2017

# Learning to optimize: Training deep neural networks for wireless resource management

Haoran Sun
*Iowa State University*

# Learning to optimize: Training deep neural networks for wireless resource management

by

**Haoran Sun**

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Industrial Engineering

Program of Study Committee:

Mingyi Hong, Major Professor

Cameron MacKenzie

Peng Wei

Iowa State University

Ames, Iowa

2017

## DEDICATION

I would like to dedicate this thesis to my beloved parents for their endless love and support.

# TABLE OF CONTENTS

# LIST OF TABLES

## LIST OF FIGURES

## ACKNOWLEDGEMENTS

I would like to take this great opportunity to express my very profound gratitude to those people who have helped me during my graduate studies for last two years.

First and foremost, I would like to thank my advisor Dr. Mingyi Hong for his guidance on conducting research and academic writing. I enjoy the discussion with him and appreciate his endless help and support during my studies. His support has been invaluable to me during this process. Thanks to him so much.

I would also like to thank Mr. Xiangyi Chen, Dr. Qingjiang Shi, Dr. Xiao Fu and Prof. Nikos D. Sidiropoulos. I benefit a lot from their advice and ideas, either conducting the research or writing the thesis. Thanks for their help.

Finally, I would like to thank my committee members: Dr. Cameron Mackenzie and Dr. Peng Wei. I especially appreciate the time and effort they have spent, and their generosity advice to improve this thesis. Thank them so much.

## PREFACE

This thesis is modified from a paper entitled “Learning to Optimize: Training Deep Neural Networks for Wireless Resource Management”, which accepted by 2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC). Substantial contributions by other co-authors have been removed, and these parts are referred to the original paper in this thesis.

## ABSTRACT

For decades, optimization has played a central role in addressing wireless resource management problems such as power control and beamformer design. However, these algorithms often require a considerable number of iterations for convergence, which poses challenges for real-time processing. In this work, we propose a new learning-based approach for wireless resource management. The key idea is to treat the input and output of a resource allocation algorithm as an unknown non-linear mapping and to use a deep neural network (DNN) to approximate it. If the non-linear mapping can be learned accurately and effectively by a DNN of moderate size, then such DNN can be used for resource allocation in almost *real time*, since passing the input through a DNN to get the output only requires a small number of simple operations. In this work, we first characterize a class of 'learnable algorithms' and then design DNNs to approximate some algorithms of interest in wireless communications. We use extensive numerical simulations to demonstrate the superior ability of DNNs for approximating two considerably complex algorithms that are designed for power allocation in wireless transmit signal design, while giving orders of magnitude speedup in computational time.

# CHAPTER 1. OVERVIEW

## 1.1 Introduction

Resource management tasks, such as transmit power control, transmit/receive beamformer design, and user admission control, are critical for future wireless networks. Extensive research has been done to develop various resource management schemes; see the recent overview articles Hong and Luo (2013); Bjornson and Jorswieck (2013).

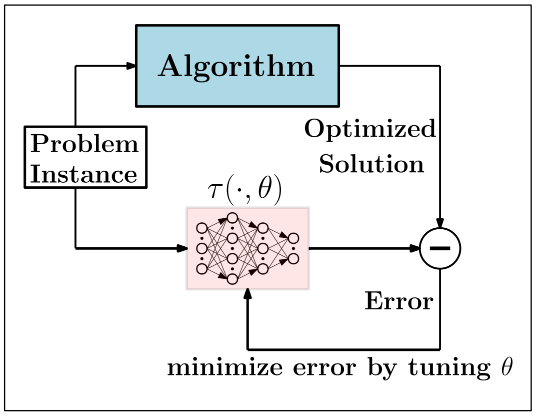
For decades, numerical optimization has played a central role in addressing wireless resource management problems. Well-known optimization-based algorithms for such purposes include those developed for power control (e.g., iterative water-filling type algorithms proposed in Yu et al. (2002); Scutari et al. (2008), interference pricing proposed in Schmidt et al. (2009), SCALE proposed in Papandriopoulos and Evans (2009)), transmit/receive beamformer design (e.g., the WMMSE algorithm proposed in Shi et al. (2011), pricing-based schemes proposed in Kim and Giannakis (2011), semidefinite relaxation based schemes proposed in Luo et al. (2010)), admission control (e.g., the deflation based schemes proposed in Liu et al. (2013), convex approximation based schemes in Matskani et al. (2008)), user/base station (BS) clustering (e.g., the sparse optimization based schemes proposed in Hong et al. (2013)), just to name a few. These algorithms have been very successful in handling their respective resource management problems. By nature, they all belong to the class of *iterative algorithms*, which take a given set of real-time network parameters like channel realizations and signal to noise ratio requirements as their inputs, run a number of iterations, and produce the "optimized" resource allocation strategies as their outputs.

Despite the excellent performance of many of these algorithms in achieving high system utility, implementing them into real systems still faces many serious obstacles. One of the most challenging one is the high real-time computational cost. For example, WMMSE-type algorithms require complex operations such as matrix inversion and bisection in each iteration Shi et al. (2011); Baligh et al. (2014). Water-filling type algorithms such as Yu and Cioffi (2002) and interference pricing algorithms proposed in Schmidt et al. (2009) involve singular value decomposition at each iteration (when dealing with MIMO interference systems). Algorithms such as the deflation-based joint power and admission control require successively solving a series of (possibly large-scale) linear programs. The high computational requirement of these algorithms makes their real-time implementation challenging, because they are typically executed in a time frame of milliseconds (due to the fast changing nature of the communication system parameters such as channels conditions, number of users, etc.).

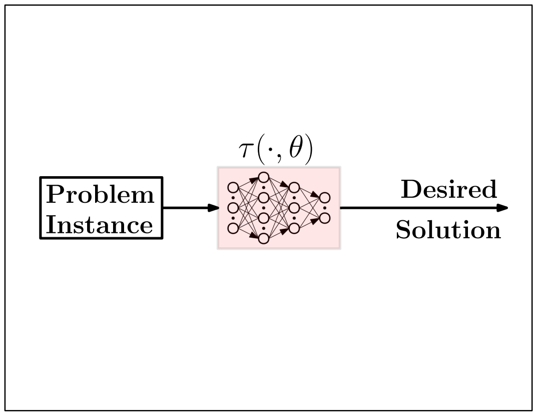
## 1.2 Proposed Method

In this work, we propose a new machine learning-based approach for wireless resource management shown in Figure 1.1. The main idea is to treat a given resource optimization algorithm as a "black box", and try to *learn* its input/output relation by using a *deep neural network* (DNN). Different from unfolding a specific algorithm for problem in Gregor and LeCun (2010) and Hershey et al. (2014), the resource management algorithms often entail computationally heavy iterations involving operations such as matrix inversion, SVD, and/or bi-section, their iterations are not amenable to approximation by a *single* layer of the network. Therefore, we advocate modeling/approximating the *unknown* end-to-end input-output mapping realized by a given algorithm using a carefully trained DNN. Note that there are other choices for finding a nonlinear mapping, e.g., kernel regression. We chose DNN here since its multi-layer structure bears some resemblance to the structure of iterative algorithms, as shown in Gregor and LeCun (2010); Hershey et al. (2014). In addition, for large-scale training sets, kernel methods usually have severe memory issues unless they employ heuristic approximation techniques that may sacrifice performance, which we wish to avoid for applications such as wireless transmit signal design.

The key advantage of using a DNN vis-a-vis an iterative algorithm is that the DNN (or neural network, for this matter) has high real-time computational efficiency compared with a typical iterative optimization algorithm (also see arguments made in Gregor and LeCun (2010)): Its *run-time stage* does not involve numerical optimization, but only some simple operations like vector multiplication/addition, and simple (scalar) nonlinear transformations. The upshot is that, if the training stage can approximate the algorithm accurately enough, then simple real-time resource allocation is possible. Overall, our approach can be viewed as using a DNN to "learn to optimize" a resource allocation strategy of interest according to given network parameters.



(a) Training Stage



(b) Testing Stage

Figure 1.1 Proposed Method

## 1.3 Contribution

The main contribution of this work is three-fold: First, we propose the first deep learning based scheme for real-time resource management over interference-limited wireless networks, which bridges the seemingly unrelated areas of machine learning and wireless resource allocation (in particular, power control over interfering networks). Second, we attempt to learn the behavior of entire optimization algorithms, different from learning the algorithm behavior layer by layer or learning to optimize, our methods benefits from both the theoretical analysis of existing algorithms and the computational efficiency of the deep neural networks. Third, we conduct theoretical analysis and extensive numerical experiments to demonstrate the achievable performance of the proposed approach. As a proof-of-concept, the preliminary results provided in this paper indicate that DNNs have great potential in the real-time management of the wireless resource. Beyond the considered scenarios, the results also suggest that DNNs can serve as a tool for approximating iterative optimization algorithms, and the proposed approach appears promising for other signal processing applications that require real-time processing.

To promote reproducible research, the codes for generating most of the results in the paper will be made available on the authors' website: `https://github.com/Haoran-S/ISUthesis`.

# CHAPTER 2. REVIEW OF LITERATURE

## 2.1 Wireless Communication Problem

### 2.1.1 System Model

We consider the following basic interference channel (IC) power control problem in Figure 2.1, for a wireless network consisting of $K$ single-antenna transmitter and receiver pairs. Let $h_{kk} \in \mathbb{C}$ denote the direct channel between transmitter $k$ and receiver $k$, and $h_{kj} \in \mathbb{C}$ denote the interference channel from transmitter $j$ to receiver $k$. All channels are assumed to be constant in each resource allocation slot. Furthermore, we assume that the transmitted symbol of transmitter $k$ is a Gaussian random variable with zero mean and variance $p_k$ (which is also referred to as the transmission power of transmitter $k$). Further, suppose that the symbols from different transmitters are independent of each other. Then the signal to interference-plus-noise ratio (SINR) for each receiver $k$ is given by

$$\mathrm{sinr}_k \triangleq \frac{|h_{kk}|^2 p_k}{\sum_{j \neq k} |h_{kj}|^2 p_j + \sigma_k^2},$$

where $\sigma_k^2$ denotes the noise power at receiver $k$.

We are interested in power allocation for each transmitter so that the weighted system throughput is maximized. Mathematically, the problem can be formulated as the following *nonconvex* problem

$$\begin{aligned} \max_{p_1,\dots,p_K} \quad & \sum_{k=1}^{K} \alpha_k \log\left(1 + \frac{|h_{kk}|^2 p_k}{\sum_{j \neq k} |h_{kj}|^2 p_j + \sigma_k^2}\right) \\ \text{s.t.} \quad & 0 \le p_k \le P_{\max},\ \forall\, k = 1, 2, \dots, K, \end{aligned} \tag{2.1}$$

where $P_{\max}$ denotes the power budget of each transmitter; $\{\alpha_k > 0\}$ are the weights. Problem (2.1) is known to be NP-hard in Luo and Zhang (2008).

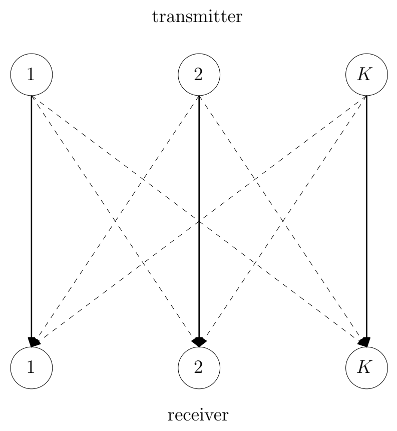


Figure 2.1 Interference Channel (IC)

### 2.1.2 Related Algorithms

Various power control algorithms have been proposed in Shi et al. (2011); Schmidt et al. (2009); Papandriopoulos and Evans (2009), among which the Weighted Minimize Mean Square Error (WMMSE) algorithm has been very popular. In what follows, we review a particular version of the WMMSE applied to solve the power control problem (2.1).

The weighted sum-rate (WSR) maximization problem is difficult mainly due to the presence of the $\log(\cdot)$ function. The WMMSE algorithm overcomes this difficulty by converting the problem to a higher dimensional space where it is easily solvable, using the well-known MMSE-SINR equality proposed in Verdu (1998), i.e., $\text{mmse}_k = \frac{1}{1+\text{sinr}_r}$, where $\text{mmse}_k$ denotes the minimum-mean-squared-error (MMSE) of user $k$. Here, we modify the WMMSE algorithm proposed in Shi et al. (2011) so that it can work in the real domain, which will greatly simplify our subsequent implementation of DNN based schemes.

Specifically, observing that the rate function remains unchanged if $h_{kj}$ is replaced by $|h_{kj}|$, we consider applying the WMMSE algorithm to an equivalent *real* interference channel model, given by

$$\hat{s}_k = u_k(|h_{kk}|v_k s_k + \sum_{j \neq k} |h_{kj}| v_j s_j + n_k), \quad k = 1, 2, \ldots, K$$

where $s_k$ is the transmitted symbol; $v_k$ is the amplifier gain used by transmitter $k$; $u_k$ is a receiver-side amplifier gain used to obtain the estimated real symbol denoted by $\hat{s}_k$; $n_k$ is the channel noise with variance $\sigma_k^2$.

Assuming that $\{s_k\}_{k=1}^K$ and $\{n_k\}_{k=1}^K$ are independent from each other, the MSE between $\hat{s}_k$ and $s_k$ is defined as

$$\begin{aligned} e_k &\triangleq \mathbb{E}_{s,z}(\hat{s}_k - s_k)^2 \\ &= (u_k|h_{kk}|v_k - 1)^2 + \sum_{j\neq k}(u_k|h_{kj}|v_j)^2 + \sigma_k^2 u_k^2 \end{aligned} \tag{2.2}$$

where $\mathbb{E}_{s,z}$ is the expectation operator taken with respect to the variables $\{s_k\}_{k=1}^K$ and $\{n_k\}_{k=1}^K$. Using (2.2), the WSR maximization problem (2.1) can be addressed by solving an equivalent weighted MSE minimization problem, given below Shi et al. (2011).

$$\begin{aligned} &\min_{\{w_k,u_k,v_k\}_{k=1}^K} \quad \sum_{k=1}^K \alpha_k \left(w_k e_k - \log(w_k)\right) \\ &\text{s.t.} \quad 0 \le v_k \le \sqrt{P_k},\ k = 1, 2, \ldots, K. \end{aligned} \tag{2.3}$$

The WMMSE solves (2.3) using the block coordinate descent method (Bertsekas and Tsitsiklis (1997)), i.e., each time optimizing one set of variables while keeping the rest fixed; see Fig. 1. for its detailed steps. It has been shown in Shi et al. (2011) that the WMMSE is capable of reaching a stationary solution of problem (2.1). Note that since the interfering multiple-access channel (IMAC) can be viewed as a special IC with co-located receivers, the WMMSE algorithm in Fig. 1 can be applied to solving the power allocation problem of IMAC as well [1].

## 2.2 Deep Neural Network

In recent years, the deep neural network has been widely used in the machine learning community since its superior in tons of approximation and prediction tasks. It has proven successful in areas like artificial intelligence (AI), machine learning (ML) (Hinton et al. (2012a); Krizhevsky et al. (2012)), where they are used as effective tools for transforming data to latent spaces that are suitable for ML tasks such as speech recognition (Hinton et al. (2012a)), image classification (Krizhevsky et al. (2012)), and natural language processing (Liu (2012)).

A simple or shallow fully connected neural network is shown as Figure 2.3, which consist of an input layer, one hidden layer, and an output layer. Such kind of network is very powerful due

[1] In the IMAC, we assume that within each cell the BS does not perform successive interference cancellation.

```
1.  Initialize $v_k^0$ such that $0 \leq v_k^0 \leq \sqrt{P_k}$, $\forall\ k$;
2.  Compute
        $$u_k^0 = \frac{|h_{kk}|v_k^0}{\sum_{j=1}^K |h_{kj}|(v_j^0)^2 + \sigma_k^2}, \forall\ k;$$
3.  Compute
        $$w_k^0 = \frac{1}{1 - u_k^0 |h_{kk}| v_k^0}, \forall\ k;$$
4.  Set $t = 0$
5.  **Repeat**
6.      $t = t + 1$   //iterations
7.      Update $v_k$:
        $$v_k^t = \left[\frac{\alpha_k w_k^{t-1} u_k^{t-1} |h_{kk}|}{\sum_{j=1}^K \alpha_j w_j^{t-1} (u_j^{t-1})^2 |h_{jk}|^2}\right]_0^{\sqrt{P_{\max}}}, \forall\ k;$$
8.      Update $u_k$:
        $$u_k^t = \frac{|h_{kk}|v_k^t}{\sum_{j=1}^K |h_{kj}|^2 (v_j^t)^2 + \sigma_k^2}, \forall\ k;$$
9.      Update $w_k$:
        $$w_k^t = \frac{1}{1 - u_k^t |h_{kk}| v_k^t}, \forall\ k;$$
10. **until**
        $$\left| \sum_{j=1}^K \log\left(w_j^t\right) - \sum_{j=1}^K \log\left(w_j^{t-1}\right) \right| \leq \epsilon;$$
11. **output**
        $$p_k = (v_k)^2, \forall\ k;$$
```

Figure 2.2 Pseudo code of WMMSE for the scalar IC.

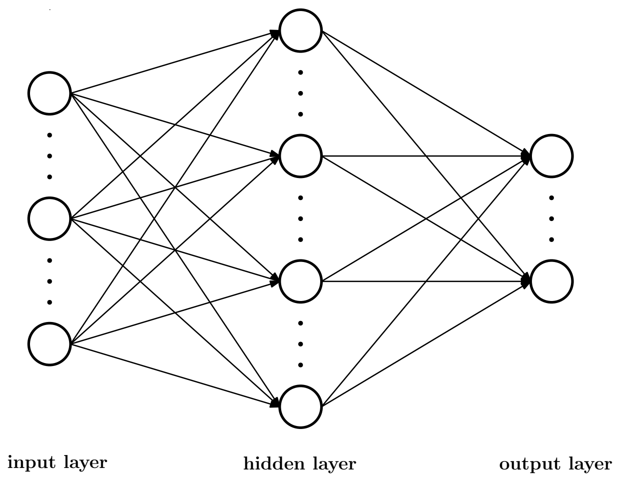


Figure 2.3 A fully connected Neural Network

to the universal approximation theorem stated in Cybenko (1989) and Hornik et al. (1989a). Cybenko (1989) has shown that any continuous function on compact subsets of $R^n$ can be approximated to arbitrary accuracy by a feed-forward network with a single hidden layer, as long as we have sufficient neurons with sigmoid activation for that hidden layer. Hornik et al. (1989a) has generalized such result to multi-layer architectures and other activation functions. Besides the powerful approximation ability, such network has computation efficiency benefits from their structure.

As shown in Figure 2.4, the neural network only involves two computational operations, one is matrix multiplication, and the other is activation function $f$. The matrix multiplication is widely used recently and could be implemented incredibly fast in practice as proposed in Coppersmith and Winograd (1990), and the activation function $f$ can be chosen to simple non-linear functions such as ReLU proposed in Nair and Hinton (2010) and Leaky ReLU in Maas et al. (2013) which can also be implemented pretty fast in practice.

If we replace one hidden layer with multiple hidden layers in our network, it becomes a "deep" feed-forward network. Such deep structure is powerful since it exponentially decreased the total number of network's weights while keep the same accuracy as claimed in Delalleau and Bengio (2011); Eldan and Shamir (2016); Telgarsky (2016) and Liang and Srikant (2016).

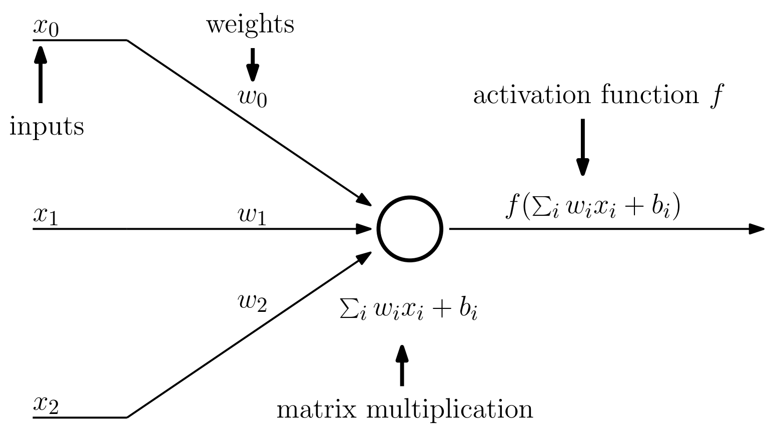


Figure 2.4 Computational graph of a Neural Network

In addition, Mhaskar et al. (2016) prove that “deep networks can approximate the class of compositional functions with the same accuracy as shallow networks but with exponentially lower number of training parameters”. Since the computational complexity of the neural network is measured by the number of weights and bias, if we have exponentially fewer parameters, we could exponentially speed up the computational time in testing stage, which gives us a way to design a run-time algorithm based on deep neural networks. Figure 2.5 shows an example of the neural network approximation ability when we approximate a polynomial plus cosine function $f(x) = 30 + 20x + 5x^2 + 1000\cos(\frac{x}{2})$ by a neural network with three hidden layers, each with 200, 80, 80 neurons respectively. It is readily seen that as long as we have enough hidden neurons, we can approximate the continuous function pretty well.

## 2.3 Learning to Optimize

In the machine learning community, there have been several attempts of approximating an iterative optimization algorithm using DNNs. The authors of Gregor and LeCun (2010) proposed to use a non-linear feed-forward multi-stage network to approximate the solutions generated by the iterative soft-thresholding algorithm (ISTA) for sparse optimization (Beck

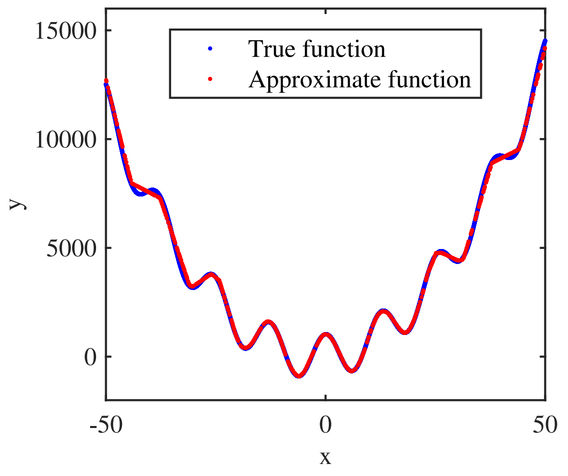


Figure 2.5 Use DNN to approximate continuous function

and Teboulle (2009)). In particular, the ISTA algorithm is first “unfolded”, and each of its first couple of iterations is mapped to a “layer” of the network. Then the parameters of the network are learned by offline training. The authors of Hershey et al. (2014) proposed a few improved network architectures and applied the resulting DNN to approximate algorithms for other tasks such as non-negative matrix factorization (NMF). The authors of Sprechmann et al. (2013) also proposed a similar idea to learn the alternating direction method of multipliers (ADMM) algorithm for different tasks. Recently, the authors of Andrychowicz et al. (2016) proposed to learn the per-iteration behavior of gradient based algorithms by gradient based algorithms, and the authors of Li and Malik (2016) proposed to use reinforcement learning to perform the same task. However, these algorithm still uses the unfolding idea, and repeated evaluation of gradients and functional values is required. Overall, the unfolding-based ideas are feasible either because the iterations of the algorithms have very simple structures (i.e., linear filter followed by a non-linear activation function that promotes sparsity) such as the algorithms in Gregor and LeCun (2010); Sprechmann et al. (2013) and Hershey et al. (2014) or due to the fact that per-iteration gradient information is assumed to be known like Li and Malik (2016) and Andrychowicz et al. (2016). Therefore it is reasonable that each iteration can be well approximated by using specially crafted neural networks. However, for complex algorithms such as those involves inversion and projection operations like WMMSE algorithms proposed in Shi et al. (2011), it is no longer clear whether each of its iteration can be explicitly modeled using a few layers of the neural network. Further, it is important to note that none of these existing works have provided a rigorous theory about their approaches. For example, it is not even clear whether algorithms such as ISTA can be accurately approximated by using the unfolding idea.

# CHAPTER 3. METHODS AND PROCEDURES

In this section, we propose to use DNN to approximate WMMSE in an end-to-end fashion. In the proposed scheme, WMMSE is treated as an unknown nonlinear mapping, and a DNN is trained to learn its input/output relation. Our motivation is to leverage the high computational efficiency of DNN in its testing stage to design a fast real-time resource management scheme.

## 3.1 Universal Approximation

At this point, it remains unclear whether a multi-layer neural network can be used to approximate the behavior of a given iterative algorithm, like WMMSE, for solving the nonconvex optimization problem (2.1). The answer to such a question is by no means trivial, because of the following reasons: 1) it can be seen that the WMMSE algorithm described in Fig. 1 is complicated and involves operations such as inversion and projection; 2) we do not explicitly *unfold* the WMMSE algorithm and model each of its iterations (such as what has been proposed in Gregor and LeCun (2010); Hershey et al. (2014)).

Sun et al. (2017) has shown that for any algorithms whose iterations represent continuous mappings, we can fix its initialization and learn its input/output relation by a well trained neural network. The intuition to "learn" the iterative algorithm is to "learn" a continuous mapping from the problem parameter to the optimal solution. Thus the fixed initialization is necessary since it prevents multiple optimal solutions from the same set of problem parameters. Consider solving the following optimization problem

$$\min_{x \in X} f(x) = (x^2 - z)^2$$

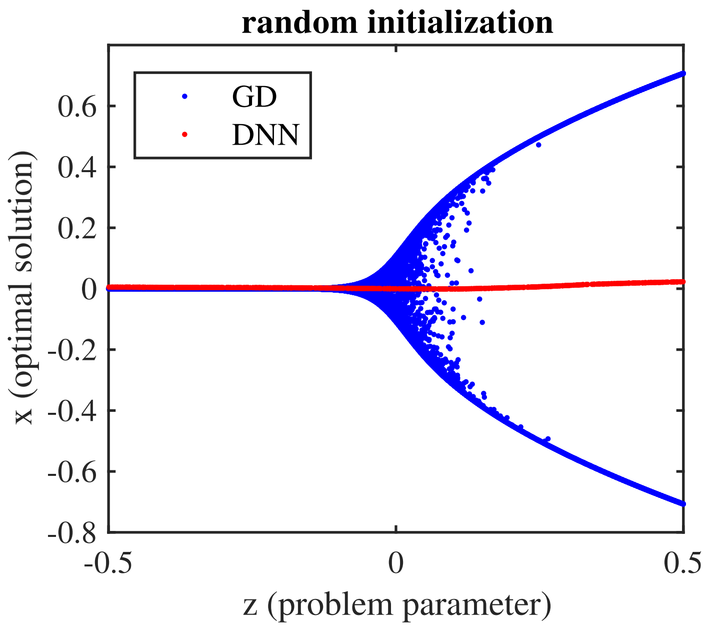


Figure 3.1 Use DNN to approximate iterative algorithm, random initialization

where $x$ is the optimization variable and $z$ is the random generated problem parameter. Then, if we solve it by the iterative gradient descent algorithm

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

where $k$ represents the iteration number, $x_0$ represents the random generated initial solution and $\alpha$ represents the constant stepsize, we have multiple optimal solutions $x^* = +\sqrt{z}$ and $-\sqrt{z}$ when $z$ is greater than zero. If we mimic such process by a DNN then the result could be unlearnable as shown in Figure 3.1. However, if we fix the initialization and run gradient descent, the results could be easily approximated like Figure 3.2.

It is also important to note that although the Theorem 1 proposed in Sun et al. (2017) states that a single hidden layer can achieve arbitrarily accurate approximation, it requires a large number of hidden units as well. To improve the computation efficiency of our "Learner", a "deep" neural network is highly needed as shown in Mhaskar et al. (2016) . It can also be verified that each iteration of the WMMSE represents a continuous mapping, thus by Hornik et al. (1989b) and Theorem 1 in Sun et al. (2017) this algorithm can be approximated arbitrarily

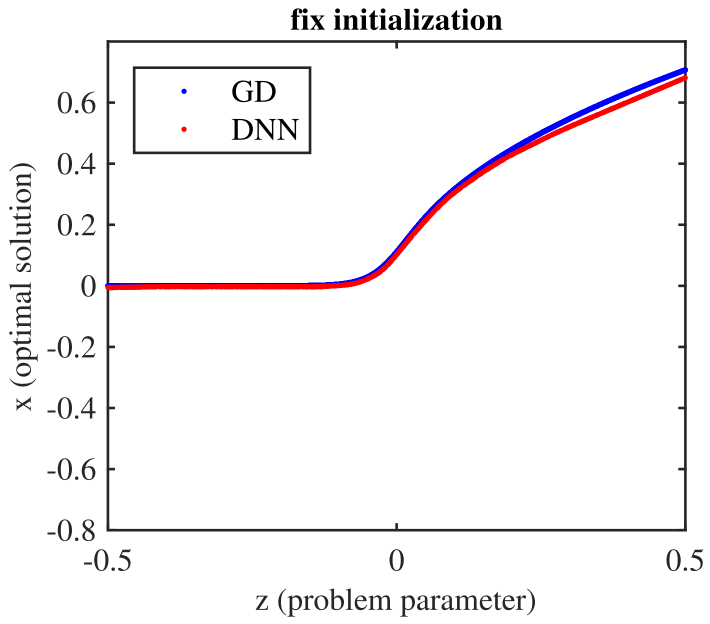


Figure 3.2 Use DNN to approximate iterative algorithm, fix initialization

well by a feedforward network with a single hidden layer.

## 3.2 System Setup

### 3.2.1 Network Structure

Our proposed approach approximates the WMMSE algorithm using a fully connected neural network with one input layer, multiple hidden layers, and one output layer as shown in Figure 3.3. The input of the network is the channel coefficients $\{|h_{kj}|\}$, and the output of the network is the power allocation $\{p_k\}$. The reason for using multiple hidden layers (as compared to using a single hidden layer) is the following: It is widely accepted that by doing so, the total number of hidden units can be significantly reduced, resulting in better computation efficiency while keeping comparable accuracy. Further, we use ReLU as the activation function for the hidden layers: $y = \max(x, 0)$ (where $x$ is the input and $y$ is the output of the activation function). Additionally, to enforce the power constraint in (2.1) at the output of DNN, we also choose a

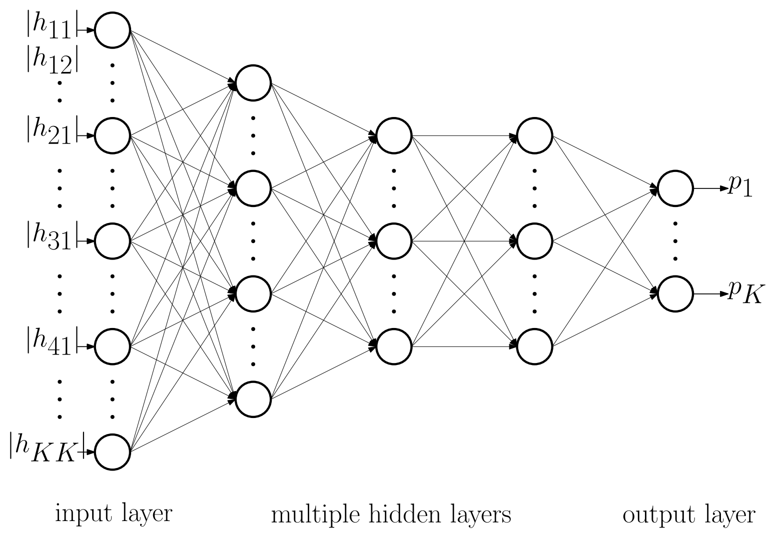


Figure 3.3 The DNN structure used in this work.

special activation function for the output layer, given below

$$y = \min(\max(x, 0), P_{max}). \tag{3.1}$$

### 3.2.2 Data Generation

The training data is generated in the following manner. First, we generate the channel realizations $\{h_{jk}^{(i)}\}$, following certain distributions (to be specified in Chapter 4), where $(i)$ is used to denote the index of the training sample. For simplicity we fix $P_{\max}$ and $\sigma_k$ for all $k$. Then, for each tuple $(P_{\max}, \{\sigma_k\}, \{|h_{kj}^{(i)}|\})$, we generate the corresponding optimized power vectors $\{p_k^{(i)}\}$, by running the WMMSE, with $v_k^0 = \sqrt{P_{max}}$, $\forall\ k$ as initialization, and with $\text{obj}_{new} - \text{obj}_{old} < 10^{-3}$ as termination criteria. We call the tuple $(\{|h_{kj}^{(i)}|\}, \{p_k^{(i)}\})$ the $i$th training sample. Then, we repeat the above process for multiple times to generate the entire training data set, as well as the validation data set. The validation set is used to perform the cross-validation, model selection and early stopping during the training stage. Typically, the size of the validation data set is small compared with that of the training set. We use $\mathcal{T}$ and $\mathcal{V}$ to collect the indices for the training and validation sets, respectively.

### 3.2.3 Training Stage

We use the entire training data set $(\{|h_{kj}^{(i)}|\}, \{p_k^{(i)}\})_{i\in\mathcal{T}}$ to optimize the weights of the neural network. The cost function we use is the mean square error between the label $\{p_k^{(i)}\}$ and the output of the network. The optimization algorithm we use is an efficient implementation of mini-batch gradient descent called the *RMSprop* algorithm, which divides the gradient by a running average of its recent magnitude proposed in Hinton et al. (2012b). We choose the decay rate to be 0.9 as suggested in Hinton et al. (2012b) and select the proper learning rate and batch size by cross-validation. To further improve the training performance, we initialize the weights using the truncated normal distribution[1] (TensorFlow (2017)). Furthermore, we divide the weights of each neuron by the square root of its number of inputs to normalize the variance of each neuron's output, which has been proposed in Glorot and Bengio (2010).

### 3.2.4 Testing Stage

In the testing stage, we first generate the channels following *the same* distribution as the training stage. For each channel realization, we pass it through the trained network and collect the optimized power. Then, we compute the resulting sum-rate of the power allocation generated by DNN and compare it with that obtained by the WMMSE.

[1]We generate a variable from the truncated normal distribution in the following manner: First, generate a variable from standard normal distribution. Then if its absolute value is larger than 2, it is dropped and re-generated.

# CHAPTER 4. RESULTS

This section presents numerical examples to showcase the effectiveness of the proposed approach.

## 4.1 Simulation Setup

In our numerical results, codes for implementing the proposed neural network based approach are implemented in Python 3.6.0 with TensorFlow 1.0.0 on one computer node with two 8-core Intel Haswell processors, two Nvidia K20 Graphical Processing Units (GPUs), and 128 GB of memory. The GPUs are used in the training stage to reduce the training time, but they are not used in the testing stage. To rigorously compare the computational performance of WMMSE with the proposed approach, we have implemented the WMMSE in both Python and C. The Python implementation is used to verify the computational performance under the same platform while the C implementation is used to achieve the best computational efficiency of WMMSE. We consider the following two different channel models:

**Model 1: Gaussian IC.** Each channel coefficient is generated according to a standard normal distribution, i.e., Rayleigh fading distribution with zero mean and unit variance. Rayleigh fading is a reasonable channel model that has been widely used to simulate the performance of various resource allocation algorithms. In our experiment, we consider three difference cases with $K \in \{10, 20, 30\}$.

**Model 2: IMAC.** For practical consideration, a multi-cell Interfering Multiple Access Channel (IMAC) model is considered with a total of $N$ cells and $K$ users. The distance between centers of adjacent cells is set to be 200 meters. In each cell, one BS is placed at the center of the cell and the users are randomly and uniformly distributed; see Figure 1 of Liao et al. (2014) or

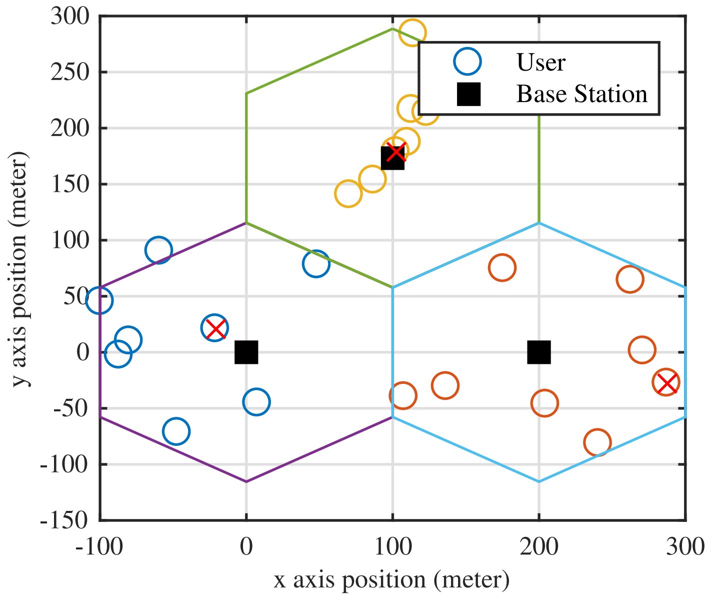


Figure 4.1 Interfering Multiple Access Channel (IMAC) with $N = 3, K = 24$

Figure 4.1 for an illustration of the network configuration. We assume that the cells all have the same number of users. The channel between each user and each BS is randomly generated according to a Rayleigh fading distribution with zero mean and variance $(200/d)^3L$, where $d$ denotes the distance between the BS and user, and the quantity $L$ represents the shadow fading following a log-normal distribution with zero mean and variance 64. In our experiment, we consider four difference network scenarios, with $(N, K) \in \{(3, 16), (3, 24), (3, 60), (7, 28)\}$.

We mention that for each network scenario (i.e., IC/IMAC with different number of BSs/users), we randomly generate one million realizations of the channel, among which 99% is the training data and 1% is the validation data.

## 4.2 Parameter Selection

For all our numerical results, we choose the following parameters for the neural network. We use a network with three hidden layers, one input layer, and one output layer. The $1^{st}$ hidden layer contains 200 neurons, and both $2^{nd}$ and $3^{rd}$ hidden layers consist of 80 neurons. The input to the network is the set of channel coefficients, therefore the size of it depends on the channel models. More specifically, for Gaussian IC the input size is $K^2$ while for the IMAC it is $N \times K$. The output is the set of power allocation, therefore its size is $K$ for both Gaussian IC and IMAC.

To find parameters for the training algorithm, we perform cross-validation for different channel models as follows:

**Model 1: Gaussian IC.** We study the impact of the batch size and learning rate on the mean square error (MSE) evaluated on the validation set, as well as the total training time. Based on the result shown in Figure 4.2 and Figure 4.3, we see that larger batch size leads to slower convergence, while smaller batch size incurs unstable convergence behavior. Similarly, larger learning rate leads to a higher validation error, while the lower learning rate leads to slower convergence. Thus, we choose the batch size to be 1000 and the learning rate to be 0.001, which results in relatively small MSE.

**Model 2: IMAC.** The parameter selection process is almost the same as that in the Gaussian IC channel except that we choose to gradually decrease the learning rate when the validation error does not decrease. Further, the batch normalization technique proposed in Ioffe and Szegedy (2015) is used for the $N = 7$ and $K = 28$ case to speed up the training.

## 4.3 Sum-Rate Performance

We evaluate the sum-rate performance of the DNN-based approach in the testing stage compared to the following schemes: 1) the WMMSE; 2) the random power allocation strategy (RAND), which simply generates the power allocation as: $p_k \sim \text{Uniform}(0, P_{\max})$, $\forall\, k$; 3) the maximum power allocation (MP): $p_k = P_{\max}$, $\forall\, k$. The latter two schemes serve as heuristic baselines. The simulation results are shown in Figure 4.4 that describe the rates achieved by different algorithms for both the Gaussian IC and the IMAC. Each curve in the figure represents the result obtained by averaging $10,000$ randomly generated testing data points. Figure 4.4 (a) shows the Gaussian IC with $K = 10$, Figure 4.4 (b) shows the IMAC with $N = 3$ and $K = 24$.

It is observed that the sum-rate performance of DNN is very close to that of the WMMSE, while significantly outperforming the other two baselines. Compared with WMMSE, on average 97.92% (resp. 93.02%) accuracy is achieved for Gaussian IC Channel (resp. IMAC). It is

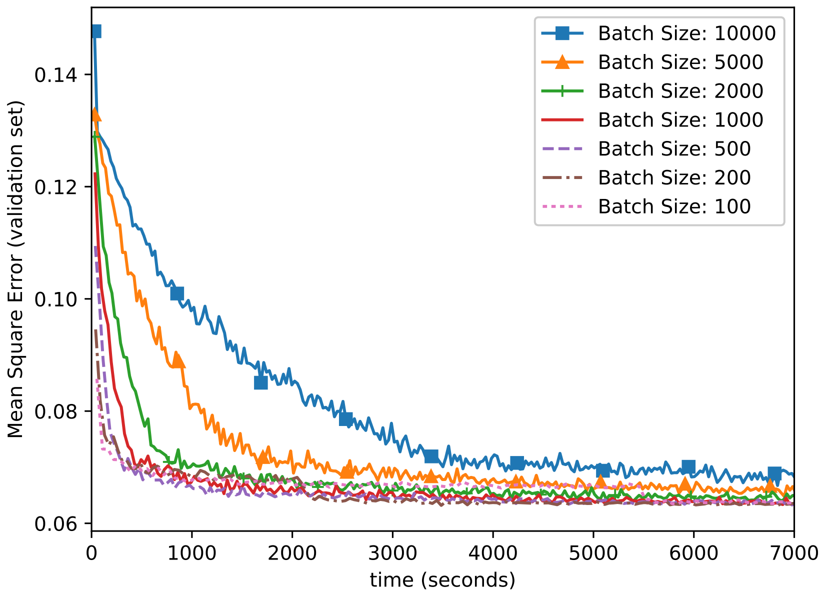


Figure 4.2 Batch size selection for Gaussian IC, $K = 30$

worth noting that for Gaussian IC, we have observed that the "optimized" allocation strategies obtained by WMMSE are binary in most cases (i.e., $p_k$ takes the value of either 0 or $P_{\max}$). Therefore, we also discretize the prediction to binary variables to increase the accuracy.

## 4.4 Scalability Performance

In this subsection, we demonstrate the scalability of the proposed DNN approach when the size of the wireless network is increased. Furthermore, to verify the learning capability of DNN in approximating other baseline algorithms, we also use DNN to learn a greedy binary power control algorithm in Gjendemsjo et al. (2008). The average achieved performance (averaged using $10,000$ test samples) over IC (resp. IMAC) are presented in TABLE 4.1 (resp. in TABLE 4.3). Further, the percentage of achieved performance of DNN over that of the WMMSE/Greedy are presented in TABLE 4.2 (for the IC) and 4.4 (for the IMAC). It can be seen that our proposed method achieves very good scalability for prediction accuracy and computational efficiency. For example, compared with the IMAC WMMSE (which is implemented in C) with $N = 3$ and $K = 60$, the DNN obtains 90.58% sum-rate while achieving about **three**

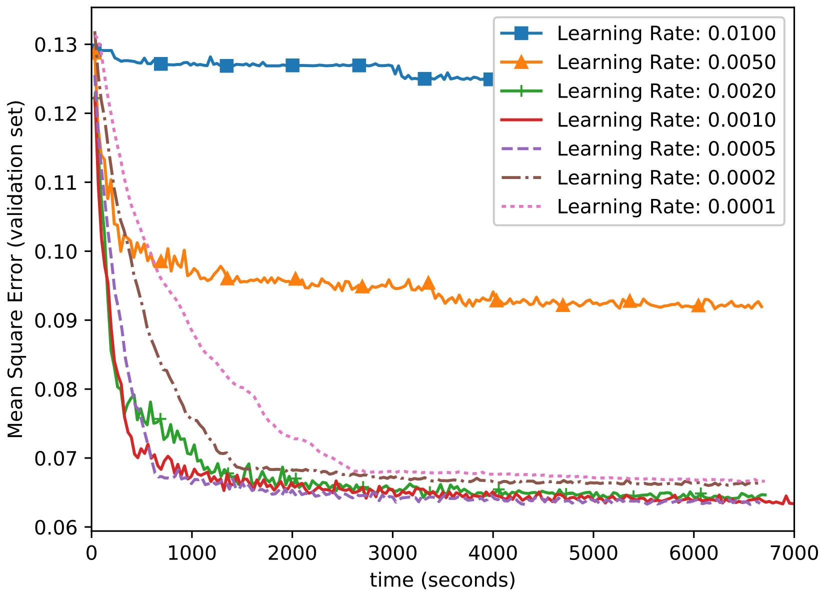


Figure 4.3 Learning rate selection for Gaussian IC, $K = 30$

**hundred times** speed up.

Table 4.1 CPU Time and Sum-Rate for Gaussian IC Channel

| | average sum-rate (bit/sec.) | | | total CPU time (sec.) | | | |
|---|---|---|---|---|---|---|---|
| # of users (K) | WMMSE | Greedy | DNN | WMMSE (Python) | WMMSE (C) | Greedy | DNN |
| 10 | 2.840 | 2.877 | 2.781 | 12.32 | 0.23 | 7.69 | 0.04 |
| 20 | 3.633 | 3.664 | 3.366 | 38.22 | 1.16 | 56.43 | 0.06 |
| 30 | 4.143 | 4.165 | 3.549 | 78.06 | 2.87 | 194.71 | 0.09 |

Additionally, in Figure 4.5 - 4.7, we show the distribution of the sum-rates over the entire test dataset. Apparently, our DNN approach gives an excellent approximation of the entire rate profile generated by the WMMSE.

Furthermore, we also include the scenario in which only $K/2$ users are present in the testing, while the DNN is trained with $K$ users (we name this case "half user" in TABLE 4.5). The purpose is to understand the generalization capability of the trained model. We observe that in this case, the DNN still performs well. This result suggests that it is not necessary to train

Table 4.2 Relative CPU Time and Sum-Rate for Gaussian IC Channel

| # of users (K) | sum-rate performance | | computational time speed up | | |
|---|---|---|---|---|---|
| | $\frac{\text{DNN}}{\text{WMMSE}}$ | $\frac{\text{DNN}}{\text{Greedy}}$ | $\frac{\text{WMMSE (Python)}}{\text{DNN (Python)}}$ | $\frac{\text{WMMSE (C)}}{\text{DNN (Python)}}$ | $\frac{\text{Greedy}}{\text{DNN}}$ |
| 10 | 97.92% | 96.66% | 308.0 | 5.8 | 192.3 |
| 20 | 92.65% | 91.87% | 637.0 | 19.3 | 940.5 |
| 30 | 85.66% | 85.21% | 867.3 | 31.9 | 2163.4 |

Table 4.3 CPU Time and Sum-Rate for IMAC Channel

| # of base stations | average sum-rate (bit/sec.) | | | | total CPU time (sec.) | | |
|---|---|---|---|---|---|---|---|
| and users (N,K) | WMMSE | DNN | RAND | MP | WMMSE(Python) | WMMSE(C) | DNN(Python) |
| $(3, 18)$ | 20.713 | 19.744 | 6.964 | 6.964 | 41.38 | 1.24 | 0.04 |
| $(3, 24)$ | 22.521 | 20.948 | 6.810 | 6.815 | 64.61 | 2.46 | 0.04 |
| $(7, 28)$ | 35.271 | 32.474 | 13.750 | 13.919 | 182.50 | 7.07 | 0.05 |
| $(3, 60)$ | 28.704 | 26.001 | 6.647 | 6.631 | 202.18 | 15.23 | 0.05 |

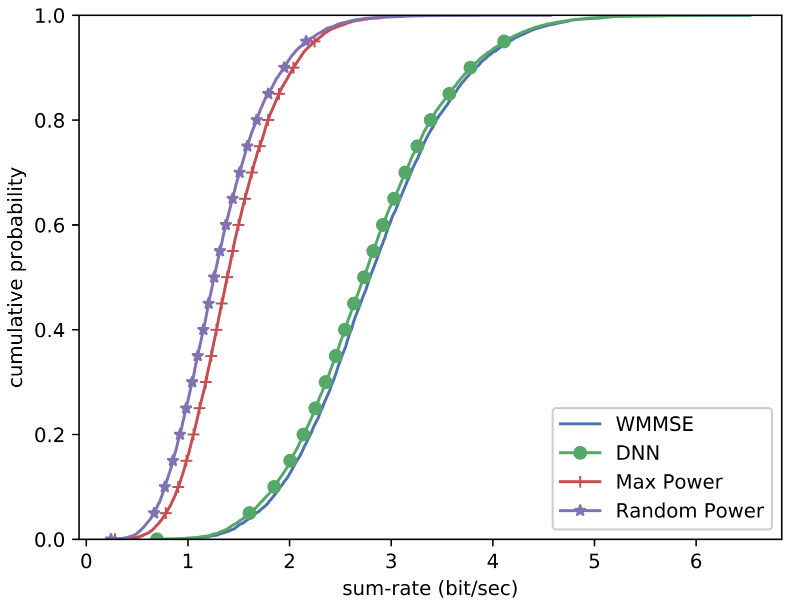
one DNN for each network configuration, which makes the proposed approach more flexible in practice. Note that in TABLE 4.5 the input size for WMMSE is reduced from $K^2$ to $K^2/4$ while the input size for DNN is still $K^2$. Therefore compared with the results in TABLE 4.1, we can observe that the computational time for WMMSE has been significantly reduced.

Table 4.4 Relative CPU Time and Sum-Rate for IMAC Channel

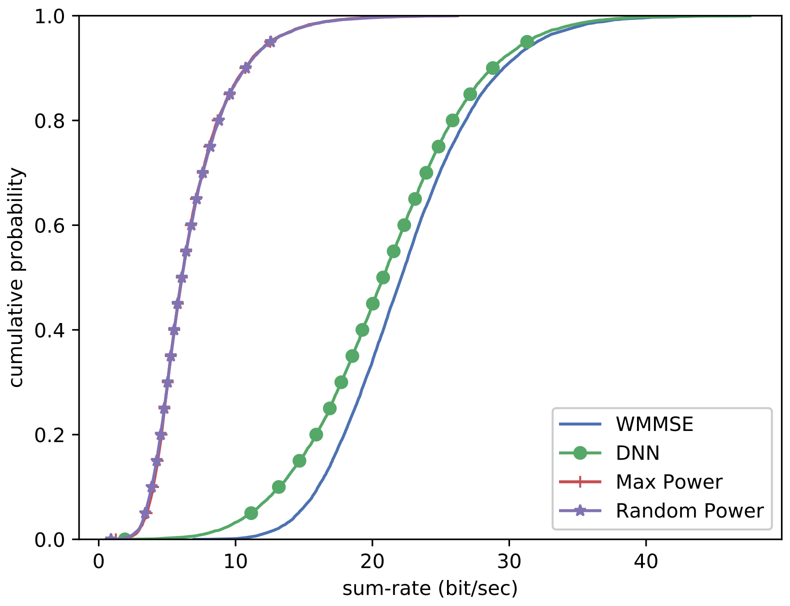
| # of base stations and users (N,K) | sum-rate performance | | | computational time speed up | |
|---|---|---|---|---|---|
| | $\frac{\text{DNN}}{\text{WMMSE}}$ | $\frac{\text{RAND}}{\text{WMMSE}}$ | $\frac{\text{MP}}{\text{WMMSE}}$ | $\frac{\text{WMMSE (Python)}}{\text{DNN (Python)}}$ | $\frac{\text{WMMSE (C)}}{\text{DNN (Python)}}$ |
| $(3, 18)$ | 95.32% | 33.62% | 32.07% | 1034 | 31 |
| $(3, 24)$ | 93.02% | 30.24% | 30.26% | 1615 | 61.5 |
| $(7, 28)$ | 92.07% | 38.98% | 39.46% | 3650 | 141.4 |
| $(3, 60)$ | 90.58% | 23.16% | 23.10% | 4044 | 304.6 |

Table 4.5 Sum-Rate and Computational Performance for Gaussian IC Channel (Half User Case)

| # of users (K) | sum-rate performance | | | computational time performance | | |
|---|---|---|---|---|---|---|
| | WMMSE | DNN | $\frac{\text{DNN}}{\text{WMMSE}}$ | WMMSE(Python) | DNN | $\frac{\text{WMMSE}}{\text{DNN}}$ |
| 10 | 2.088 | 2.071 | 99.22% | 4.16 s | 0.04 s | 113 |
| 20 | 2.811 | 2.608 | 92.78% | 12.26 s | 0.06 s | 216 |
| 30 | 3.303 | 2.899 | 87.77% | 24.13 s | 0.09 s | 256 |



(a) Gaussian IC Channel



(b) IMAC Channel

Figure 4.4 The cumulative distribution function (CDF)

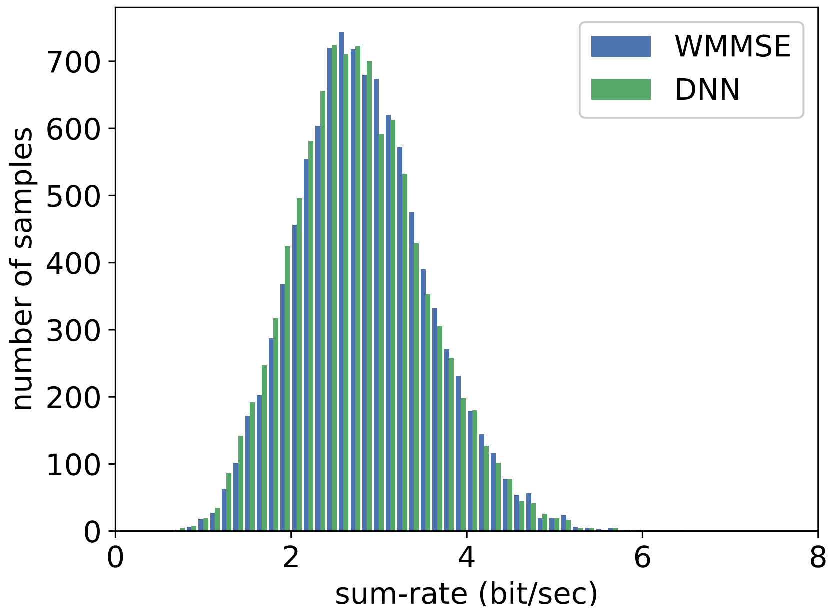


Figure 4.5 Distributions of Gaussian IC, K=10.

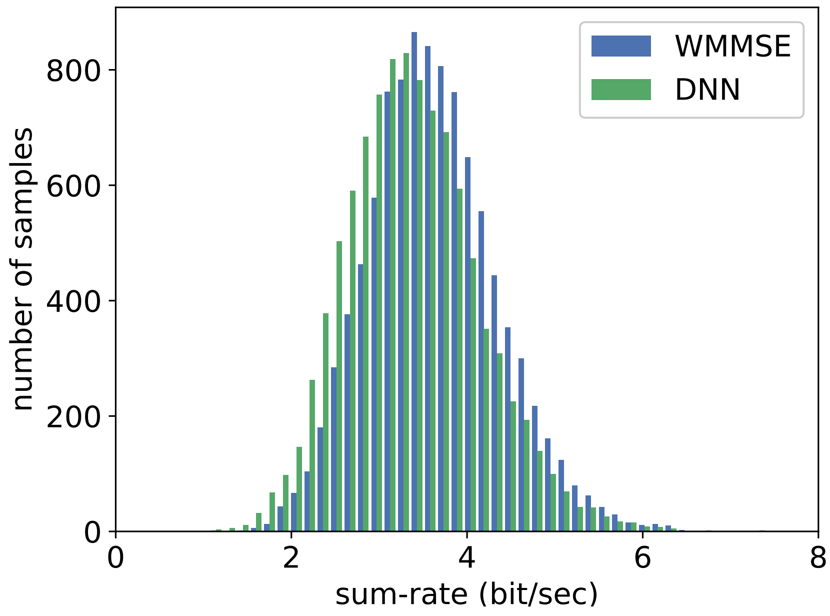


Figure 4.6 Distributions of Gaussian IC, K=20.

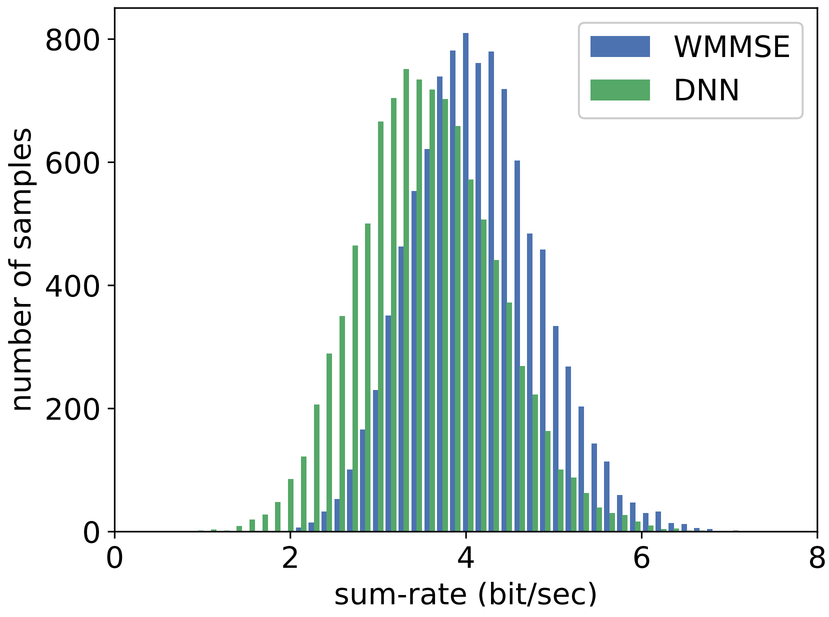


Figure 4.7 Distributions of Gaussian IC, K=30.

# CHAPTER 5. SUMMARY AND DISCUSSION

In this work, we designed a DNN based algorithm for wireless resource allocation. Our results show that, for the power control problems over either IC or IMAC channel, DNNs can well-approximate the behavior of WMMSE, leading to high sum-rate performance and low computational complexity. In many aspects, our results are very encouraging. The key message is that DNNs have great potential for real-time wireless resource allocation problems. However, the current work only represents a very preliminary step towards understanding the capability of DNNs (or related learning algorithms) for this type of problems. There are many interesting questions to be addressed in the future, and some of them are listed below:

1. Can we deal with the scenario where the testing distribution does not match the training distribution?

2. How to further reduce the computational complexity of DNN? Any theoretical justification for using deep networks?

3. How to generalize our proposed approach to more challenging problems such as beamforming for IC/IMAC?

# BIBLIOGRAPHY

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989.

Baligh, H., Hong, M., Liao, W.-C., Luo, Z.-Q., Razaviyayn, M., Sanjabi, M., and Sun, R. (2014). Cross-layer provision of future cellular networks: A WMMSE-based approach. *IEEE Signal Processing Magazine*, 31(6):56–68.

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imgaging Science*, 2(1):183 – 202.

Bertsekas, D. P. and Tsitsiklis, J. N. (1997). *Parallel and Distributed Computation: Numerical Methods, 2nd ed*. Athena Scientific, Belmont, MA.

Bjornson, E. and Jorswieck, E. (2013). Optimal resource allocation in coordinated multi-cell systems. *Foundations and Trends in Communications and Information Theory*, 9.

Coppersmith, D. and Winograd, S. (1990). Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

Delalleau, O. and Bengio, Y. (2011). Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674.

Eldan, R. and Shamir, O. (2016). The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940.

Gjendemsjo, A., Gesbert, D., Oien, G. E., and Kiani, S. G. (2008). Binary power control for sum rate maximization over multiple interfering links. *IEEE Transactions on Wireless Communications*, 7(8):3164–3173.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256.

Gregor, K. and LeCun, Y. (2010). Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 399–406.

Hershey, J. R., Roux, J. L., and Weninger, F. (2014). Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012a). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

Hinton, G., Srivastava, N., and Swersky, K. (2012b). Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online]*.

Hong, M. and Luo, Z.-Q. (2013). Signal processing and optimal resource allocation for the interference channel. In *Academic Press Library in Signal Processing*. Academic Press.

Hong, M., Sun, R., Baligh, H., and Luo, Z.-Q. (2013). Joint base station clustering and beamformer design for partial coordinated transmission in heterogenous networks. *IEEE Journal on Selected Areas in Communications.*, 31(2):226–240.

Hornik, K., Stinchcombe, M., and White, H. (1989a). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.

Hornik, K., Stinchcombe, M., and White, H. (1989b). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Kim, S.-J. and Giannakis, G. B. (2011). Optimal resource allocation for MIMO Ad Hoc Cognitive Radio Networks. *IEEE Transactions on Information Theory*, 57(5):3117 –3131.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Li, K. and Malik, J. (2016). Learning to optimize. *arXiv preprint arXiv:1606.01885*.

Liang, S. and Srikant, R. (2016). Why deep neural networks? *arXiv preprint arXiv:1610.04161*.

Liao, W.-C., Hong, M., Liu, Y.-F., and Luo, Z.-Q. (2014). Base station activation and linear transceiver design for optimal resource management in heterogeneous networks. *IEEE Transactions on Signal Processing*, 62(15):3939–3952.

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167.

Liu, Y.-F., Dai, Y.-H., and Luo, Z.-Q. (2013). Joint power and admission control via linear programming deflation. *IEEE Transactions on Signal Processing*, 61(6):1327 –1338.

Luo, Z.-Q., Ma, W.-K., So, A.-C., Ye, Y., and Zhang, S. (2010). Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine*, 27(3):20 –34.

Luo, Z.-Q. and Zhang, S. (2008). Dynamic spectrum management: Complexity and duality. *IEEE Journal of Selected Topics in Signal Processing*, 2(1):57–73.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30.

Matskani, E., Sidiropoulos, N., Luo, Z.-Q., and Tassiulas, L. (2008). Convex approximation techniques for joint multiuser downlink beamforming and admission control. *IEEE Transactions on Wireless Communications*, 7(7):2682 –2693.

Mhaskar, H., Liao, Q., and Poggio, T. (2016). Learning functions: When is deep better than shallow. *arXiv preprint arXiv:1603.00988*.

Nair, V. and Hinton, G. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Papandriopoulos, J. and Evans, J. S. (2009). SCALE: A low-complexity distributed protocol for spectrum balancing in multiuser DSL networks. *IEEE Transactions on Information Theory*, 55(8):3711–3724.

Schmidt, D., Shi, C., Berry, R., Honig, M., and Utschick, W. (2009). Distributed resource allocation schemes. *IEEE Signal Processing Magazine*, 26(5):53 –63.

Scutari, G., Palomar, D. P., and Barbarossa, S. (2008). Optimal linear precoding strategies for wideband noncooperative systems based on game theory – part I: Nash equilibria. *IEEE Transactions on Signal Processing*, 56(3):1230–1249.

Shi, Q., Razaviyayn, M., Luo, Z.-Q., and He, C. (2011). An iteratively weighted MMSE approach to distributed sum-utility maximization for a MIMO interfering broadcast channel. *IEEE Transactions on Signal Processing*, 59(9):4331–4340.

Sprechmann, P., Litman, R., Yakar, T. B., Bronstein, A. M., and Sapiro, G. (2013). Supervised sparse analysis and synthesis operators. In *Advances in Neural Information Processing Systems*, pages 908–916.

Sun, H., Chen, X., Shi, Q., Hong, M., Fu, X., and D., S. N. (2017). Learning to optimize: Training deep neural networks for wireless resource management. In *Signal Processing Advances in Wireless Communications (SPAWC), 2017 IEEE 18th International Workshop on. IEEE,*.

Telgarsky, M. (2016). Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*.

TensorFlow (2017). Deep mnist for experts. [Online]. Available: https://www.tensorflow.org/getstarted/mnist/pros.

Verdu, S. (1998). *Multiuser Detection.* Cambridge University Press, Cambridge, UK.

Yu, W. and Cioffi, J. M. (2002). FDMA capacity of Gaussian multiple-access channel with isi. *IEEE Transactions on Communications*, 50(1):102–111.

Yu, W., Ginis, G., and Cioffi, J. M. (2002). Distributed multiuser power control for digital subscriber lines. *IEEE Journal on Selected Areas in Communications*, 20(5):1105–1115.