) [1] -based resource allocation algorithms can be implemented online with reduced complexity and can optimize resource allocations in complicated networks. Based on the type of ML technique, the required informa-

tion can be learned directly from the data samples instead of using less tractable mathematical models. However, significant human intervention may be required to make accurate inferences and decisions based on the dimensionality of data, which is referred to as *feature engineering*.

Deep learning (DL), which is a sub-class of ML, hierarchically extracts high-level features and correlations from input data without human efforts through multiple layers of nonlinear processing units. DL models enable end-to-end optimization of a complete communication system having several processing blocks, such as channel encoding, modulation, and signal detection, instead of optimizing each block separately. Note that a communication system suffers from a variety of hardware and transmission channel impairments including inter-modulation and amplifier distortion, quantization loss, quadrature imbalance, multi-path fading, interference, and path-loss. Furthermore, various kind of resources (or degrees of freedom), such as antennas, channels, bands, beams, codes, and bandwidths, are available that need to be efficiently utilized. The combination of these impairments and degrees of freedom makes the optimization of a complete communication system using the traditional techniques exceedingly difficult. This motivates the use of data-driven optimization of a communication system using DL techniques.

A trained DL model can thus achieve multiple objectives without the need of retraining and Graphics Processing Unit (GPU)-based parallel computing enables DL to make inferences within milliseconds [2, 3]. Along this line, the development of artificial intelligence-powered devices, such as Intel Movidius Neural Compute Stick, will also motivate the use of DL for future wireless communications applications. With the DL approach for radio resource management in a large cellular network, a multi-layer neural network model can be built to which the relevant training samples

can be provided as inputs, and then the resource allocation solution can be obtained as outputs without intensive computations (as each layer performs simple operations like matrix-vector multiplications [4]). Furthermore, with the advent of software defined networking (SDN) [5] and cloud storage technologies, it is possible to deal with the storage, analysis, and computation of *big data* as well as training the DL models on SDN controllers.

On the other hand, Reinforcement Learning (RL) [6] method can obtain the optimal solution of a control problem by interacting with the environment. An agent in RL interacts with the environment by taking action and then receiving feedback from the environment in terms of reward. The agent follows a policy to maximize some notion of discounted cumulative reward through a series of actions. However, the traditional RL is suitable only for systems with low-dimensional state space and may not work for systems with high-dimensional state space. This is because, in conventional RL, a policy is stored in tabular form and it is not feasible for massive action and state space due to the lack of generalization. So, instead of a tabular method, a function approximation such as a DNN can be used in that case. Recently, DRL [7] has emerged as a promising technique to handle complicated control problems. By combining Deep Learning (DL) with Reinforcement Learning (RL), the DRL can extract useful information from high-dimensional data and can learn the optimal action policy.

1.3 Deep Learning-Based Wireless Resource Allocation: State-of-the-Art

DL techniques have recently been used in a variety of wireless resource management problems (e.g. channel and power allocation, throughput maximization, spectrum

sharing). We categorize the major works into two groups: supervised DL and DRL. A summary of these works is provided in Table 1.1 for a qualitative comparison among these works in terms of the inputs and outputs, data training and generation method used, radio resources considered, objectives of resource management, and their limitations.

1.3.1 Supervised Learning

In [8], the authors propose a distributed power control method based on a data-driven convolutional neural network (CNN) which exploits the spatial features in channel gain to estimate the transmit power. The main objective is to maximize throughput through power allocation. However, there is no guarantee that the proposed method can give the centrally optimized solution. In [9], the authors develop an energy-efficient power control scheme based on a trained DNN which takes the communication channels as input and predicts the transmit power. The primary objective is to maximize the energy efficiency. In [10], the authors formulate a weighted throughput maximization problem and solved it using a recurrent DNN architecture. The model is trained by applying the concept of universal function approximation of DNN and Lagrangian duality. That is, the non-convex problem is formulated as a finite-dimensional unconstrained optimization problem in the dual domain with bounded sub-optimality. Primal-dual descent methods as well their zeroth-ordered equivalents are used to perform data generation.

In [11], throughput maximization of a device-to-device (D2D) network is considered with maximum power constraint of a D2D transmitter and maximum interference received by cellular BSs. The input data is generated through simulations. The accuracy/quality of the learning solutions is, however, not described clearly. Also,

in [4], a throughput maximization problem is considered with power allocation variables in interference-limited wireless networks using supervised DL by generating data through approximate optimal solutions. The model takes the channel coefficients as input and allocate power as output.

1.3.2 Deep Reinforcement Learning (DRL)

In [12], the authors use a DQL for power allocation in a cloud-RAN to minimize the total power consumption while ensuring the demand of each user. The proposed model takes the demand of the users as input and then allocate powers to the remote radio heads. The proposed resource allocation scheme is not centrally optimized. In [13], the authors propose a distributed DRL based resource allocation scheme for a vehicle-to-vehicle (V2V) communication system. The main objective is to minimize interference with latency constraints. The model takes the instantaneous channel as an input and allocates both channel and power. In [14], the authors model the throughput maximization problem of a small cell network with unlicensed spectrum as a non-cooperative game and propose a distributed DRL-based solution. The primary objective is to maximize throughput in each small cell while maintaining fairness with co-existing networks. The authors use a long short-time memory (LSTM) [20] network as an agent who takes the unlicensed channel information as an input and allocates channel in the time domain. In [15], the authors develop a distributed DQL based spectrum sharing approach for primary and secondary users in a non-cooperative fashion. The primary users use a fixed power control strategy while the secondary users learn autonomously to adjust the transmission power to share the shared spectrum. The model takes the SINR information as input and then allocate power.

Table 1.1: Summary of existing works on resource allocation using deep learning

Publication	**Model**	**Objective**	**Resources**	**Input**	**Approach**
Sun *et al.* [4]	DNN	Throughput maximization	Power	Channel coefficients	Centralized
Lee *et al.* [8]	CNN	Maximize throughput through power control	Power	Normalized channel	Centralized, distributed
Zappone *et al.* [9]	DNN	Maximize the energy efficiency through power control	Power	Channel	Centralized
Eisen *et al.* [10]	RNN	Weighted throughput maximization	Power	Channel gain	Centralized
Kim *et al.* [11]	DNN	Throughput maximization with power and interference constraints	Power	Unclear	Distributed
Xu *et al.* [12]	DQL	Minimize the total power consumption with users' QoS constraints	Power	Demand of the users	Centralized
Ye *et al.* [13]	DQL	Interference minimization with latency constraints	Channel, Power	Instantaneous channel	Distributed
Challita *et al.* [14]	DQL	Maximize throughput with fairness constraints	Channel, Time	Unlicensed channel	Distributed
Li *et al.* [15]	DQL	Spectrum sharing with power control at secondary user	Power	SINR	Distributed
Zhao *et al.* [16]	DQL	Maximize the long-term utility of the network with users' QoS constraints	Channel	QoS of the users	Distributed
Nasir *et al.* [17]	DQL	Maximize the weighted sum-rate of the network through power control	Power	CSI	Distributed
Meng *et al.* [18]	DQL	Maximize the overall sum-rate of the network through power control	Power	Normalized CSI	Distributed
Naparstek *et al.* [19]	DQL	Maximization of certain network utility in distributed manner	Channel	Channel capacity	Distributed

In [16], the authors use the DRL approach to perform joint user association and resource allocation (UARA) in the downlink of the heterogeneous network. Notably, the authors use Double Deep Q-Network (DDQN) [21] to obtain the optimal policy. The ultimate goal is to maximize the long-term utility of the network while ensuring QoS requirements. In [17], the authors use a multi-agent DQL approach to allocate power in wireless networks. The principal objective is to maximize the weighted sum-rate of the system. The random variations in CSI, as well as the delays in CSI, are incorporated in the deep Q-learning model. In [18], the authors propose different DRL architectures such as *REINFORCE*, DQL and deep deterministic policy gradient (DDPG) [22] for power allocation in multi-user cellular networks. The ultimate target is to maximize the overall sum-rate of the network. Simulation results show that the deep Q-learning performs better compared to other DRL approaches. In [19], the authors use the DRL approach for dynamic spectrum access in wireless networks. More specifically, the authors use LSTM based deep Q-learning approach. The primary goal is to maximize each user's specific network utility in a distributed manner, i.e. without exchanging information.

1.4 Scope and Contributions of the Thesis

The existing DL-based methods generally focus on single variables only (e.g. power) and consider heuristic algorithms or simulations to generate the data and train the model. Therefore, learning solutions are not efficient solutions. On the other hand, most of the DRL-based methods focus on power allocation at the small base stations (SBSs) or cognitive radios (CRs) on a small-scale network set up and take a completely distributed DRL approach (e.g. the SBSs or the CRs do not cooperate or exchange information among themselves). Some of the studies perform power allo-

cation on multi-cell wireless networks on a large scale but in a distributed manner. Therefore, the solutions can be very far from the optimal solution. Also, these studies either consider one user with multiple subbands per cell or multi-user with one shared frequency band per cell. Therefore, these approaches do not apply to multi-cell networks with multiple users per cell sharing the same frequency subbands.

The main contributions of this thesis are summarized as follows:

- I provide a brief overview of the DL architectures and the training and testing procedure of the DL model. I also discuss how the DL model is applicable to solve a resource allocation problem in a wireless network.

- I formulate the resource allocation problem of a multi-cell network having multiple users sharing the same frequency subbands. Then, I propose a supervised DL model to compute optimal sub-band and power allocation solutions for a multi-cell network with an objective to maximize the total network throughput. This is a well-known non-convex combinatorial optimization problem. As such, to generate near-optimal training data for the DL model and to test the DL model, I utilize a genetic algorithm (GA). I also define the offline training and online testing procedure of the proposed model.

- I investigate the limitations of the supervised DL approach for resource allocation in wireless networks. To overcome the limitations, I propose a centralized DRL-based resource allocation scheme for multi-cell networks. The proposed scheme is one of the first such schemes to address the power allocation problem under maximal power constraints in a multi-cell network having multiple users sharing the same frequency subbands. I define the state space, action space and the reward function for the DRL agent. I also define the online training procedure of the proposed DRL-based resource allocation scheme. Simulation results

with different network size and training parameters are presented to show the scalability and robustness of the proposed algorithm.

1.5 Organization of the Thesis

The remainder of the thesis is organized into three chapters as follows:

- Chapter 2 discusses the basic concepts of a DNN, different deep learning architectures and the data training procedure. Then the resource allocation problem of a multi-cell network having multiple users sharing the same frequency sub-bands is formulated. To this end, a brief a supervised DL model to compute optimal sub-band and power allocation solutions for a multi-cell network is proposed. Detailed system model, training data generation process, training procedure, training parameters and simulation results are also provided.

- Chapter 3 discusses the limitations of the supervised DL approach for resource allocation in wireless networks. Then a centralized DRL-based resource allocation framework for multi-cell networks is presented. Detailed system model, online training procedure and simulation results are also presented.

- Chapter 4 provides a summary of the research presented in this thesis along with the future research directions.

Symbols and notations used throughout the chapters are given in a table at the beginning of each chapter.

Chapter 2

Supervised Deep Learning for Radio Resource Allocation in Multi-Cell Networks

In this chapter, I provide an introduction to the deep learning, the deep learning architectures, and the training procedure. Therein, I specifically focus on *Auto-encoder* which belongs to the unsupervised learning category of DL architectures. To this end, I formulate the resource allocation problem of a multi-cell network having multiple users sharing the same frequency subbands. Then I develop a supervised DL model to compute optimal subband and power allocation solutions for a multi-cell network with an objective to maximizing the total network throughput. More specifically, I use the *Auto-encoder* approach to build a DNN and to pre-train the proposed DL based resource allocation model. Training data generation is the most crucial part of the proposed resource allocation method. I use a genetic algorithm (GA) to generate near-optimal training data for the proposed resource allocation model.

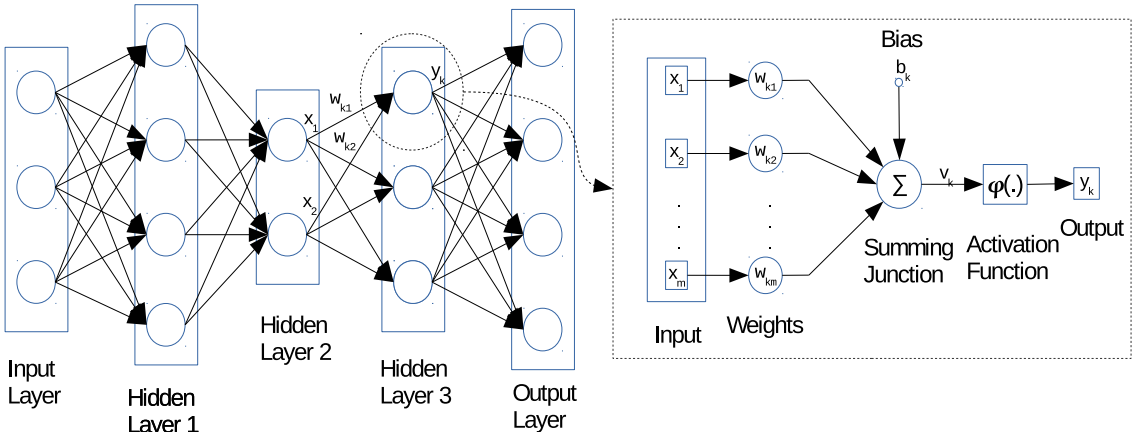


Figure 2.1: A deep neural network with three hidden layers.

2.1 Fundamentals of Deep Learning

DL is a specific methodology to train and build neural networks. A basic DL model is a multi-layered feed-forward neural network that enables feature extraction and transformation of input data. I briefly review the basic composition of deep neural networks (DNN), relevant DNN architectures and data training procedure with feed-forward and backpropagation DNN.

2.1.1 Deep Neural Network (DNN)

A DNN consists of an input layer, multiple hidden layers, and the output layer (Fig. 2.1). Each layer has multiple units commonly known as neurons. Each neuron performs a non-linear operation on the inputs. A weighted summation of the inputs is first computed, and then before sending the weighted summation to an activation function (e.g. a Sigmoid function $\varphi(x) = \frac{1}{1+e^{-x}}$), a bias value is added. The activation function creates a non-linear relationship between the input and the output [2]. The non-linear representation of a neuron is shown in Fig. 2.1. Every neuron has a vector

of weights and a bias value. Therefore, for a layer, there will be a weight vector and a bias vector. If $\mathbf{x}$ is the input of hidden layer i, then the output of the hidden layer i is $\mathbf{h}^{(i)} = \varphi^{(i)}(\mathbf{w}^{(i)T}\mathbf{x} + \mathbf{b}^{(i)})$, where $\varphi^{(i)}$ is the activation function, $\mathbf{w}^{(i)}$ is the weight vector, and $\mathbf{b}^{(i)}$ is the bias vector of hidden layer i.

2.1.2 Deep Learning Architectures

DL relies on four primary neural network architectures: unsupervised pre-trained networks (UPNs), convolutional neural networks (CNNs), recurrent neural networks (RNN), and recursive neural networks. The DL architectures can be used along with three learning models: supervised, unsupervised, and reinforcement learning. In supervised DL, a supervisor helps the model to learn the features from data, i.e. the model already knows the output of the algorithm before it starts learning it. Therefore, in supervised learning, the dataset contains the target for each input. Supervised learning is used in classification and regression problems. In unsupervised learning, there is no supervisor to provide correct answers. Therefore, in unsupervised learning, the dataset does not have any target. And the primary goal is to correctly infer the outputs for a given set of unlabeled input data. In reinforcement learning (a popular example of which is *Q-learning* [23]), there is a software agent who learns by interacting with the environment. The agent senses its current state and the state of the environment and then choose an action. For every action it takes, there is a consequence. The agent either receives a reward for a good move or a penalty for a wrong move. The primary job of the agent is to maximize the cumulative reward through a series of actions. When a neural network is used as an agent, it is referred to as deep reinforcement learning (DRL).

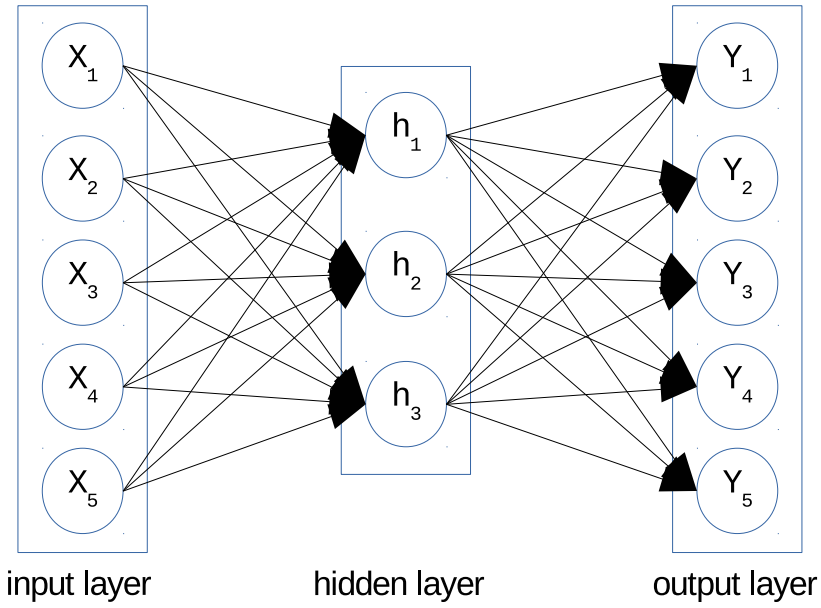


Figure 2.2: Structure of an Auto-encoder.

Auto-encoder: In this chapter, I focus on *Auto-encoder* (AE) which falls under the category of UPNs. An AE is an unsupervised learning model [24] which reconstructs the inputs at the outputs. By doing so, AE learns the feature space of the data set. It normally consists of an input layer with one or more hidden layers and an output layer. The number of neurons in the input layer and the output layer should always be the same. A simple AE is shown in Fig. 2.2.

An AE consists of two parts: an encoder which transforms the input to a hidden code and a decoder which reconstructs the input from the hidden code. Auto-encoders and Principle Component Analysis (PCA) [25] are closely related as both of them try to minimize the same objective function. The only difference is that AE can perform both linear and non-linear transformation, but PCA can perform the only linear transformation. AE is also used for reducing the dimensionality of data. There are various kinds of AE available like denoising AE, stacked AE, sparse AE, and

variational AE.

Sparse Auto-Encoder: A Sparse Auto-Encoder is an extension of AE which can learn sparse features of the data set. By adding a sparse penalty into the traditional AE, the performance of the AE can be significantly improved [26]. In a traditional AE, the number of hidden units is small compared to the inputs. An AE can still learn useful features even if the number of hidden units is large. This can be achieved by adding a sparsity constraint.

A neuron is active if the output is 1 or close to 1 and inactive if the output is close to 0. I want the neurons to be inactive for most of the time. Let, $a_j^{(2)}(x)$ is the activation of neuron j of layer 2 or the hidden layer when x is given as an input. So, the average activation of neuron j is

$$\overline{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(2)}(x^{(i)})\right]. \tag{2.1}$$

A low activation value means a neuron in the hidden layer activates for a small number of training examples, i.e. a neuron responds to a particular feature of the training examples. We add a constraint $\overline{\rho}_j = \rho$ to make the average activation of each hidden neuron j to be close to zero. Here, ρ is called the sparsity parameter.

A penalty term known as sparsity regularization is used in the optimization objective which will penalize $\overline{\rho}_j$ for deviating from ρ. The sparsity regularization $\Omega_{sparsity}$ defined by the Kullback-Leibler (KL) divergence [27] is

$$\begin{aligned}\Omega_{sparsity} &= \sum_{j=1}^{s_2} KL(\rho\|\overline{\rho}_j) \\ &= \sum_{j=1}^{s_2}\left(\rho\log\frac{\rho}{\overline{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\overline{\rho}_j}\right).\end{aligned} \tag{2.2}$$

Here, s_2 is the number of neurons in the layer-2 or hidden layer. $KL(\rho\|\overline{\rho}_j)$ is the KL divergence between the Bernoulli random variables of mean ρ and $\overline{\rho}_j$. The sparsity regularization has the following properties:

- $KL(\rho\|\overline{\rho}_j) = 0$ if $\overline{\rho}_j = \rho$,
- otherwise, it increases monotonically as $\overline{\rho}_j$ diverges from ρ.

On the training process of a sparse auto-encoder, it is possible to make the sparsity regulariser small by increasing the weights $W^{(1)}$ of the first layer and decreasing the values of the hidden code $h^{(1)}$ [28]. A regularization term on the weights to the cost function prevents it from happening. This is known as L_2 regularization defined as,

$$\Omega_{weights} = \frac{1}{2}\sum_{l}^{L}\sum_{j}^{n}\sum_{i}^{k}\left(w_{ji}^{(l)}\right)^2 \tag{2.3}$$

where L is number of hidden layers, n is the number of observations and k is the number of features in the training data set. The overall cost function or the objective function of the sparse auto-encoder is

$$J_{sparse}(W,b) = J(W,b) + \lambda * \Omega_{weights} + \beta * \Omega_{sparsity} \tag{2.4}$$

where $J(W,b)$ is the cost function of a traditional AE. λ is the coefficient of L_2 regularization term and β is the coefficient of sparsity regularization term.

Stacked Auto-Encoder (SAE): A Stacked Auto-Encoder is a neural network which is consist of multiple layers of sparse AE. The output of each layer is connected to the inputs of the successive layer. Usually, greedy layer-wise training is used for stacked auto-encoder. We first train the first layer of the SAE with the inputs to get

the weights and biases of that layer. We use those weights and biases to generate hidden codes of the first layer. These hidden codes are then used to train the second layer. The output of one layer is used as an input to train the subsequent layer.

2.1.3 Data Training Procedure

Let us assume that I have to learn a target function $\mathbf{y} = f^*(\mathbf{x})$. Here, $\mathbf{x}$ is the input vector and $\mathbf{y}$ is the output vector. The DL model is therefore $\mathbf{y} = f(\mathbf{x}; \theta)$, where θ denotes the unknown parameters, i.e. weights and biases. The goal is to learn θ precisely so that the model can be closer to the original one. Different algorithms (e.g. Stochastic Gradient Descent (SGD) [29], Momentum [30], Nesterov Momentum [31], AdaGrad [32], RMSProp [33], and Adam [34]) can be used to learn θ.

To learn θ, a training labeled dataset composed of inputs and outputs is typically used to train the model. In the training phase, I first initialize the weights and biases randomly. Then, I feed those inputs to the input layer of the system. The output of the input layer is used as an input for the hidden layer. In this way, the data propagates through the hidden layers and finally reaches the output layer. The propagation of information from the input layer to the output layer is known as the *forward pass*. A cost function (e.g. mean squared error [MSE]) is used to measure the quality of the model by calculating the error between the predicted and the original value. The resulting error signal is then propagated backward through the hidden layers and updates the weights and biases of each layer which is known as the *backward pass*. This training process continues until the error rate reaches a threshold value. When training data completes a forward and backward pass, it is called a complete training cycle or epoch. The training process can also be terminated if it reaches the maximum number of epochs. Such a neural network is referred to as a *feed-forward*

and back-propagation neural network and is used in the proposed DL model.

Table 2.1: Symbols

Symbol	**Description**
K	Number of base stations (BSs)
F	Number of frequency sub-bands
B	Bandwidth of each sub-band in MHz
$P_{k,f}$	Power allocated by cell k in frequency sub-band f
$P_k^{\max}$	Maximum power of a cell k
$\mathcal{U}_k$	Set of users associated with cell k
$\mathcal{U}$	Set of all users
$\mathcal{A}_k$	Allocation vector of cell k
$A_{k,f}$	The user assigned to sub-band f in cell k
$\mathbb{I}(.)$	Indicator function
$\text{SINR}_{u,k,f}$	SINR of user u in cell k over frequency sub-band f
η_u	Receiver noise
$G_{u,k,f}$	Link gain from cell k to user u over frequency sub-band f
$H_{u,k,f}$	Rayleigh fading gain of user u from cell k over frequency sub-band f
X_α	Log-normal shadowing
PL_u	Path-loss of user u
$\mathcal{C}_{u,k}$	CQI vector of a user u of cell k
$\mathcal{V}_{u,k}$	Location indicator of user u of cell k
$R_{u,k}$	Distance of user u in cell k from the BS
R	Cell radius

2.2 Sub-band and Power Allocation in Multi-cell Networks: A Supervised Deep Learning Approach

In the following, I develop a DL-based resource allocation model with an objective to maximizing the total network throughput by performing joint resource allocation (i.e. both power and channel). I use a supervised learning approach to train the DNN model and obtain the training data by solving the non-convex optimization problem through a genetic algorithm. I also use the stacked auto-encoder approach to pre-train the model. I observe that a pre-trained model converges more quickly

compared to an untrained model and performs better.

2.2.1 Network Model

In order to increase the readability, all the symbols that are used throughout this chapter are listed in Table 2.1.

I consider a downlink cellular network of K base stations (BSs). Each BS $k \in \{1, \cdots, K\}$ has F frequency sub-bands. The bandwidth of each sub-band is B MHz. The power allocated by cell k in frequency sub-band f is $P_{k,f}$ which is discrete. The total power of a cell k is limited by a maximum value $P_k^{\max}$ such that $\sum_{f \in F} P_{k,f} \leq P_k^{\max}, \quad \forall k \in \{1, \cdots, K\}$. Let $\mathcal{U}_k$ denote the set of users who are associated with cell k, and $\mathcal{U}$ is the set of all users in the network. The vector $\mathcal{A}_k$ denotes allocation of sub-bands in cell k, where each element $A_{k,f}$ is an integer denoting the user who is assigned sub-band f in cell k.

The corresponding throughput maximization problem is given by

$$\max \sum_{k \in \{1,\cdots,K\}} \sum_{u \in \mathcal{U}_k} \sum_{f=1}^{F} [\mathbb{I}(A_{k,f} = u) B \log(1 + \alpha \text{SINR}_{u,k,f})] \tag{2.5}$$

$$\text{s.t.} \sum_{f \in F} P_{k,f} \leq P_k^{\max}, \quad \forall k \in \{1, \cdots, K\} \tag{2.6}$$

where $\mathbb{I}(.)$ is an indicator function, α is a constant for a given target Bit Error Rate (BER) which is defined as $\alpha = -1.5/\log(5\text{BER})$. We assume BER to be 10^{-6}. The signal-to-interference-plus-noise ratio (SINR) of user u when served by cell k which transmits over frequency sub-band f is expressed as $\text{SINR}_{u,k,f} = \frac{P_{k,f} G_{u,k,f}}{\eta_u + \sum_{l \neq k} P_{l,f} G_{u,l,f}}$. where η_u represents the receiver noise and $G_{u,k,f}$ denotes the link gain from cell k to user u over frequency sub-band f defined as $G_{u,k,f} = 10^{-(PL_u + X_\alpha)/10}.|H_{u,k,f}|^2$, where $H_{u,k,f}$ is the Rayleigh fading gain of user u from cell k over frequency sub-band f,

X_α is the log-normal shadowing, and PL_u is the path-loss of user u.

The utility of the network which is the total network throughput is defined as follows:

$$U = \sum_{k\in\{1,\cdots,K\}} \sum_{u\in\mathcal{U}_k} \sum_{f=1}^{F} \left[\mathbb{I}(A_{k,f} = u) B \log\left(1 + \alpha \mathrm{SINR}_{u,k,f}\right)\right]. \tag{2.7}$$

All users periodically send their channel quality as a channel quality indicator (CQI) to their nearest BS, where $\mathcal{C}_{u,k}$ is the CQI vector of a user u of cell k over all frequency sub-bands. That is, $\mathcal{C}_{u,k}$ is a vector of discrete values $\mathcal{C}_{u,k} \coloneqq (\mathcal{C}_{u,k,1}, \mathcal{C}_{u,k,2}, \ldots, \mathcal{C}_{u,k,F})$. In LTE, the CQI values range from 1 to 15 [35]. In addition, users are also classified into cell-center users and cell-edge users depending on the users' locations. A location indicator $\mathcal{V}_{u,k}$ is used to indicate whether a user u of cell k is cell-center user or cell-edge user as follows:

$$\mathcal{V}_{u,k} = \begin{cases} 1 & \text{if } R_{u,k} > R/2 \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

where $R_{u,k}$ is the distance of user u in cell k from the BS and R is the cell radius. For each user class, sub-band and power allocations will be learned and predicted separately.

2.2.2 Supervised Deep Learning Approach

Now I present a deep learning approach that can predict optimal sub-band and power allocation solutions for multi-cell networks with a certain accuracy. Specifically, this deep learning model takes $\mathcal{C}_{u,k}$ vector along with $\mathcal{V}_{u,k}$ of all users in a network as input and predict the power and sub-band allocations as output. That is, for K cells, U users and F sub-bands, the size of the input data is $(K \times U \times (F + 1))$. I convert

the output data from decimal base to n-bit binary and also their complement in the output. Therefore, for K cells, the size of output data is $(2 \times 2 \times n \times K \times F)$. The approach proceeds in the following phases:

Data generation: It requires the optimal solution of the sub-band and power allocation problem which is a non-convex combinatorial optimization problem. One way is to check all possible combinations of power and channel allocation for all the BSs which is referred to as *exhaustive search*. For example, with 15 cells, 5 sub-bands, and 5 discrete power levels, then there will be 5^5 or 3125 possible combinations available for the power setting of one BS. Therefore, I will need to check 3125^{15} combinations, which is practically infeasible. As such, to generate data to train and test our DNN model, I resort to a GA, which is a heuristic searching algorithm inspired by the theory of natural evolution [36]. That is, first I generate a set of randomly initialized individuals and calculate a fitness score for every individual. Two individuals are then selected based on their fitness scores and then a randomly generated crossover point is used to exchange their genes resulting in a new offspring. The new offspring is then added to the population. Some genes of the new offspring can be mutated to maintain diversity within the population. GA continuously generates new offsprings until the population becomes converged, i.e. a new offspring is not significantly improved from the previous generation. Then I have a set of solutions for the problem.

Data generation algorithm: The data generation using GA is shown in **Algorithm 1**.

Step 1: Allocate a random power vector for each cell. After that, I calculate the CQI value of each sub-band for each user in the network. I also estimate the location indicator for every user using Eq. 3.8. Note that the CQI values and the location

Algorithm 1 Data generation using GA

1: Initialize the dataset D to capacity N
2: **for** sample $= 1, \cdots, N$ **do**
3: Allocate a random power vector for each cell.
4: Calculate the CQI vector as well as the location indicator for every user in the network.
5: Label the CQI vector and the location indicator as input.
6: Use GA to find the optimal power vector of every BS which maximizes the total network utility (Eq. 2.7).
7: Allocate random power vectors for every cell to generate *initial population* of GA.
8: Calculate the total network utility (Eq. 2.7) as a *fitness score* for every individual.
9: **repeat**
10: Choose two sets of individuals of high fitness score from the population.
11: Choose a random crossover point for each pair and exchange the genes (i.e. power levels of the power vector) among them.
12: Choose one or more genes of the new offspring with a very low mutation probability and change their values.
13: Check the fitness score of the new offspring. If fitness score improved from their parents, put them on the population.
14: **until** population has converged.
15: Calculate the sub-band allocation from the optimal power vector.
16: Label the power vector and allocation vector as output.
17: Store the $(\text{input}, \text{output})$ pair in D.
18: **end for**

indicator for every user are the inputs of our DNN model.

Step 2: Find the power vector of every BS which maximizes the total network utility (Eq. 2.7). I use GA to solve this problem as explained in the following.

Step 2.1: Generate random power vectors for every cell. This is known as the *initial population* of GA.

Step 2.2: Using the power vectors, I compute the total network utility (Eq. 2.7), which refers to a *fitness function* in a GA. Therefore, for every individual, I have a fitness score.

Step 2.3: I choose two sets of individuals of high fitness score from the initial

population.

Step 2.4: I then choose a random crossover point for each pair and exchange the genes among them. Here, genes are the different power levels of the power vector.

Step 2.5: With a very low mutation probability, I randomly choose one or more genes of the new offspring and change their values.

Step 2.6: I check the fitness score of the new offspring. If the fitness score improved from their parents, I would put them on the population. I keep repeating **Steps 2.2-2.6** until there is no significant improvement in the fitness scores.

Step 3: Once I find the optimal power vector, I need to calculate the sub-band allocation. A sub-band will be allocated to that user who will maximize the throughput. The power vector and allocation vector are the outputs.

Step 4: I need to repeat **Steps 1-4** until I have a certain amount of data to train our DNN model.

Once data generation is complete, I train the proposed model with the data. Finally, I use the pre-trained DNN model to predict sub-band and power allocations for every BS in the network. This prediction will be performed online.

Training phase: The training labeled dataset consists of labeled input data and labeled target data or the output data. I use a stacked AE to initialize the weights and biases for the neurons in the hidden layers. In this way, I pre-train the proposed model, and then I add a Softmax layer, which assigns probabilities to each class in such a way that the total probability must be 1.0. Finally, I stack the encoder part of the AE with the Softmax layer and fine-tune the model with the training labeled dataset. The complete DNN is shown in Fig. 2.3.

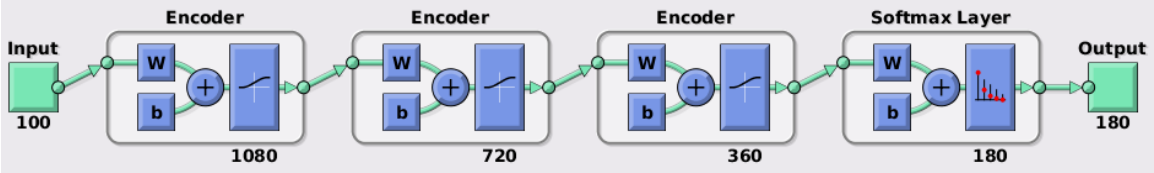


Figure 2.3: DNN model for power and sub-band allocation.

Testing phase: After training the model, I test the model on a new labeled dataset and calculate the accuracy of the model. In a practical setting, all users in the network periodically send their CQI values to their serving BSs, which extract the CQI value of each sub-band and add a location indicator, i.e. cell-center user or cell-edge user. Therefore, for every user, there will be a vector of CQI and location indicator. Each BS then sends the processed information of all users to a central entity (e.g. SDN controller), which runs the DNN model. The DNN model will generate the allocation vector and power vector for all the BSs. Once the prediction is made, the controller will send back the allocation and power vectors to their designated BSs.

2.3 Simulations and Results

I consider a network of $K = 5$ BSs with a coverage radius $R = 500$ m, maximum transmit power = 40 W, directional antennas per cell = 3, number of users per cell = 5, bandwidth of a sub-band $B = 2.88$ MHz, white noise power density = -174 dBm/Hz, number of sub-bands = 3, number of power levels = 3, and power levels = $\{6.4, 12.8, 19.2\}$ W.

2.3.1 Input Data Generation

First, I generate data for the training of DNN model using the GA approach. To confirm the accuracy of the solutions obtained from GA, I compare GA with the

Table 2.2: Performance comparison between exhaustive search and genetic algorithm

Parameter	**Exhaustive Search**	**Genetic Algorithm**
Avg. Execution time	1460.00 sec.	118.76 sec.
Max. Execution time	2247.50 sec.	158.65 sec.
Min. Execution time	1509.00 sec.	95.13 sec.
Accuracy	100%	85.25%

exhaustive search. For this, I first simulate a network of 4 macro-cells where each cell has 5 users. Then using **Step 1**, I calculate the CQI values and the location indicator of each user in the network. For the exhaustive search, instead of **Steps 2 and 3**, I perform the exhaustive search to maximize the total network utility, i.e. the optimal powers. We repeat **Steps 1-4** to generate 1000 samples for comparison between the exhaustive search and GA. The comparisons are shown in Table 2.2.

An exhaustive search takes much longer time compared to GA. This is because, in an exhaustive search, I need to check the whole solution space whereas GA takes a much shorter time and generate optimal solutions with a probability of 0.85 (i.e. prediction accuracy is 85%). For a large scale system, it is not feasible to generate data using the exhaustive search. Therefore, I use GA to create training and testing data for the proposed DNN model. I use a network of 5 BSs where the cells are partially overlapped, and five users are located randomly within each macro-cell. I apply 3 frequency sub-bands and 3 power levels for each macro-cell. By using **Steps 1-4**, I produce around 17000 samples for the DNN model. Then I use 80% of this labeled dataset for training and the remaining 20% for testing purposes.

2.3.2 Training the DNN

After data generation, I train our DNN model with our training labeled dataset. The training labeled dataset has two parts: input and output. The input data size is

$5 \times 5 \times (3+1) = 100$ and for the output data, I convert the data from decimal base to 3-bits binary base. I also use their complements in the output, i.e. if the binary representation 3 is $(011)_b$, then its complement is $(100)_b$. Then, the output data size becomes $2 \times 2 \times 3 \times 5 \times 3 = 180$. I use a stacked AE model for pre-training as well as to build our DNN model. I first train an AE with our input data. The training parameters are shown in Table 2.3.

Table 2.3: Training parameters

Parameter	**Value**
Number of Hidden layers	3
Layers	{Input, Encoder, Encoder, Encoder, Softmax}
No. of neurons per layer	{100, 1080, 720, 360, 180}
Max. no. of epochs (pre-training) (AE1, AE2, AE3, Softmax)	(1000, 500, 500, 1000)
Max. no. of epochs (fine-training)	1000
Encoders transfer function	sigmoid
SparsityProportion, ρ (AE1, AE2, AE3)	(0.15, 0.1, 0.1)
L2WeightRegularization, λ (AE1, AE2, AE3)	(0.004, 0.002, 0.002)
SparsityRegularization, β (AE1, AE2, AE3)	(4, 4, 4)

Then I generate hidden codes of the first AE by using the input of the training data as an input of the AE. Then these hidden codes are used to train the second auto-encoder. Using this approach, we pre-train several AEs, and at the end, I add a Softmax layer. To pre-train the Softmax layer, I use the hidden codes of the last AE as the input and output part of the training labeled dataset as target or output.

2.3.3 Results and Discussions

Impact of the number of hidden layers on prediction accuracy: After pre-training, I stack all encoders of the AE along with the Softmax layer and fine-tune network with the training labeled dataset. Once training is completed, I test the accuracy of our model using the testing labeled dataset. To find the optimal number of hidden layers for the proposed model, I vary the number of hidden layers, i.e. the number of stacked auto-encoders and calculate test accuracy. Fig. 2.4 shows the test accuracy of our DNN model vs. the number of hidden layers. From this figure, I can see that at first the accuracy increases with the number of hidden layers and then it starts to decrease. More hidden layers in a neural network mean it can learn more features. As the number of hidden layer increases, the model also starts to learn the irrelevant features (noise) of the training data. Then the model is not able to perform well on the new examples and the test accuracy deteriorates. The model overfits the training data and this situation is commonly known as *overfitting*.

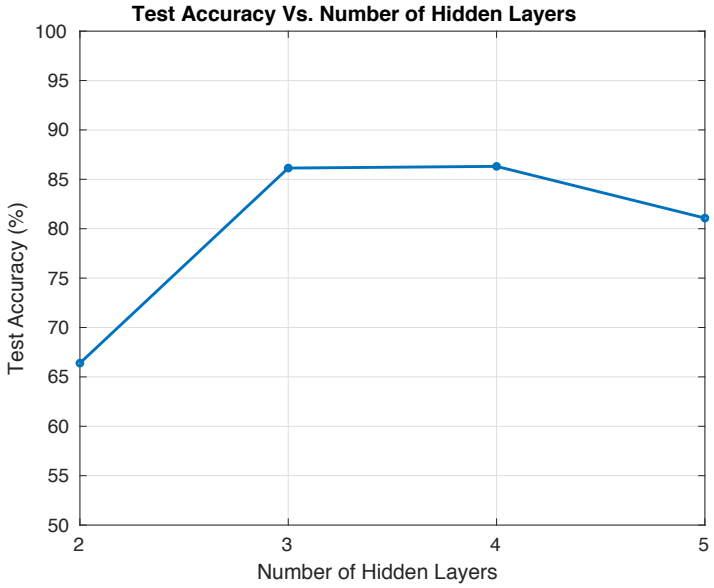


Figure 2.4: Test accuracy vs. number of hidden layers.

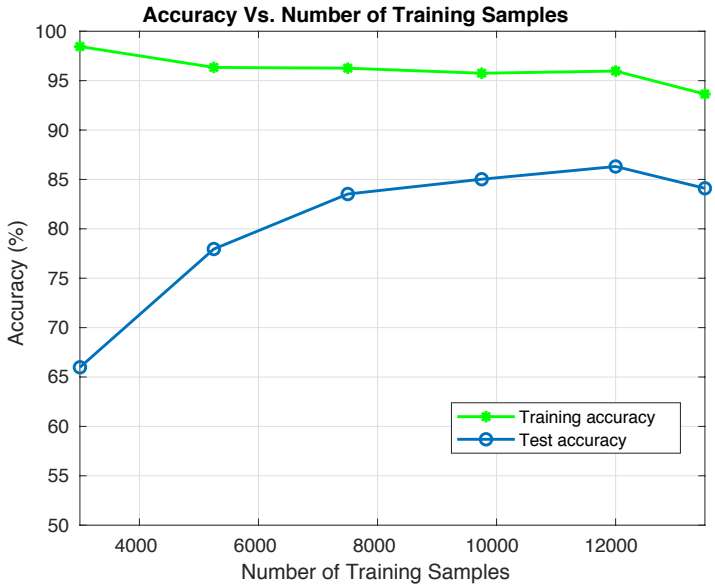


Figure 2.5: Test accuracy vs. number of samples.

I achieve the maximum test accuracy of 86.31% for 4 hidden layers and slightly less test accuracy of 86.14% for 3 hidden layers. The complete DNN model along with its number of neurons per layer is shown in Fig. 2.3. Next, I vary the number of training samples and observe its effects on both training accuracy and test accuracy. Note that training accuracy refers to the accuracy we observe when we apply the model to the training data.

Impact of the number of training samples on prediction accuracy: From Fig. 2.4, I observe that the test accuracy is low for a small number of training samples and the accuracy increases gradually with the training examples. On the other hand, the training accuracy is high for a small number, and the accuracy decreases slowly with the training examples. Initially, for a small number of training examples, the DNN learns very few distinguishing features and this is enough to express the input-output relationship of the training data. This is why the training accuracy is high for

a few training samples. However, for testing, the learned features are not sufficient enough to predict the output correctly. With the increase of training samples, the DNN model extracts more abstract features from the data, i.e. the model learns more about the relationship between the inputs and outputs. The testing accuracy keeps increasing with the training examples, and after some specific training examples, it starts to saturate. The reason is that as we continue to increase the training samples size there are no new distinguishing features to be learned and thus the prediction accuracy may not increase any further. However, the training accuracy decreases because the model also learns the common features (noise) of the data which ultimately degrades the performance of the model. Eventually, with a further increase in training data, the model starts to overfit the data leading to reduced test accuracy.

2.4 Summary

I have presented a deep supervised learning approach to solve the sub-band and power allocation problem in multi-cell networks. Simulation results show that the prediction accuracy increases with the size of data samples and the number of hidden layers. However, a continuous increase of the number of hidden layers will not improve the accuracy significantly and in some cases, the model may even start to learn noisy features. Obtaining the optimal configuration of the DL model, e.g. number of hidden layers and the size of input data samples is a fundamental challenge. Also, it is crucial to develop efficient offline methods to generate optimal/near-optimal data samples for training the DNN architectures. Application of deep learning in scenarios with massive connections and spatiotemporal correlations due to user mobility and variations in wireless channel fading characteristics would be of immediate relevance.

Chapter 3

A Deep Q-Learning Method for Downlink Power Allocation in Multi-Cell Networks

3.1 Introduction

3.1.1 Overview

Resource allocation using supervised DL approach is not feasible for large-scale wireless systems because the heuristic algorithms (e.g. GA, sequential fractional programming (SFP) [37], bisection approach [38], etc.) used to generate training data are computationally expensive and time-consuming. Therefore, this approach is not suitable for large-scale systems. On the other hand, RL such as Q-learning [39] is already applied for cognitive radio applications [40–43]. However, the RL is can only work on the low dimensional case and therefore not suitable for resource allocation in large-scale systems. With the advancement of RL, DRL can extract useful in-

formation from the high dimensional data such as image and can learn the optimal action policy [44]. Power allocation under maximal power constraints in a multi-cell network (e.g. a cloud-RAN) with an objective to maximize the total network throughput is a well-known non-convex combinatorial optimization problem and is NP-hard [45]. Model-based algorithms such as Fractional Programming (FP) [46] and Weighted Minimum Mean Squared Error (WMMSE) [47] are usually used in this scenario. But, both the algorithms formulate the power allocation problem as a convex optimization problem.

3.1.2 Contribution

The contributions of this chapter can be summarized as follows:

- First, I provide a brief overview of DRL, DRL architectures and the training procedure.
- Then, I propose a centralized downlink power allocation scheme based on DRL for multi-cell network to maximize the total network throughput. The proposed scheme is novel and it is one of the first such schemes to address the power allocation problem under maximal power constraints in a multi-cell network having multiple users sharing the same frequency subbands.
- I define the state space, action space and the reward function for the DRL agent. I also define the online training procedure of the proposed DRL based power allocation scheme.
- Unlike supervised learning approach, there is no need for optimal/ near-optimal training data. This is why the proposed power allocation scheme is computationally scalable to large-scale scenarios.

- Simulation results with different network size and training parameters are presented to show the scalability and robustness of the algorithm. I also compare our DRL model with the near-optimal solution derived through a GA. Simulation results show that the proposed DRL based resource allocation model can perform well in a large-scale scenario.

3.2 Deep Reinforcement Learning (DRL)

In reinforcement learning, there is a software agent who learns by interacting with the environment. The agent senses its current state and the state of the environment and then choose an action. For every action it takes, there is a consequence. The agent either receives a reward for a good move or a penalty for a bad move. The primary job of the agent is to maximize the cumulative reward through a series of actions. When a DNN is used as an agent, it is referred to as DRL. Therefore, DRL is the combination of DNN and RL as shown in Fig. 3.1.

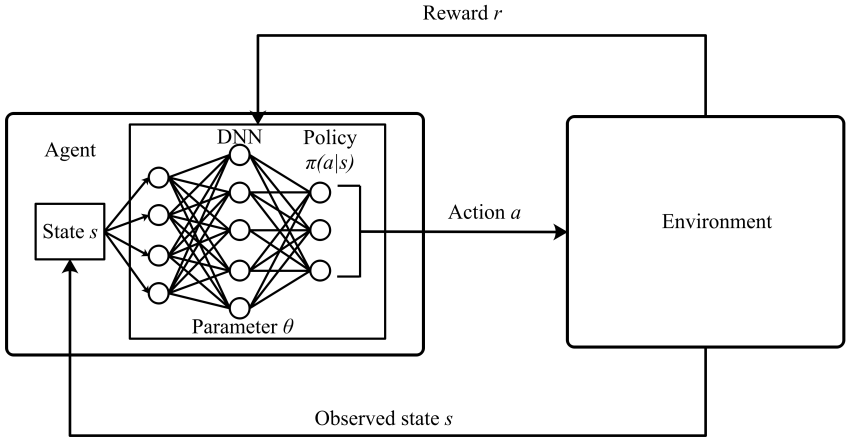


Figure 3.1: Deep reinforcement learning.

Table 3.1: Symbols

Symbol	**Description**
s_t	State at time step t
a_t	Action at time step t
A	Action space
r_t	Reward at time step t
R_t	Accumulated reward
γ	Discount factor
Q	Action-value function
$\hat{Q}$	Target action-value function
y	Approximate target values
$L_i(\theta_i)$	Loss function at each iteration i
$\mathbb{V}_{s'}[y]$	Variance of the target
λ	Speed of decay
τ	Total number of steps elapsed
K	Number of base stations (BSs)
F	Number of frequency sub-bands
B	Bandwidth of each sub-band in MHz
$P_{k,f}$	Power allocated by cell k in frequency sub-band f
$P_k^{\max}$	Maximum power of a cell k
$\mathcal{U}_k$	Set of users associated with cell k
$\mathcal{U}$	Set of all users
$\mathcal{A}_k$	Allocation vector of cell k
$A_{k,f}$	The user assigned to sub-band f in cell k
$\mathbb{I}(.)$	Indicator function
$\text{SINR}_{u,k,f}$	SINR of user u in cell k over frequency sub-band f
η_u	Receiver noise
$G_{u,k,f}$	Link gain from cell k to user u over frequency sub-band f
$H_{u,k,f}$	Rayleigh fading gain of user u from cell k over frequency sub-band f
X_α	Log-normal shadowing
PL_u	Path-loss of user u
$\mathcal{C}_{u,k}$	CQI vector of a user u of cell k
$\mathcal{V}_{u,k}$	Location indicator of user u of cell k
$R_{u,k}$	Distance of user u in cell k from the BS
R	Cell radius
$\mathbb{A}_k$	Action space for cell k
$\mathfrak{a}_k \in \mathbb{A}_k$	Selected action for cell k

3.2.1 Deep Reinforcement Learning Architectures

In order to increase the readability, all the symbols that are used throughout this chapter are gathered in Table 3.1.

The deep reinforcement learning (DRL) algorithms can be categorized into three categories [48]: value-based, policy-based, and actor-critic methods. In value-based DRL, the agent learns the optimal action from the action-state value function. Deep Q-learning (DQL) and SARSA [49] are the most popular in value-based DRL methods. In a policy-based method such as *REINFORCE* [49] learning, the agent optimize the policy directly. In actor-critic methods, the critic updates the action-value function and the actor update the policy. Deep deterministic policy gradient (DDPG) [22] algorithm is actor-critic based.

In this chapter, I focus on the deep Q-learning (DQL) which is a value-based DRL. At time step t, the DQL agent receives state s_t from a state space $\mathcal{S}$ and takes an action a_t from an action space A. The agent follows a policy $\pi(a_t|s_t)$ i.e., a mapping from state s_t to action a_t, to choose the action. After executing action a_t, the agent receives a reward r_t and move to new state s_{t+1}. The agent continues the process until it reaches the terminal state and then it restarts. The goal of the agent is to maximize the discounted accumulated reward defined as $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$. Here, $\gamma \in (0, 1]$ is the discount factor which determines the importance of future rewards compared to current reward. An action-value function $Q_\pi(s, a) = E[R_t|s_t = s, a_t = a]$ is the expected return for selecting action a in state s and then follow a policy π. An optimal action-value function $Q^*(s, a) = \max_\pi Q_\pi(s, a)$ is the maximum action value achievable by following any policy for state s and action a. The optimal action-value function can be decomposed into Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s^{'}, a^{'})|s, a]. \tag{3.1}$$

In DQL, a neural network is used to approximate the optimal action-value function, $Q(s,a;\theta) \approx Q^*(s,a)$. Here, $Q(s,a;\theta)$ is called the Q-network and θ is the parameter of neural network. Iterative update is used to train the Q-network and thus reduce the mean-squared error of the Bellman equation. For that, the optimal target values $r + \gamma \max_{a'} Q^*(s^{'}, a^{'})$ are replaced with approximate target values $y = r + \gamma \max_{a'} Q(s^{'}, a^{'}; \theta_i^{-})$, where θ_i^{-} are the parameters of the Q-network from some previous iteration. The loss function $L_i(\theta_i)$ at each iteration i is

$$\begin{aligned} L_i(\theta_i) &= \mathbb{E}_{s,a,r}[(E_{s'}[y|s,a] - Q(s,a;\theta_i))^2] \\ &= \mathbb{E}_{s,a,r,s'}[(y - Q(s,a;\theta_i))^2] + \mathbb{E}_{s,a,r}[\mathbb{V}_{s'}[y]]. \end{aligned} \tag{3.2}$$

Here, $\mathbb{V}_{s'}[y]$ is the variance of the targets which is independent of θ_i and therefore can be ignored. The parameters from the previous iteration θ_i^{-} are kept fixed when optimizing i-th loss function $L_i(\theta_i)$. Gradient Descent algorithm is used to optimize the loss function.

3.2.2 Training Procedure of DQL

Training procedure of deep Q-network is shown in **Algorithm 2**. Two separate Q-network is used for action-value function Q and target action-value function $\hat{Q}$. The target Q-network $\hat{Q}$ is used to generate targets y_j in the training process. The agent either with probability ε selects a random action or selects the action of maximum action-value. This method is known as ε-greedy policy. Initially, the agent starts with a high ε value and then, slowly it decreases the value based on the experience.

Algorithm 2 Deep Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
for episode $= 1$, M **do**
  for $t = 1, \cdots, \infty$ **do**
    With probability ε select a random action a_t
    Otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = r_j$ if if episode terminates at step $j + 1$
    Otherwise set $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a^{'}; \theta^{-})$
    Perform gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every B steps reset $\hat{Q} = Q$
  end for
end for

The agent uses the following equation to decrease the value of ε

$$\varepsilon = \varepsilon_{\min} + (\varepsilon_{\max} - \varepsilon_{\min})e^{-\lambda\tau}. \tag{3.3}$$

Here, λ is the speed of decay and τ is the total number of steps elapsed. Experience Replay [50] is a mechanism where the agent learns from the previous experiences. After executing an action at time step t, the agent stores the experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ in a data set $D_t = \{e_1, e_2, \cdots, e_t\}$ which is also known as replay memory. After executing an action, the agent samples a random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) of size $\mathcal{M}$ from the replay memory D to train the Q-network. The samples are taken randomly to break the correlations between the consecutive samples and thus reduces the variance of the updates [7]. The targets y_j of the

minibatch are updated according the following equation:

$$y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}. \quad (3.4)$$

Then, gradient descent step is performed on the loss function $(y_j - Q(s_j, a_j; \theta))^2$ to update the network parameter θ. After every B steps, the network parameters of the Q network is copied to $\hat{Q}$ network and is used to generate targets y_j for the following B updates.

3.3 System Model

I consider a downlink cellular network of K base stations (BSs). Each BS $k \in \{1, \cdots, K\}$ has F frequency sub-bands. The bandwidth of each sub-band is B MHz. The power allocated by cell k in frequency sub-band f is $P_{k,f}$ which is discrete. The total power of a cell k is limited by a maximum value $P_k^{\max}$ such that $\sum_{f \in F} P_{k,f} \leq P_k^{\max}, \quad \forall k \in \{1, \cdots, K\}$. Let $\mathcal{U}_k$ denote the set of users who are associated with cell k, and $\mathcal{U}$ is the set of all users in the network. The vector $\mathcal{A}_{k,f}$ denotes allocation of sub-band in cell k, where each element $A_{k,f}$ is an integer denoting the user who is assigned sub-band f in cell k.

The corresponding throughput maximization problem is given by

$$\max \sum_{k \in \{1, \cdots, K\}} \sum_{u \in \mathcal{U}_k} \sum_{f=1}^{F} [\mathbb{I}(A_{k,f} = u) B \log(1 + \alpha \text{SINR}_{u,k,f})] \quad (3.5)$$

$$\text{s.t.} \sum_{f \in F} P_{k,f} \leq P_k^{\max}, \quad \forall k \in \{1, \cdots, K\} \quad (3.6)$$

where α is a constant for a given target Bit Error Rate (BER) which is defined as $\alpha = -1.5/\log(5\text{BER})$. We assume BER to be 10^{-6}. The signal-to-interference-plus-noise ratio (SINR) of user u when served by cell k which transmits over frequency sub-band f is expressed as $\text{SINR}_{u,k,f} = \frac{P_{k,f}G_{u,k,f}}{\eta_u + \sum_{l \neq k} P_{l,f}G_{u,l,f}}$. where η_u represents the receiver noise and $G_{u,k,f}$ denotes the link gain from cell k to user u over frequency sub-band f defined as $G_{u,k,f} = 10^{-(PL_u + X_\alpha)/10}.|H_{u,k,f}|^2$, where $H_{u,k,f}$ is the Rayleigh fading gain of user u from cell k over frequency sub-band f, X_α is the log-normal shadowing, and PL_u is the path-loss of user u.

The utility of the network which is the total network throughput is defined as follows:

$$U = \sum_{k \in \{1,\cdots,K\}} \sum_{u \in \mathcal{U}_k} \sum_{f=1}^{F} \left[\mathbb{I}(A_{k,f} = u) B \log\left(1 + \alpha \text{SINR}_{u,k,f}\right)\right]. \tag{3.7}$$

All users periodically send their channel quality as a channel quality indicator (CQI) to their nearest BS, where $\mathcal{C}_{u,k}$ is the CQI vector of a user u of cell k over all frequency sub-bands. That is, $\mathcal{C}_{u,k}$ is a vector of discrete values $\mathcal{C}_{u,k} \coloneqq (\mathcal{C}_{u,k,1}, \mathcal{C}_{u,k,2}, \ldots, \mathcal{C}_{u,k,F})$. For example, in LTE, the CQI value ranges from 1 to 15 [51]. In addition, users are also classified into cell-center users and cell-edge users depending on the users' locations. A location indicator $\mathcal{V}_{u,k}$ is used to indicate whether a user u of cell k is cell-center user or cell-edge user as follows:

$$\mathcal{V}_{u,k} = \begin{cases} 1 & \text{if } R_{u,k} > R/2 \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

where $R_{u,k}$ is the distance of user u in cell k from the BS and R is the cell radius. Subband f of cell k will be allocated to an user $u \in \mathcal{U}_k$ for which it will maximize

the throughput of subband f in cell k. Therefore, the subbands are allocated based on the following equation:

$$A_{k,f} = \arg\max_{u \in \mathcal{U}_k} B \log\left(1 + \alpha \mathrm{SINR}_{u,k,f}\right). \tag{3.9}$$

3.4 Power Allocation in Multi-cell Networks: A DRL Approach

In the following, I develop a DRL-based resource allocation model for multi-cell networks with multiple users per cell sharing the same frequency subbands to maximize the total network throughput by performing power allocation.

3.4.1 DQL Approach

Now I present a DQL approach that can perform near-optimal power allocation on multi-cell networks. Specifically, this DQL model uses $\mathcal{C}_{u,k}$ vector along with $\mathcal{V}_{u,k}$ of all users in a network as state and then takes action. Here, each action corresponds to a power allocation. That is, for K cells, U users and F sub-bands, the state size is $(K \times U \times (F+1))$. The total number of actions depends on the number of power levels we are using. For n number of power levels and F frequency sub-bands, I can have a maximum n^F power combinations. Some of the combinations will be discarded due to the maximal power constraint. Let m denote the total number of combinations possible where each combination corresponds to an action. For each cell, I have m number of actions. Let $\mathbb{A}_k$ denote the action space for k cell and $\mathfrak{a}_k \in \mathbb{A}_k$ is the selected action for cell k. Therefore, for K number of cells, I have $K \times m$ number of actions. The DQN model has to take action from m number of actions for each cell. Therefore, in total, the DQN model has to take K number of actions. At time step

t, the selected action $a_t = [\mathfrak{a}_1, \mathfrak{a}_2, \cdots, \mathfrak{a}_K]$. The approach proceeds in the following phases:

Problem formulation: First I need to formulate the problem to apply the DQL approach. The job of the agent is to maximize the total network throughput (Eq. (2.7)). The episode starts from a initial state and continues as long as the throughput increases, i.e. *current_throughput* > *previous_throughput*. Here, *current_throughput* the network throughput achieved by executing the recent actions and *previous_throughput* is due to the previous actions. The episode ends when it reaches the terminal state, i.e., the throughput decreases due to the recent actions.

Training Phase: The training process of the proposed DQL based power allocation is shown in **Algorithm 3**. I use DQL with experience replay [7] to train the proposed model. In the proposed model, the specific steps are as follows.

Step 1: Define the Q-Network, i.e. the number of layers and neurons per layer and the activation functions. I use the input layer size same as the state size and output layer size as the total number of actions. I initialize two Q-network with random weights: one is for the **action-value function** Q and another for **target action-value function** $\hat{Q}$. I also initialize the replay memory D to some capacity N.

Step 2: Allocate a random power vector for each cell. After that, I calculate the CQI value of each sub-band for each user in the network. I also estimate the location indicator for every user using Eq. (3.8). Note that the CQI values and the location indicator for every user represent the initial state s_t.

Step 3: Select the action which is consist of minimum power value possible for all the subbands for each cell. Then, I execute these actions and calculate the total network

Algorithm 3 DQL with experience replay for power allocation

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
for episode $= 1, \cdots, M$ **do**
  Allocate a random power vector for each cell.
  for $t = 1, \cdots, \infty$ **do**
    Calculate the CQI vector as well as the location indicator for every user in the network.
    Use the CQI vector and the location indicator as state s_t.
    for k $= 1, \cdots, K$ **do**
      With probability ε select a random action $\mathfrak{a}_k$ for cell k
      Otherwise select $\mathfrak{a}_k = \arg\max_{a \in \mathbb{A}_k} Q(s_t, a; \theta)$
    end for
    Execute action $a_t = [\mathfrak{a}_1, \mathfrak{a}_2, \cdots, \mathfrak{a}_K]$ and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = r_j$ if episode terminates at step $j + 1$
    Otherwise set $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^{-})$
    Perform gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every B steps reset $\hat{Q} = Q$
  end for
end for

throughput using Eq. (3.7). Use this throughput as *previous_throughput* for next step.

Step 4: Use the ε-greedy policy to select the actions randomly or use the Q-network to choose the actions. I use the state s_t as input to the Q-network to calculate the action-value for each action as we have m number actions for each cell. Therefore, for the first cell, select the action which action-value is maximum among the first m actions. Then, for the remaining cells, select the action consecutively with maximum action-value from the next m actions and so on.

Step 5: Execute the selected actions, i.e. map the actions with their corresponding power vectors and calculate the new state s_{t+1} in the same way mentioned in

Step 2. The agent then receives a positive reward of $r_t = +1$ I also calculate the total network throughput using Eq. (3.7) and save the value as *current_throughput*. Then, I check whether the new step is a terminal state or not using the following condition: *current_throughput* > *previous_throughput*. Finally, I store the transitions (s_t, a_t, r_t, s_{t+1}) in replay memory D.

Step 6: Perform experience replay on the Q-network. The experience replay mechanism has the following steps.

Step 6.1: Sample a minibatch of transitions of size $\mathcal{M}$ randomly from the replay memory D.

Step 6.2: Use Eq. (3.4) to update the targets y_j of that minibatch. I use the target action-value $\hat{Q}$ network to generate the targets.

Step 6.3: Perform a gradient descent step on the loss function (Eq. (3.2)) to update the action-value Q-network parameters.

Step 6.4: Clone the action-value function Q parameters to get the target action-value function $\hat{Q}$ at every B updates.

Step 7: Repeat **Steps 5-6** until the agent reaches the terminal state.

Step 8: Repeat **Steps 2-7** to train the Q-network for certain amount of time.

Testing Phase: After training our model, I need to test how close the model can predict compared to the optimal one in terms of total network throughput. I use GA to find a near-optimal solution.

For testing, the following steps are added to the training steps:

Step 9: Repeat **Steps 2-7** and save the second last action and the network throughput for that action. Therefore, if s_{t+1} is the terminal state and a_t is the action for which the agent reaches the terminal state, then I need to save the action a_{t-1} and the network throughput for that action. Here, the action a_{t-1} is considered to be the

optimal action.

Step 10: Find the power vector of every BS which maximizes the total network utility (Eq. (3.7)). I use GA to solve this problem.

Step 11: Apply the optimal power vector solution to Eq. (3.7) to calculate the total network throughput. I also save the optimal solution and network throughput.

Step 12: Keep repeating the steps until I have a certain amount of testing data.

All the training and testing will be performed online. In a practical setting, all users in the network periodically send their CQI values to their serving BSs, which extract the CQI value of each sub-band and add a location indicator, i.e. cell-centre user or cell-edge user. Therefore, for every user, there will be a vector of CQI and location indicator. Each BS then sends the processed information of all users to a central entity (e.g. SDN controller), which runs the DQL model. The DQL agent selects the power vector for all the BSs as an action. Once the agent chooses the action, the controller will send back the power vectors to their designated BSs. The BSs then allocate the power accordingly.

3.5 Experimental Setup

I present the simulation settings of the proposed DQL based power allocation scheme. I implement the proposed algorithm using Tensorflow [52]. I consider three different simulation scenarios: Scenario 1 with $K = 5$ BSs, Scenario 2 with $K = 10$ BSs, Scenario 3 with $K = 15$ BSs, cell coverage radius $R = 500$m, maximum transmit power $= 40$W, directional antenna per cell $= 3$, number of users per cell $= 5$, bandwidth of a sub-band $B = 2.88$MHz, white noise power density $= -174$dBm/Hz, number of sub-bands $= 3$, number of power levels $= 5$, and power levels $= \{6.4, 9.6, 12.8, 16, 19.2\}$W.

3.5.1 Training the DQL Model

First I need to define the DQN. I use a deep neural network of one hidden layer as our DQN. I use Rectified Linear Unit (ReLU) as an activation function for the hidden layer. The state size, i.e. the input layer size of the Q-network for three different scenario is $5 \times 5 \times (3+1) = 100$, $10 \times 5 \times (3+1) = 200$, $15 \times 5 \times (3+1) = 300$. For each cell, the total number of power combinations possible for 5 power levels is $5^3 = 125$. Some of the power combinations will be discarded due to the maximal power constraint. After applying the limitation, I have 72 power combinations for each cell. So, the total number of actions, i.e. the output layer size of the DQN for three different scenario is $5 \times 72 = 360$, $10 \times 72 = 720$, $15 \times 72 = 1,080$. The training parameters of the DQN are shown in Table 3.2.

Table 3.2: Training parameters

Parameter	**Value**
Number of hidden layers	1
Layers	{ Input, Hidden Layer, Output }
No. of neurons per layer	Scenario 1 : $\{100, 720, 360\}$ Scenario 2 : $\{200, 1440, 720\}$ Scenario 3 : $\{300, 2160, 1080\}$
Replay memory size	$80,000$
Batch size	64
Update target frequency, B	1000
Learning rate	0.00025
Loss function	MSE
Optimizer	RMSprop
Maximum value of ε, $\varepsilon_{\max}$	1
Minimum value of ε, $\varepsilon_{\min}$	0.01
Speed of decay, λ	0.001
No. of epochs per training	1

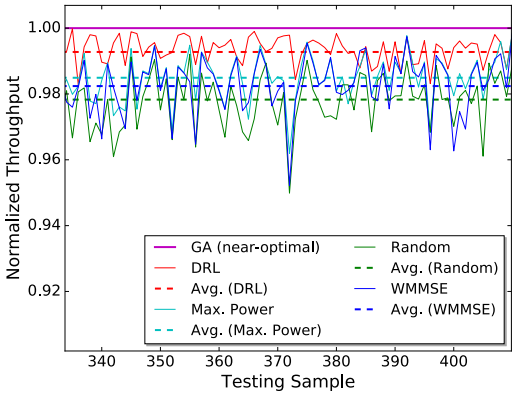
3.5.2 Testing the DQL Model

After training the DQL model for about 80 hrs, I compare our model with the optimal power allocation. I use the GA approach to find near-optimal power allocation. I also compare our power allocation (PA) model with other power allocation models such as WMMSE [47], maximum power allocation (PA), and random power allocation model. I use 12.8 W power for each subband for maximum power allocation.

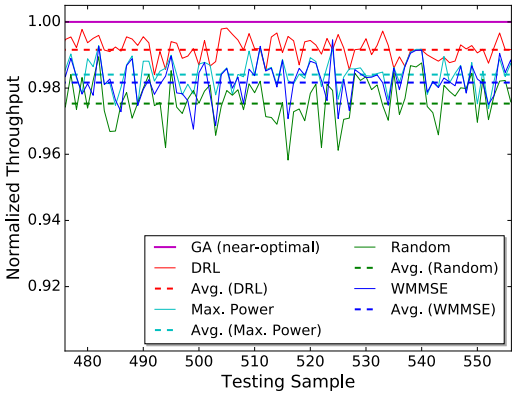
3.5.3 Results and Discussions

For comparison purpose, I calculate the total network throughput for the power allocation solution achieved through different PA model as well as for the near-optimal PA solution derived through GA. Then I calculate the normalized total network throughput of different PA models by dividing it with the total network throughput of the GA solution. Fig. 3.2 shows the normalized throughput of different PA models vs. testing samples for different network scenarios. From the figure, it is evident that the proposed DRL-based PA model performs better than other PA models.

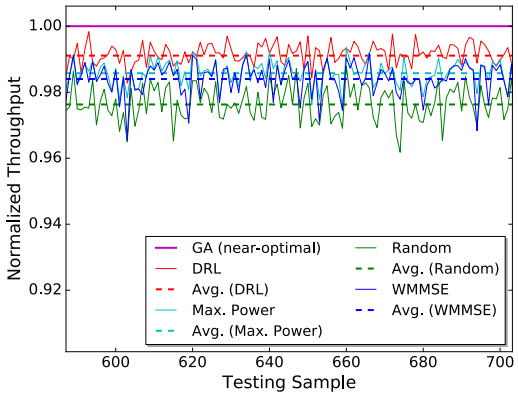
Impact of the wireless network size on the DRL performance: The average normalized throughput from our proposed DRL based PA model for scenario-1 is 0.99276, scenario-2 is 0.99157 and scenario-3 is 0.99109. From Fig. 3.2, it is also evident that with the increase of the wireless network size (i.e. the number of cells), the performance of the proposed DRL-based PA model decreases gradually. The average normalized network throughput decreases with the increase of the wireless network size. This is because, with the increase of the wireless network size, the state space and the action space also increases. As a result, the DQN needs to explore more state-action space to find the optimal action policy. Therefore, more exploration is



(a) Scenario 1: 5 cells



(b) Scenario 2: 10 cells



(c) Scenario 3: 15 cells

Figure 3.2: Normalized throughput vs. testing samples.

required for large state-action space. This is why the performance of our DRL model degrades gradually with the increase of wireless network size.

Impact of the hidden layer size of the DQN on the DRL performance: The number of hidden layers of the DQN is an important parameter as the DQN approximate the action-value function (Q). The DQN approximates the state-action relationship by extracting useful information from the sate. More hidden layers in DQN means it can learn more features. I vary the hidden layer size of the DQN

and repeat the simulations. Fig. 3.3 shows the average normalized throughput of the proposed model vs. the number of hidden layers for different network scenarios. It is apparent that the performance of the DRL model slightly degrades with the increase of the hidden layer size of the DQN. This is because with the increase of more hidden layers, the DQN learns irrelevant features (noise) and as a result of that *overfitting* occurs which eventually degrades the performance of the DQN.

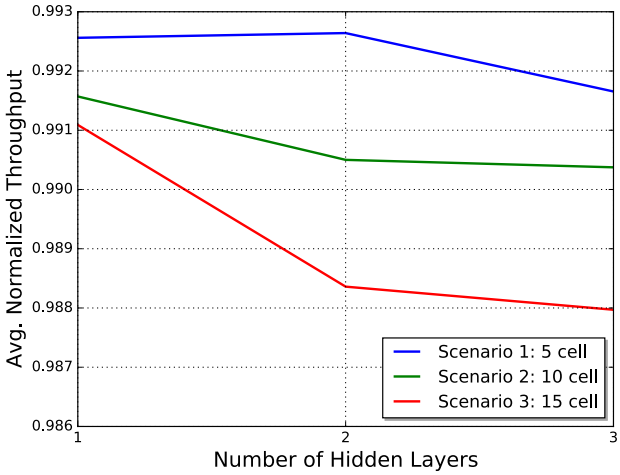


Figure 3.3: Average normalized throughput vs. number of hidden layers.

Impact of the learning rate on the DRL performance: The learning rate is an important hyperparameter that controls the amount of change in weights of the DQN during the training procedure. It controls how quickly or slowly a DQN learns from data. Finding the optimal learning rate is challenging since a small learning rate may result in larger training time and a large learning rate may result in an unstable training process. Next, I vary the learning the rate of the DRL keeping other parameters fixed. Fig. 3.4 shows the average normalized throughput of the proposed model vs. learning rate for different network scenarios. The optimal learning rate for scenario-1 is 0.0025 and for scenario-2 and scenario-3 is 0.025.

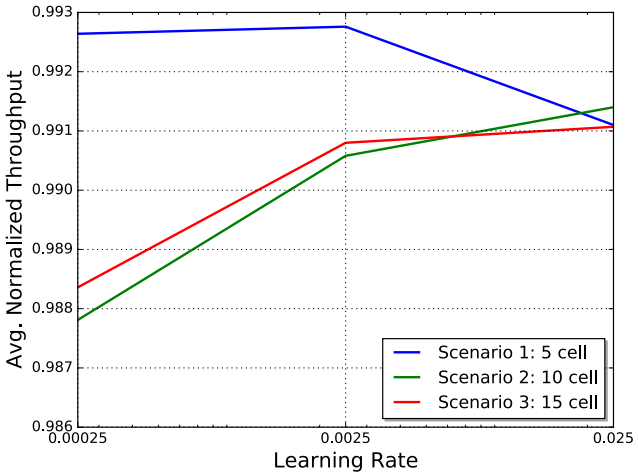


Figure 3.4: Average normalized throughput vs. learning rate.

3.6 Summary

I have presented a novel DRL-based method for power allocation in multi-cell networks. Specifically, I have used DQL with experience replay for the proposed method. Simulation results show that the DQN with one hidden layer is enough to approximate the action-value function for our case. The learning rate is the most important hyperparameter in DRL and finding optimal learning is challenging. To find the optimal learning rate for different network scenarios, we have varied the learning rate and observed the performance of the proposed model. I have evaluated the performance of the proposed DRL-based power allocation method with other power allocation methods such as WMMSE, maximum power allocation, and random power allocation for different network scenarios. Simulation results show that the proposed method is scalable to large-scale scenarios and it performs better compared to other PA models.

Chapter 4

Conclusion and Future Directions

4.1 Conclusion

In this thesis, I have provided a brief tutorial on deep learning, different deep learning architectures, the training procedure and the applicability of deep learning as a tool to model the resource allocation problem of the multi-cell wireless network. As its main contributions, this thesis provided optimal/ near optimal resource allocation solution for a multi-cell wireless network.

A centralized resource allocation scheme using supervised deep learning has been developed to perform sub-channel and power allocation for multi-cell wireless networks. The main objective is to maximize the total network throughput. The proposed model takes the CQI values of all the users in a multi-cell network as input and allocates resources (i.e. both sub-band and power) for each cell. The proposed deep learning-based resource algorithm framework needs only onetime offline training. After training, it can perform resource allocation in an online fashion. I have used GA to generate optimal/ near optimal training and testing data. Finding the optimal configuration of the DL model e.g. number of hidden layers and the training

data size is challenging. For those reasons, I have varied the number of hidden layers and the number of training samples and observed the performance of the proposed algorithm.

The challenging part of the supervised DL-based resource allocation is the generation of training data. For a large-scale network, generating the optimal training data is computationally expensive and time-consuming. Therefore, the supervised DL approach is not feasible for a large-scale network. Therefore, I use the DRL approach for a large-scale network. Unlike the supervised DL approach, the DRL approach does not require any optimal/ near-optimal data for training. I have developed a centralized DRL based resource allocation framework for a multi-cell network. I have defined the state space, action space and the reward function of the DRL agent. I have also described the online training procedure of the proposed DRL model. I have simulated the proposed DRL based resource allocation scheme on different network scenario of a different network to show the scalability and robustness of the algorithm. Finding the optimal learning rate and the number of hidden layers is also challenging for DRL model. Numerous simulation results with different training parameters have been presented to show their effect on the performance of the proposed model. Simulation results indicate the proposed DRL-based resource allocation scheme is scalable to large-scale scenarios.

4.2 Future Research Directions

The resource allocation problem is a fundamental problem in wireless networks. Some of the possible future extensions of the work presented in this thesis are as follows:

- In this work, I have not considered uncertainty in the channel state information. This work can be extended by incorporating the uncertainty in the wireless

propagation environment.

- In this work, I have not considered the users' mobility. But in a dense and heavily-loaded network, a considerable number of users can be mobile and they may frequently change their associations with the cells. Therfore, one possible future extension can be incorporate mobility into the resource allocation model.

- This work only focuses on maximizing the total network throughput. This work can be extended by incorporating fairness or users' QoS requirement (e.g. minimum rate guarantee for each user) in the resource allocation model.

- Recently, generative adversarial network (GAN) has emerged as a promising technique to create synthetic data from noise. A GAN has two components: a generator which tries to generate data from a latent space as close to real data distribution and a discriminator which attempts to distinguish the actual data distribution from the fake one. The generator and the discriminator compete with others by playing a min-max game. As a result of the min-max game, the generator generates data with the same distribution as the real data and therefore the discriminator cannot identify the difference between the actual data and the fake data. Therefore, a GAN can be used to generate synthetic training data for DL based resource allocation scheme.

- One major drawback of DRL is that it can end up having a local optimum instead of a global optimum. On the contrary, GAN is well known for reaching global optimum because of its loss function (i.e. binary cross entropy) used in training. Therefore, by combining GAN with RL, a GAN-RL might perform better compared to DRL. A GAN-RL based resource allocation framework can be developed for multi-cell wireless networks.

Bibliography

[1] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[2] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. MIT press Cambridge, 2016, vol. 1.

[3] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[4] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," in *IEEE Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2017, pp. 1–6.

[5] N. McKeown, "Software-defined networking," *INFOCOM keynote talk*, vol. 17, no. 2, pp. 30–32, 2009.

[6] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction mit press," *Cambridge, MA*, 1998.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[8] 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.

[9] A. Zappone, M. Debbah, and Z. Altman, "Online energy-efficient power control in wireless networks by deep neural networks," *IEEE Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2018.

[10] M. Eisen, C. Zhang, L. Chamon, D. D. Lee, and A. Ribeiro, "Learning optimal resource allocations in wireless systems," *arXiv:1807.08088*, 2018.

[11] J. Kim, J. Park, J. Noh, and S. Cho, "Completely distributed power allocation using deep neural network for device to device communication underlaying LTE," *arXiv:1802.02736*, 2018.

[12] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs," in *IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[13] H. Ye, Y. G. Li, and B.-H. F. Juang, "Deep reinforcement learning for resource allocation in v2v communications," *IEEE Transactions on Vehicular Technology*, 2019.

[14] U. Challita, L. Dong, and W. Saad, "Proactive resource management in LTE-U systems: A deep learning perspective," *arXiv preprint arXiv:1702.07031*, 2017.

[15] X. Li, J. Fang, W. Cheng, H. Duan, Z. Chen, and H. Li, "Intelligent power control for spectrum sharing in cognitive radios: A deep reinforcement learning approach," *IEEE Access*, vol. 6, pp. 25 463–25 473, 2018.

[16] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[17] Y. S. Nasir and D. Guo, "Deep reinforcement learning for distributed dynamic power allocation in wireless networks," *arXiv preprint arXiv:1808.00490*, 2018.

[18] F. Meng, P. Chen, L. Wu, and J. Cheng, "Power allocation in multi-user cellular networks: Deep reinforcement learning approaches," *arXiv preprint arXiv:1901.07159*, 2019.

[19] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Transactions on Wireless Communications*, vol. 18, no. 1, pp. 310–323, 2019.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[23] C. Watkins and P. Dayan, "gtechnical note: Q-learning, h machine learning, vol. 8," 1992.

[24] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.

[25] I. Jolliffe, *Principal component analysis*. Springer, 2011.

[26] W. Sun, S. Shao, R. Zhao, R. Yan, X. Zhang, and X. Chen, "A sparse auto-encoder-based deep neural network approach for induction motor faults classification," *Measurement*, vol. 89, pp. 171–178, 2016.

[27] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 2011, pp. 265–272.

[28] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision Research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[29] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[30] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992.

[31] Y. E. NESTEROV, "A method for solving the convex programming problem with convergence rate $o(1/k^2)$," *Dokl. Akad. Nauk SSSR*, vol. 269, pp. 543–547, 1983.

[32] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2021068

[33] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning," *Coursera, video lectures*, vol. 264, 2012.

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

[35] "Lte evolved universal terrestrial radio access (e-utra): Physical layer procedures (3gpp ts 36.213 version 8.8.0 release 8)," *ETSU TS 136 213 V8.8.0 Technical Specification*, 2009-10.

[36] S. Sivanandam and S. Deepa, "Genetic algorithm optimization problems," in *Introduction to Genetic Algorithms*. Springer, 2008, pp. 165–209.

[37] A. Zappone, E. Björnson, L. Sanguinetti, and E. Jorswieck, "Globally optimal energy-efficient power control and receiver design in wireless networks," *IEEE Transactions on Signal Processing*, vol. 65, no. 11, pp. 2844–2859, 2017.

[38] S. Boyd, V. Balakrishnan, and P. Kabamba, "A bisection method for computing the h norm of a transfer matrix and related problems," *Mathematics of Control, Signals and Systems*, vol. 2, no. 3, pp. 207–219, 1989.

[39] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[40] M. Bennis and D. Niyato, "A q-learning based approach to interference avoidance in self-organized femtocell networks," in *2010 IEEE Globecom Workshops*. IEEE, 2010, pp. 706–710.

[41] H. Li, "Multiagent-learning for aloha-like spectrum access in cognitive radio systems," *EURASIP Journal on Wireless Communications and Networking*, vol. 2010, no. 1, p. 876216, 2010.

[42] J. Lundén, V. Koivunen, S. R. Kulkarni, and H. V. Poor, "Reinforcement learning based distributed multiagent sensing policy for cognitive radio networks," in *2011 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2011, pp. 642–646.

[43] A. Alsarhan and A. Agarwal, "Spectrum sharing in multi-service cognitive network using reinforcement learning," in *2009 First UK-India International Workshop on Cognitive Wireless Systems (UKIWCWS)*. IEEE, 2009, pp. 1–5.

[44] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[45] Z.-Q. Luo and S. Zhang, "Dynamic spectrum management: Complexity and duality," *IEEE journal of selected topics in signal processing*, vol. 2, no. 1, pp. 57–73, 2008.

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

[47] 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.

[48] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[49] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction (2nd edition, in preparation)," 2017.

[50] L.-J. Lin, "Reinforcement learning for robots using neural networks," CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 1993.

[51] "Lte evolved universal terrestrial radio access (e-utra): Physical layer procedures (3gpp ts 36.213 version 8.4.0 release 8)," in *ETSI TS 136 213 V8.8.0 Technical Specification*, 2009-10.

[52] M. A. et al., "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.