LEARNING BASED TRANSMISSION OPTIMIZATION

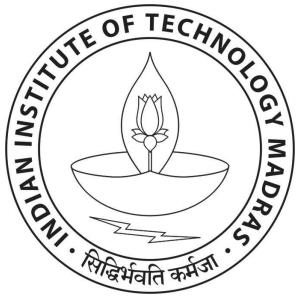
A Project Report

submitted by

ANIRUDH R

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY & MASTER OF TECHNOLOGY



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2023

THESIS CERTIFICATE

This is to certify that the thesis titled LEARNING BASED TRANSMISSION OPTIMIZATION, submitted by **Anirudh R**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor Of Technology & Master Of Technology**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Srikrishna Bhashyam
Research Guide
Professor
Dept. of Electrical Engineering
IIT Madras, 600 036

Place: Chennai

Date: May 29, 2023

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my guide, **Prof. Srikrishna Bhashyam**, for introducing me to research work and for helping me explore my interests over the past two years. This project was only possible with his continued guidance and support.

A special thanks to my friends for making my time at IIT Madras extraordinarily enjoyable and memorable. I will cherish my memories at IIT Madras for years to come.

I would also like to thank IIT Madras for these wonderful five years of my life and for giving me the opportunity to learn from admirable professors and providing all the necessary resources to support learning and creativity.

Finally, I would like to express my gratitude to my family for their unconditional love, support and encouragement throughout my studies and the process of this thesis. None of it would have been possible without them.

ABSTRACT

KEYWORDS: Communication Networks; Transmission Optimization; Machine Learning; Deep Learning; Neural Networks; Beamforming

With advancement in communication technology, we have come to use higher frequencies for data transmission (due to higher data rates) which often require a Line-Of-Sight path for successful transmission. This motivated the adoption of efficient Intelligent Reflective Surfaces (IRS) that offered an alternative path for data transmission. But, the passive nature of the IRS makes it infeasible to estimate the channels at the IRS and subsequently makes it difficult to optimize transmission. Satisfactory channel estimation often requires a huge number of pilots.

In this project, we look to perform transmission optimization entirely using neural networks within which the channel estimation happens implicitly. We avoid explicit channel estimation from the pilots. We, firstly, implement a simple Multi Layer Perceptron (MLP) to solve the optimization problem but this model is pilot length dependent. We further propose few other Recurrent Neural Network (RNN) based models that will be pilot length independent and thereby more suited to the problem. We also propose a more customized network incorporating the Unfolded Weighted MMSE beamforming algorithm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
NN	Neural Network
MLP	Multi Layer Perceptron
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
GRU	Gated Recurrent Unit
BPTT	Back Propagation Through Time
WMMSE	Weighted Minimum Mean Square Error
MMSE	Minimum Mean Square Error
PGD	Projected Gradient Descent
BS	Base Station
IRS	Intelligent Reflective Surface
CSI	Channel State Information
BF	Beamformer
DL	Deep Learning
ML	Machine Learning

NOTATION

γ	Learnable learning rate used in Unfolded WMMSE [Sub Section 5.2.1]
M	Number of antennas in the Base Station
N	Number of reflective elements in the Intelligent Reflective Surface
K	Number of users
L	Pilot length
P	Total downlink transmit power constraint

CHAPTER 1

INTRODUCTION

Technology and innovation have helped us improve at performing various tasks that are essential to everyday life. One such fundamental aspect of our lives is *communication*. From Alexander Graham Bell's first patent for the telephone in 1876 to smart phones that we use today, communication has seen vast improvements with the help of advancing technology. We started from not even being able to transmit speech clearly to having live video calls with people across the globe.

As for wireless cellular data, we progressed from the primitive 1G communication network introduced in the 1980s to the current day 5G. With newer generations of cellular communication, we have come to use higher and higher frequencies for data transmission. This is done as higher frequencies allow for higher data rates. But, like most things in life, there is no free lunch. The drawback with higher frequency communication is that they often require a Line-Of-Sight path for successful transmission. This is a huge issue particularly in cluttered environments like cities where we have a lot of buildings, vehicles and people. So, while using higher frequencies is a viable way forward from the data rate angle, finding solutions for the Line-Of-Sight issue is pivotal to making effective use of this technology.

One way of tackling this issue of a cluttered environment while using high frequency transmissions is to make use of an *Intelligent Reflective Surface (IRS)*. This effectively helps tackle the lack of a Line-Of-Sight path from Base Station (BS) to user by providing an alternate path for the transmission to reach the user. The IRS is just a reflective surface, but additionally with the ability to impart a programmable phase to a signal reflecting off it. Also, due to IRS's passive and simple structure, the IRS requires very little energy to induce the desired phase shifts onto the reflected signals, and can be flexibly integrated into common objects such as walls and ceilings. This allows easy deployment of IRSs in existing wireless communication networks.

Now, the goal will be to find the optimal beamformers and phase values for transmission, used at the BS and the IRS respectively.

In this project, we will build and implement learning based approaches to solving this problem. There are traditional optimization based approaches to solving this problem but learning provides a novel approach to this problem with its own advantages.

1.1 Overview

The report is structured as follows,

- Chapter 2 explains the system model we use and the specific problem we will try to solve.
- Chapter 3 shows some traditional methods used to solve this problem and some recent learning based approaches that have been tried.
- Chapter 4 discusses the motivations to use learning based approaches for this problem.
- Chapter 5 covers relevant theory.
- Chapter 6 discusses the various neural network architectures that we will be training and testing.
- Chapter 7 discusses the baselines that we will compare our models against.
- Chapter 8 contains the experiments conducted on the different models.

CHAPTER 2

PROBLEM SETUP

2.1 System Model

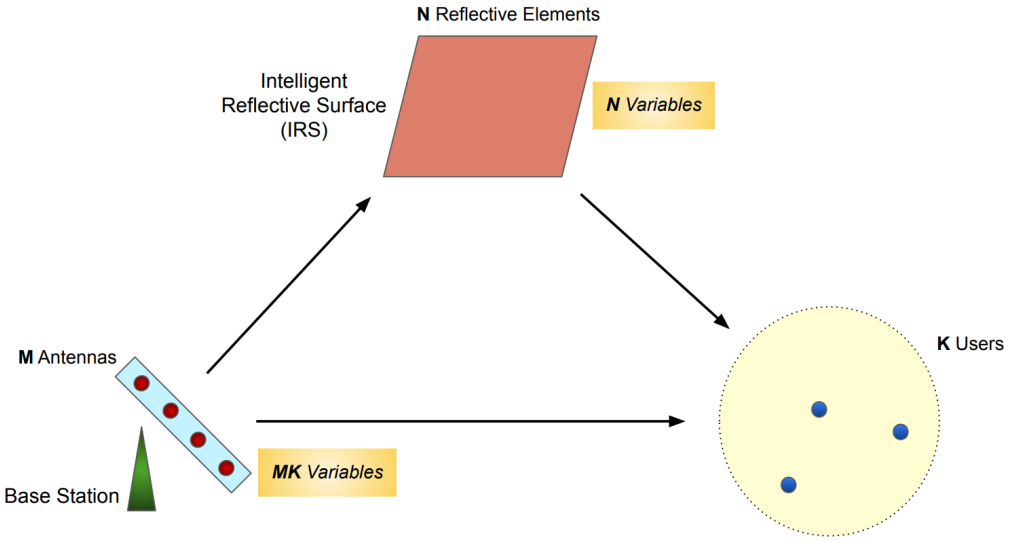


Figure 2.1: System Model

Consider an Intelligent Reflective Surface (IRS) assisted multi-user Multiple Input Single Output (MISO) system where K single-antenna users are served by a single Base Station (BS) with M antennas. The IRS equipped with N passive reflective elements is deployed between the BS and users serving as an alternative path for signals to reach from the BS to users.

Each of the N reflective elements of the IRS are controlled with an IRS controller to impart a specific phase to the incident signal. By controlling the phase imparted by the IRS, we can make the IRS work cooperatively with the BS during transmission.

Let $\boldsymbol{v} = \left[e^{j\theta_1}, e^{j\theta_2}, \cdots, e^{j\theta_N}\right]^T$ be the chosen phase shifts at the IRS, where $\theta_i \in [0, 2\pi)$ is the phase shift of the i-th reflective element in radians.

Let $\boldsymbol{G} \in \mathbb{C}^{M \times N}$ be the channel matrix between the BS and the IRS, and $\boldsymbol{h}_k^r \in \mathbb{C}^N$, $\boldsymbol{h}_k^d \in \mathbb{C}^M$ be the channel vectors between the IRS and user k, and between the BS and the user k, respectively. We can stack $\boldsymbol{h}_k^d$ and $\boldsymbol{h}_k^r$ of all the users to get $\boldsymbol{D} =$

$\left[\boldsymbol{h}_1^d, \boldsymbol{h}_2^d, \cdots, \boldsymbol{h}_K^d\right] \in \mathbb{C}^{M\times K}$ and $\boldsymbol{R} = [\boldsymbol{h}_1^r, \boldsymbol{h}_2^r, \cdots, \boldsymbol{h}_K^r] \in \mathbb{C}^{N\times K}$. With the 3 channel matrices ($\boldsymbol{G}$, $\boldsymbol{D}$ and $\boldsymbol{R}$), we can now calculate the *Effective Channel Matrix* ($\boldsymbol{H} \in \mathbb{C}^{M\times K}$) from the BS to the users as,

$$\boldsymbol{H} = \boldsymbol{D} + \boldsymbol{G}\,\mathrm{diag}(\boldsymbol{v})\boldsymbol{R} \tag{2.1}$$

2.1.1 Downlink

Let $s^k \in \mathbb{C}$ be the symbol to be transmitted from the BS to user k. The transmitted symbols are modeled as independent random variables with zero mean and unit variance. The received signal r_k at the user k is given by,

$$r_k = \sum_{i=1}^{K} \left(\boldsymbol{h}_k^d + \boldsymbol{G}\,\mathrm{diag}(\boldsymbol{v})\boldsymbol{h}_k^r\right)^H \boldsymbol{w}_i s_i + n_k \tag{2.2}$$

where $\mathrm{diag}(\boldsymbol{v})$ is the diagonal matrix of shape (N, N) formed with $\boldsymbol{v}$, $\boldsymbol{w}_k \in \mathbb{C}^M$ is the beamforming vector/beamformer (BF) at the BS corresponding to user k and $n_k \sim \mathcal{CN}(0, \sigma_0^2)$ is the additive white Gaussian noise. We can form a beamforming matrix, $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots \boldsymbol{w}_K] \in \mathbb{C}^{M\times K}$.

In this setting, we define the *Signal to Interference Noise Ratio (SINR)* for user k as,

$$\text{SINR}_k = \frac{\left|\left(\boldsymbol{h}_k^d + \boldsymbol{G}\,\mathrm{diag}(\boldsymbol{v})\boldsymbol{h}_k^r\right)^H \boldsymbol{w}_k\right|^2}{\sum_{i=1, i\neq k}^{K}\left|\left(\boldsymbol{h}_k^d + \boldsymbol{G}\,\mathrm{diag}(\boldsymbol{v})\boldsymbol{h}_k^r\right)^H \boldsymbol{w}_i\right|^2 + \sigma_0^2} \tag{2.3}$$

We can now define the rate achieved at user k as,

$$R_k = \log_2\left(1 + \text{SINR}_k\right) \tag{2.4}$$

and the sum rate for all users together will be,

$$\text{Sum Rate} = \sum_{k=1}^{K} R_k \tag{2.5}$$

2.1.2 Uplink

For dataset generation, we will use an uplink (users to BS) pilot transmission phase. Here, we will send symbols from the users and observe the output/pilots at the BS.

When all users send a symbol simultaneously, the signal observed at the BS is called a *Pilot*. During the pilot transmission phase, we will record a total of L pilots. L is called the *Pilot Length* and will be a multiple of K. The symbol sent by user k for the ℓ^{th} pilot will be $x_k(\ell)$. The ℓ^{th} received pilot $\boldsymbol{y}(\ell)$ can be expressed as,

$$\boldsymbol{y}(\ell) = \sum_{k=1}^{K} \left(\boldsymbol{h}_k^d + \boldsymbol{G}\,\mathrm{diag}(\boldsymbol{v}(\ell))\boldsymbol{h}_k^r\right) x_k(\ell) + \boldsymbol{n}(\ell), \qquad \ell = 1, \cdots, L \tag{2.6}$$

where $\boldsymbol{v}(\ell)$ is the phase shifts of IRS at time slot ℓ and $\boldsymbol{n}(\ell) \sim \mathcal{CN}(\boldsymbol{0}, \sigma_1^2 \boldsymbol{I})$ is the additive Gaussian noise. The received pilots can be stacked together to get $\boldsymbol{Y} = [\boldsymbol{y}(1), \boldsymbol{y}(2), \cdots, \boldsymbol{y}(L)] \in \mathbb{C}^{M \times L}$.

2.2 Problem Formulation

In this setup, we want to maximize the sum rate, but with some constraints. So, the constrained optimization problem we would like to solve will be,

$$\begin{aligned} \underset{\substack{\boldsymbol{W}=f(\boldsymbol{Y}) \\ \boldsymbol{v}=g(\boldsymbol{Y})}}{\text{maximize}} \quad & \textstyle\sum_{k=1}^{K} R_k(\boldsymbol{v}, \boldsymbol{W}) \\ \text{subject to} \quad & \textstyle\sum_{k=1}^{K} \|\boldsymbol{w}_k\|^2 \leq P \\ & |v_i| = 1, i = 1, 2, \cdots, N \end{aligned} \tag{2.7}$$

where P is total downlink transmit power constraint on the beamformers and v_i is the i^{th} element of $\boldsymbol{v}$. The second constraint on v_i is just to ensure that it has magnitude 1 and remains a purely phase term.

It is important here to note that the channels $\boldsymbol{G}$, $\boldsymbol{D}$ and $\boldsymbol{R}$ are not known to us. We are instead given the pilots, $\boldsymbol{Y}$. So, overall, we are given the pilots and we are required to maximize the sum rate while following some constraints.

CHAPTER 3

RELATED WORKS

A lot of existing works in this space are under the assumption that perfect channel state information (CSI) is known to the BS. With perfect CSI available, joint optimization of the IRS phase and beamformers can be done for different objectives such as minimizing power consumption at the BS [Wu and Zhang (2019)] or maximizing the minimum SINR of the users [Nadeem *et al.* (2020)] or maximizing the sum rate [Guo *et al.* (2020)].

In practice, perfect CSI will not be available. So, the *traditional approach* to transmission optimization is to first estimate the channels using pilots and then optimize the IRS phase and beamformers according to the chosen network objective.

But, channel estimation is difficult as our setup has an IRS. IRS's passive nature makes it hard to estimate the channel at the IRS.

For channel estimation, [Mishra and Johansson (2019)] proposes a binary reflection method by turning on the IRS elements one at a time and [Wei *et al.* (2021)] proposes a method based on parallel factor decomposition for estimating the BS-IRS channel and the individual IRS-user channels. However, usual practical applications require a large number of reflective elements in the IRS for sufficient gains [Björnson *et al.* (2020)]. This leads to these channel estimation methods requiring a large number of pilots.

To address this issue, [Zheng and Zhang (2019)] proposes to group the IRS elements into sub-surfaces, but at a cost of reduced beamforming capability. Also, [Chen *et al.* (2023)] proposes a compressed sensing based channel estimation method for the multi user IRS-aided system, which reduces the training overhead significantly but requires the assumption of channel sparsity. Further, [Wang *et al.* (2020)] proposes to reduce the training overhead by exploiting the common reflective channels among all the users.

Recently, *Learning based approaches* have been introduced to tackle channel estimation and beamforming [Elbir and Mishra (2022)].

For the channel estimation problem, [Liu *et al.* (2020)] proposes a deep denoising neural network to enhance the performance of the model-based compressive channel estimation for mmWave IRS systems. [Elbir *et al.* (2020)] proposes a convolutional neural network (CNN) to estimate both direct and cascaded channels from the received pilot signals through end-to-end training.

The DL based approach of learning to output the beamformers and IRS phase directly from the pilots without explicitly estimating the channel has also shown some success. In particular, [Cui *et al.* (2019)] shows that based on the geographical locations of the users, deep learning approach is able to learn the optimal scheduling without channel estimation. Location information is also utilized in [Huang *et al.* (2019)] to configure the IRS for indoor signal focusing using a deep learning approach. [Jiang *et al.* (2020)] proposes a simple Multi Layer Perceptron model to output the optimal beamformers and IRS phases. Improving on this, [Jiang *et al.* (2021)] proposes a Graph Neural Network (GNN) that better models the interference among the different users in the network. The proposed GNN is also permutation invariant that gives it better scalability and generalization ability.

CHAPTER 4

MOTIVATION

The following are the main motivations for attempting a *Learning based approach* to solving this problem:

- The IRS is a passive element. It can neither transmit nor receive signals. It can only reflect. Due to this, it is rather difficult to estimate the channels from BS to IRS and from IRS to users. IRS's passive nature requires us to make a lot more observations to estimate the channels satisfactorily.
- Practically, to achieve a significant performance improvement with this setup, the number of reflective elements in IRS, N needs to be large. This in turn further increases the size of the channels involving the IRS. So, in a practical scenario, the overhead we would have to incur to estimate the channels satisfactorily is high.
- The Machine Learning approach to solving this problem does the channel estimation implicitly as part of the network. Therefore, we won't have to take care of the estimation part separately. Also, [Jiang *et al.* (2020, 2021)] claim that such an approach significantly reduces the number of observations we will need for similar sum rate performance.
- Another issue with the 2 stage traditional approach is that there is no way to guarantee that the channel estimation method used is the best for our final objective. There may be other estimation methods that give the same channel estimation error but lead to better sum rates. But, with the ML approach, we completely leave it to the network to do the implicit channel estimation the best way possible considering our objective. As the ML model is end-to-end trained to maximize sum rate, it automatically learns to estimate the channel the best way possible for sum rate maximization.

CHAPTER 5

THEORY

5.1 Neural Networks

Neural Networks were first proposed in 1943 by Warren McCullough and Walter Pitts [McCulloch and Pitts (1943)]. They made the first computational model of a neuron to mimic the functioning of a biological neuron. It had multiple binary inputs and had to generate a binary output. It was shown that it was capable of computing many Boolean functions. This triggered a lot of research into neural networks. But, the McCullough-Pitts Neuron/Perceptron was incapable of computing the Exclusive-OR (XOR) Boolean function. This was shown by Marvin Minsky and Seymour Papert in 1969 [Minsky and Papert (2017)]. This showed the limitations of the Perceptron which led to immediate fall in serious research and funding into neural networks.

The *Back-Propagation of Errors* or the *Back-Propagation Algorithm* is an efficient application of the Leibniz chain rule to Multi Layer Perceptrons (MLP). This allows for efficient learning in a MLP. The experimental analysis of the back-propagation algorithm in 1986 by David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams [Rumelhart *et al.* (1986)] rejuvenated research into the field of MLPs again.

There has been substantial improvement in computing resources such as the GPU (Graphics Processing Unit) and TPU (Tensor Processing Unit) in the past 2 decades. With the advent of techniques to make good use of these computing resources in model training and inference [Krizhevsky *et al.* (2017)], research into Neural Networks is higher than ever. Making and using neural networks has never been easier than it is now.

Now, we will look at some neural network architectures that we will use in this project.

5.1.1 Multi Layer Perceptron (MLP)

A **Multi Layer Perceptron (MLP)** is one of the most basic Neural Network architectures available. A MLP is just a fully connected artificial feed forward neural network. This means that,

- A MLP consists of multiple layers of neurons with non-linear activation functions.
- A neuron in any layer will have the outputs from *every* neuron in the previous layer as its input. Such layers of neurons are hence called *Fully-Connected Layers*.

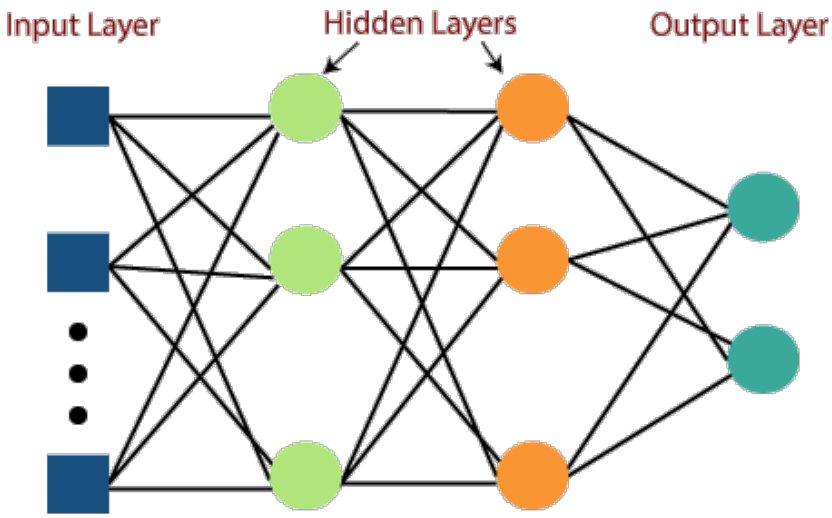


Figure 5.1: MLP, Credit: [Javatpoint (2020)]

Figure 5.1 shows the MLP architecture with an input layer and three neuronal layers (two hidden and one output layer). The input layer does not consist of neurons and is there just to represent the inputs given to the network. As the number of layers in the MLP increases, the number of hidden layers increases. A MLP may also have just 1 hidden layer, in which case, it will be a 2 layer network. Each circle in Figure 5.1 represents a neuron usually with some non-linear activation function. The non-linear activation functions are essential in allowing the network to model non-linear mappings from the input to the output and not just linear mappings. But, it is important to note that sometimes, depending on the situation, layers may not have a non-linear activation and such layers may simply be referred to as a *Linear layer*.

Each neuron has a learnable/trainable weight vector ($\boldsymbol{w}$) and bias (b) which will be modified during training with the help of the Back-Propagation algorithm.

For some neuron, let the number of neurons in the previous layer be N. Then the weight $\boldsymbol{w}$ will be of shape $(1, N)$ and the bias b will be a scalar. Let the outputs of all the neurons in the previous layer be represented as a vector $\boldsymbol{x}$ of shape $(N, 1)$. Then,

$$\text{Neuron Output} = \sigma(\boldsymbol{w}\boldsymbol{x} + b) \tag{5.1}$$

where $\sigma(.)$ will be the non-linear activation function. The non-linear activation can be changed from one neuron to another as per the user's need. It is usually taken to be one among the following,

$$\sigma(p) = \tanh(p) = \frac{1 - e^{-2p}}{1 + e^{-2p}}$$

$$\sigma(p) = \text{sigmoid}(p) = \frac{1}{1 + e^{-p}}$$

$$\sigma(p) = \text{ReLU}(p) = \max(0, p)$$

$$\sigma(p) = \text{Parametric/Leaky ReLU}(p) = \max(\alpha p, p), \quad \alpha \in [0, 1)$$

Here, ReLU is short for Rectified Linear Unit and is a very commonly used activation in the hidden layers of a MLP. Leaky ReLU is also a viable option. Depending on the task that the model is trying to solve, the activation used in the output layer may vary. For regression tasks, you may use no activation (in this case, the output layer will be a *Linear layer*) and for classification tasks, *Sigmoid* or *Softmax* are used.

It is usually the case that the activation used is same for all neurons in any given layer. This allows us to write down the computation done in the layer together. The weights of the neurons can be stacked together to form a weight matrix, $\boldsymbol{W}$ of shape (M, N) if the number of neurons in the layer is M. The scalar biases can similarly be stacked together to form a vector $\boldsymbol{B}$ of shape $(M, 1)$. If the activation used by this layer is $\sigma(.)$, we can say,

$$\text{Layer Output} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{B}) \tag{5.2}$$

The action of giving input to a network and computing its output is called the *Forward Pass*. It is common to do forward passes in batches in which case $\boldsymbol{x}$ will become a matrix of shape $(N, Batch\ Size)$. So, as we see, matrix multiplications are one of

the most fundamental operations in a MLP. This is the reason neural networks can be accelerated multi-fold with the help of GPUs that can perform matrix multiplications well.

5.1.2 Vanilla Recurrent Neural Network (Vanilla RNN)

Recurrent Neural Networks (RNN) are a family of neural networks that are commonly used for processing sequential data, such as time series data or natural language. Just like how a Convolutional Neural Network's (CNN) design is specialized for processing images and allows it to scale to images of any size with ease, the design of RNNs allow them to scale for very long sequential data naturally which may be impractical for a simple MLP. RNNs can also process input sequences of variable length.

RNNs have the concept of a **memory/state/hidden state** that allows them to *remember* what inputs have been given so far. At each time step, the state will be a function of the previous state and the current input. This also allows RNNs to extract information present in the temporal order of the input sequence (making RNNs a good fit for sequential data). At the end of the input sequence, the state can be seen as the condensed form of all the relevant information contained in the input sequence. The state information can be subsequently fed into another layer/network to make predictions (output). The format of the prediction is dependent on the task the network is trying to accomplish. The output may be a single value or a sequence in itself.

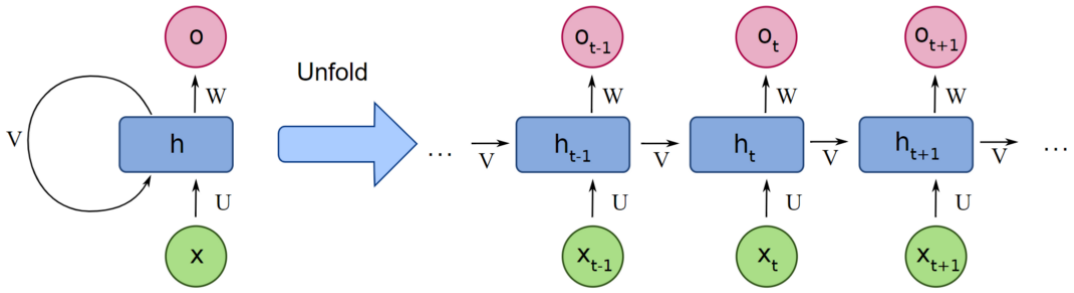


Figure 5.2: Vanilla RNN, Credit: [Kalita (2022)]

Figure 5.2 shows the *condensed form* of a Vanilla RNN on the left and the *unfolded form* on the right. The recurrence in the condensed form motivates the name for this architecture.

To understand how RNNs work, let's start with the basic equations that govern their

behavior. Suppose we have a sequence of input vectors, $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T$ with the length of the sequence being T. At each time step t, the RNN takes in the current input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$, and computes the current hidden state $\boldsymbol{h}_t$ using the following equation:

$$\boldsymbol{h}_t = f(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{h}_{t-1} + \boldsymbol{b}) \tag{5.3}$$

where f is a non-linear activation function such as the hyperbolic tangent function or the sigmoid function (Refer 5.1.1), $\boldsymbol{U}$ is the weight matrix connecting the input to the hidden state, $\boldsymbol{V}$ is the weight matrix connecting the previous hidden state to the current hidden state, and $\boldsymbol{b}$ is the bias vector. The bias vector $\boldsymbol{b}$ is not shown in the figure.

Suppose we have a single neuronal layer as the output network which computes the output at each time step taking the hidden state at that time step as the input. The output of the RNN at each time step, $\boldsymbol{o}_t$ can be written as,

$$\boldsymbol{o}_t = g(\boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{c}) \tag{5.4}$$

where g is a non-linear activation function, $\boldsymbol{W}$ is the weight matrix connecting the hidden state to the output, and $\boldsymbol{c}$ is the bias vector (not shown in the figure).

It is important to realize that the format of the output is largely dependent on the task being performed by the model. In tasks such as machine translation, the entire input sequence is first given into the network and the output sequence is only obtained after that. This is unlike Figure 5.2 where the input and output sequences are present simultaneously at each time step. In tasks such as sentiment analysis, the entire input sequence (a sentence in English) is fed into the RNN and the final hidden state alone is sent into an output network to output the sentiment in the input sequence. In this case, the output is just a category of emotion and not a sequence.

RNNs have found a variety of applications in areas such as natural language processing (NLP) and machine translation. One other common application of RNNs is language modeling, where the network is trained to predict the next word in a sentence given the previous words. This is commonly seen as part of the keyboard on our phones.

Another popular application of RNNs is in speech recognition, where the network is trained to recognize spoken words from audio input. This is seen as part of virtual assistants such as the Google assistant. RNNs can also be used in image captioning,

where the network generates a description of the image given as visual input.

RNNs are trained using the Back-Propagation algorithm, but it is instead called *Back Propagation Through Time (BPTT)*. Due to the recursive nature of the architecture, BPTT naturally suffers from two main issues during training - *Vanishing Gradients* (Gradients/derivatives become very small values which leads to trainable parameters not being updated effectively) and *Exploding Gradients* (Gradients/derivatives become very large values which makes training very unstable) [Refer Pykes (2020) for more information].

Exploding gradients can be simply solved using truncation of the gradient. But, vanishing gradients was a much harder problem to solve and was solved with the introduction of architectures such as the *Long Short-Term Memory (LSTM)* and *Gated Recurrent Unit (GRU)*.

5.1.3 Long Short Term Memory (LSTM)

Gated Recurrent Neural Networks were proposed as a solution to the vanishing gradients problem that plagued the Vanilla RNN, especially with longer input sequences. The inability to deal with longer input sequences was seen as the network being unable to *remember* over longer time frames. So, gating was introduced as a mechanism that naturally allowed the network to choose what to remember and what to forget. There have been many variants of Gated RNNs that have been proposed, but the two main architectures are LSTMs and GRUs.

The LSTM architecture [Hochreiter and Schmidhuber (1997)] contains 2 states, namely the *Hidden State* and the *Cell State*. The hidden state is meant as a short-term memory while the cell state is meant as a long-term memory. This explains the name of this architecture. The cell state helps remembering information over longer time frames and acts as a *Gradient Highway* that helps tackle the vanishing gradient problem. On the other hand, the hidden state provides short term context while processing the input sequence. The gating allows the model to control the flow of information in and out of the cell state and hidden state.

In the following equations, $\times$ represents a matrix-vector multiplication and $\odot$ represents an element-wise multiplication. For any 2 vectors $\boldsymbol{p}$ and $\boldsymbol{q}$, $[\boldsymbol{p}, \boldsymbol{q}]$ represents the

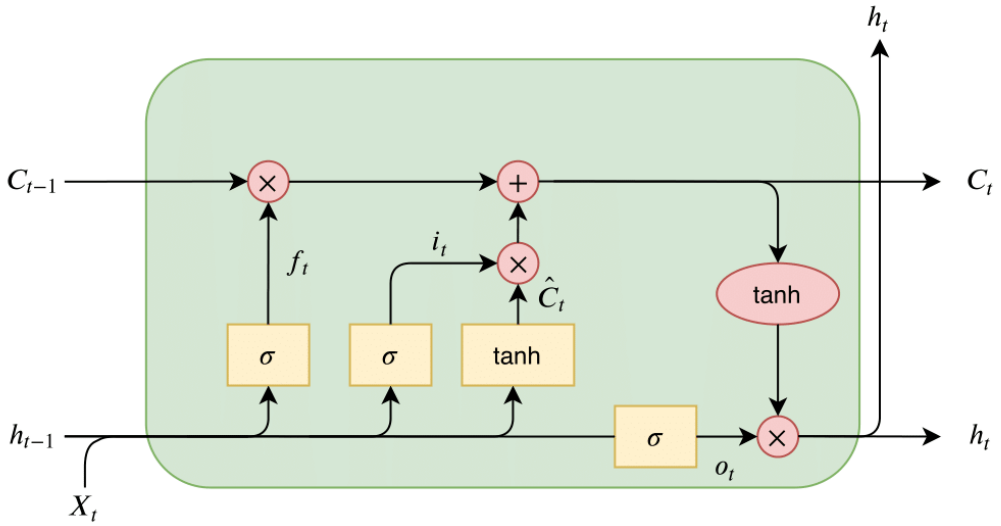


Figure 5.3: LSTM, Credit: [Ingolfsson (2021)]

concatenated vector. $\boldsymbol{X}$ represents the input, $\boldsymbol{h}$ represents the hidden state, $\boldsymbol{C}$ represents the cell state and $\sigma(.)$ represents sigmoid activation. The key equations that define the LSTM architecture are as follows,

- *Input gate:* Determines how much of the new input should be added to the cell state.

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \times [\boldsymbol{h}_{t-1}, \boldsymbol{X}_t] + \boldsymbol{b}_i) \tag{5.5}$$

- *Forget gate:* Determines how much of the previous cell state should be retained.

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \times [\boldsymbol{h}_{t-1}, \boldsymbol{X}_t] + \boldsymbol{b}_f) \tag{5.6}$$

- *Update gate:* Determines the new candidate vector to be added to the cell state.

$$\hat{\boldsymbol{C}}_t = \tanh(\boldsymbol{W}_g \times [\boldsymbol{h}_{t-1}, \boldsymbol{X}_t] + \boldsymbol{b}_g) \tag{5.7}$$

- *Cell state:* The cell state is updated based on the input, forget and update gates.

$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{C}}_t \tag{5.8}$$

- *Output gate:* Contributes to the hidden state/output of the LSTM.

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \times [\boldsymbol{h}_{t-1}, \boldsymbol{X}_t] + \boldsymbol{b}_o) \tag{5.9}$$

- *Hidden state:* The current hidden state update will be a function of the output gate and the current cell state.

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t) \tag{5.10}$$

The advantages of the LSTM architecture include:

- **Ability to learn long-term dependencies:** The presence of a long term memory (cell state) allows it to efficiently learn long term dependencies.

- **Protection against the vanishing gradient problem:** The cell state acts as a gradient highway and effectively avoids the vanishing gradients problem.

The disadvantages of the LSTM architecture include:

- **Computational complexity:** The LSTM has more parameters than a vanilla RNN, making it more computationally expensive to train and run.
- **Difficult to interpret:** The LSTM's complex architecture can make it difficult to understand how it is making decisions, which can be a challenge for applications where interpretability is important.

5.1.4 Gated Recurrent Unit (GRU)

The **Gated Recurrent Unit (GRU)** [Chung *et al.* (2014)] is a type of Gated RNN that is similar to LSTM. The GRU is designed to address the same problems as the LSTM, namely the vanishing gradient problem and the difficulty of learning long-term dependencies.

The GRU architecture includes a hidden state and a set of gates that control the flow of information into and out of the hidden state. The gates are designed to selectively update the hidden state with new information and to retain information from previous time steps. The GRU is simpler than the LSTM, with fewer parameters, making it more computationally efficient.

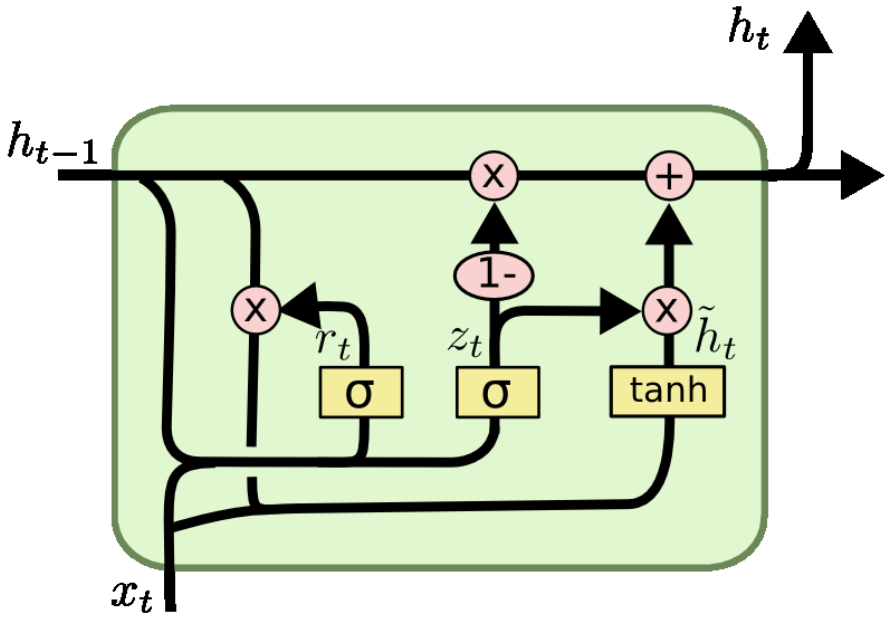


Figure 5.4: GRU, Credit: [Abdulwahab *et al.* (2017)]

In the following equations, $\times$ represents a matrix-vector multiplication and $\odot$ represents an element-wise multiplication. For any 2 vectors $\boldsymbol{p}$ and $\boldsymbol{q}$, $[\boldsymbol{p}, \boldsymbol{q}]$ represents the concatenated vector. $\boldsymbol{x}$ represents the input, $\boldsymbol{h}$ represents the hidden state and $\sigma(.)$ represents sigmoid activation. The key equations that define the GRU architecture are as follows,

- *Update gate:* Determines how much of the previous hidden state should be retained and how much of the new input should be added to the hidden state.

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z \times [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]) \tag{5.11}$$

- *Reset gate:* Determines how much of the previous hidden state should be forgotten when making the candidate hidden state.

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r \times [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]) \tag{5.12}$$

- *Candidate activation:* Determines the new candidate values for the hidden state.

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_h \times [\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t]) \tag{5.13}$$

- *Hidden state:* Hidden state is updated based on the update gate and the candidate activation.

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \tag{5.14}$$

The advantages of the GRU architecture include:

- **Computational Complexity:** The GRU is simpler than the LSTM, with fewer parameters, making it more computationally efficient.
- **Interpretability:** The GRU is easier to interpret than the LSTM due to its simpler architecture.

The disadvantages of the GRU architecture include:

- **Limited ability to learn long-term dependencies:** The GRU is less powerful than the LSTM when it comes to learning long-term dependencies which is a direct consequence of its simpler architecture.
- **Vulnerable to the vanishing gradient problem:** The GRU is still vulnerable to the vanishing gradient problem, although it is less severe than with the Vanilla RNN.

For further information on DL and neural networks, please refer Goodfellow *et al.* (2016).

5.2 Beamforming With WMMSE Algorithm

Let us analyse the setup (Figure 2.1) but without the IRS providing an alternate path from the BS to the users. To recapitulate, the BS has M antennas and there are K users.

In this altered setup, let $\boldsymbol{h}_k \in \mathbb{C}^M$ be the effective channel from the BS to user k and let $\boldsymbol{H} = [\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots \boldsymbol{h}_K]^T \in \mathbb{C}^{K \times M}$. We assume that the BS has perfect channel knowledge. Let $x_k \sim \mathcal{CN}(0, 1)$ be the symbol transmitted from the BS to user k. With linear beamforming, the received signal y_k at the user k is given by,

$$y_k = \sum_{i=1}^{K} \boldsymbol{h}_k^H \boldsymbol{v}_i x_i + n_k \tag{5.15}$$

where $\boldsymbol{v}_k \in \mathbb{C}^M$ is the beamforming vector/beamformer at the BS corresponding to user k and $n_k \sim \mathcal{CN}(0, \sigma^2)$ is the independent additive white Gaussian noise for user k. We can form a beamforming matrix, $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots \boldsymbol{v}_K]^T \in \mathbb{C}^{K \times M}$.

In this setting, we define the *Signal to Interference Noise Ratio (SINR)* and Rate achieved at user k as,

$$\text{SINR}_k = \frac{\left|\boldsymbol{h}_k^H \boldsymbol{v}_k\right|^2}{\sum_{i=1, i \neq k}^{K} \left|\boldsymbol{h}_k^H \boldsymbol{v}_i\right|^2 + \sigma^2} \tag{5.16}$$

$$R_k = \log_2 (1 + \text{SINR}_k) \tag{5.17}$$

The sum rate of all K users will be,

$$\text{Sum Rate} = \sum_{k=1}^{K} R_k \tag{5.18}$$

Now, the constrained optimization problem we would like to solve is,

$$\begin{aligned} &\max_{\boldsymbol{V}} \sum_{k=1}^{K} \log_2 (1 + \text{SINR}_k) \\ &\text{s.t. } \text{Tr}\left(\boldsymbol{V}\boldsymbol{V}^H\right) \leq P \end{aligned} \tag{5.19}$$

where P is total downlink transmit power constraint on the beamformers. The constrained optimization problem in Equation 5.19 is non-convex and has been shown to be NP-hard [Liu *et al.* (2011); Zhi-Quan Luo and Shuzhong Zhang (2008)]. Methods of finding the optimal solution to this constrained optimization problem exist, but their

high computational complexity and consequent latency make them infeasible in a practical beamforming application. Therefore, in practice, we resort to suboptimal solutions that reduce the computational complexity at the expense of performance. The WMMSE algorithm is one such iterative algorithm that converges to a local optimum and is used in practice.

The WMMSE algorithm finds a local optimum of Equation 5.19 by applying block coordinate descent to

$$\begin{aligned} &\min_{\boldsymbol{u},\boldsymbol{w},\boldsymbol{V}} \sum_{i=1}^{K} w_i e_i - \log_2 w_i \\ &\text{s.t. } \operatorname{Tr}\left(\boldsymbol{V}\boldsymbol{V}^H\right) \leq P \end{aligned} \tag{5.20}$$

which has the same optimal $\boldsymbol{V}$ as Equation 5.19 when

$$\begin{aligned} e_i &= \mathbb{E}_{x,n_i}\left\{\left|\hat{x}_i - x_i\right|^2\right\} \\ &= \sum_{j=1}^{K} \left|u_i \boldsymbol{h}_i^H \boldsymbol{v}_j\right|^2 - 2u_i \boldsymbol{h}_i^H \boldsymbol{v}_i + \sigma^2 \left|u_i\right|^2 + 1, \end{aligned} \tag{5.21}$$

where $\boldsymbol{x} \triangleq [x_1, x_2, \ldots, x_K]^T$, $\boldsymbol{x}$ and n_i are assumed to be independent, $\hat{x}_i = u_i y_i$ is the estimated data symbol at the receiver of user i, $u_i \in \mathbb{C}$ is the receiver gain of user i, $\boldsymbol{u} \triangleq [u_1, u_2, \ldots, u_K]^T$, w_i is the weight of user i, and $\boldsymbol{w} \triangleq [w_1, w_2, \ldots, w_K]^T$.

The problem in Equation 5.20 is jointly non-convex over $(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{V})$, but it is convex in each individual optimization variable $\boldsymbol{u}, \boldsymbol{w}$, and $\boldsymbol{V}$. Therefore, by iteratively optimizing over one variable while keeping the others fixed, a local optimum can be found. This procedure gives the following sequential updates:

$$\begin{aligned} u_i &= \frac{\boldsymbol{h}_i^H \boldsymbol{v}_i}{\sum_{j=1}^{K} \left|\boldsymbol{h}_i^H \boldsymbol{v}_j\right|^2 + \sigma^2} \quad \text{for } i = 1, \ldots, K, \\ w_i &= \frac{\sum_{j=1}^{K} \left|\boldsymbol{h}_i^H \boldsymbol{v}_j\right|^2 + \sigma^2}{\sum_{j=1, j\neq i}^{K} \left|\boldsymbol{h}_i^H \boldsymbol{v}_j\right|^2 + \sigma^2} \quad \text{for } i = 1, \ldots, K, \\ \boldsymbol{v}_i &= u_i w_i (\boldsymbol{A} + \mu \boldsymbol{I})^{-1} \boldsymbol{h}_i \quad \text{for } i = 1, \ldots, K, \end{aligned} \tag{5.22}$$

where

$$\boldsymbol{A} = \sum_{i=1}^{K} w_i \left|u_i\right|^2 \boldsymbol{h}_i \boldsymbol{h}_i^H, \tag{5.23}$$

and $\mu \geq 0$ is a Lagrange multiplier chosen such that the power constraint is satisfied. If

$\mu = 0$ does not satisfy the power constraint, then the optimal $\boldsymbol{V}$ must satisfy the power constraint with equality. Hence, μ can be found by solving,

$$\mathrm{Tr}\left(\boldsymbol{V}\boldsymbol{V}^H\right) = P. \tag{5.24}$$

It is important to note that finding μ involves a bisection search and an eigen decomposition. The update of $\boldsymbol{v}_i$ in Equation 5.22 also requires the inverse of the $(\boldsymbol{A} + \mu\boldsymbol{I})$ matrix.

5.2.1 Unfolded WMMSE

We will now look to unfold/unroll the above WMMSE method to represent it as a neural network. This process of unfolding iterative algorithms into neural networks is called *Deep Unfolding* [Monga *et al.* (2021)]. By unrolling, we aim to replicate the update equations in Equation 5.22 as part of our neural network. However, operations like bisection search, eigen decomposition and matrix inversion are hard to implement as neural network layers. So, we will look to change the update equation for $\boldsymbol{V}$ so that the update can be easily implemented in a neural network. The $\boldsymbol{V}$ update equation in Equation 5.22 is nothing but the solution to the following partial optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{V}} \sum_{i=1}^{K} w_i e_i - \log_2 w_i \\ &\text{s.t. } \mathrm{Tr}\left(\boldsymbol{V}\boldsymbol{V}^H\right) \leq P \end{aligned} \tag{5.25}$$

where w_i is the weight of user i and $e_i = e_i(\boldsymbol{V})$ is defined in Equation 5.21. The constrained optimization problem in Equation 5.25 can be solved using *Projected Gradient Descent (PGD)* which is an iterative first-order method. Therefore, it requires just the gradient and function values. At each iteration, we modify the optimization variable, $\boldsymbol{V}$ by taking a step along the negative gradient of the *cost function*. Then, we project the updated variable onto the feasible set defined by the constraint (This is done to ensure that the variable satisfies the constraint after the update).

We define our cost function, $f(\boldsymbol{V}) \triangleq \sum_{i=1}^{K} w_i e_i - \log_2 w_i$ and the total power constraint set can be written as, $\mathcal{C} = \left\{\boldsymbol{V} \mid \mathrm{Tr}\left(\boldsymbol{V}\boldsymbol{V}^H\right) \leq P\right\}$. The k^{th} PGD update is

given by,

$$\begin{aligned}\tilde{\boldsymbol{V}}^k &= \boldsymbol{V}^{k-1} - \gamma \nabla f\left(\boldsymbol{V}^{k-1}\right), \\ \boldsymbol{V}^k &= \Pi_{\mathcal{C}}\left\{\widetilde{\boldsymbol{V}}^k\right\},\end{aligned} \tag{5.26}$$

where $\nabla f\left(\boldsymbol{V}^k\right) = \left[\nabla f\left(\boldsymbol{v}_1^k\right), \nabla f\left(\boldsymbol{v}_2^k\right), \ldots, \nabla f\left(\boldsymbol{v}_K^k\right)\right]^T$ and $\nabla f\left(\boldsymbol{v}_i^k\right) = -2w_i u_i \boldsymbol{h}_i + 2\boldsymbol{A}\boldsymbol{v}_i^k$. $\boldsymbol{A}$ is defined in Equation 5.23 and γ is the step size. $\Pi_{\mathcal{C}}\{\}$ is the projection operator and is defined as follows,

$$\Pi_{\mathcal{C}}\{\boldsymbol{V}\} = \min_{\boldsymbol{Z}\in\mathcal{C}} \|\boldsymbol{V} - \boldsymbol{Z}\|_{\mathrm{F}} \tag{5.27}$$

Equation 5.27 can alternatively be written as,

$$\begin{aligned}\Pi_{\mathcal{C}}\{\boldsymbol{V}\} &= \begin{cases}\boldsymbol{V}, & \text{if } \mathrm{Tr}\left(\boldsymbol{V}\boldsymbol{V}^H\right) \le P \\ \frac{\boldsymbol{V}}{\|\boldsymbol{V}\|_{\mathrm{F}}}\sqrt{P}, & \text{otherwise}\end{cases} \\ \Pi_{\mathcal{C}}\{\boldsymbol{V}\} &= \frac{\boldsymbol{V}\sqrt{P}}{\max\left(0, \|\boldsymbol{V}\|_{\mathrm{F}} - \sqrt{P}\right) + \sqrt{P}}\end{aligned} \tag{5.28}$$

So, the modified WMMSE can be summarized as follows,

1. $\boldsymbol{V}$ is initialized such that the total power constraint is satisfied.
2. We update $\boldsymbol{u}$ and $\boldsymbol{w}$.
3. We perform PGD updates of $\boldsymbol{V}$ for `PGD_STEPS` iterations.
4. We repeat the above 2 steps till some convergence criterion is met or for some fixed `WMMSE_STEPS` iterations. In this project, we run for a fixed number of iterations.

With the adoption of PGD, we can now represent WMMSE as neural network layers. As we see from the steps being implemented, the only parameter that is up to our choice is the step size/learning rate (γ). γ (`PGD_STEPS`×`WMMSE_STEPS` parameters in total, one for each PGD iteration in each modified WMMSE iteration) will be the trainable parameters during neural network training. With more general neural network architectures such as MLPs or RNNs, the number of trainable parameters is often very large. But, with algorithm unfolding, the number of trainable parameters has drastically reduced as the architecture just mimics the underlying algorithm and only uses trainable parameters at the essential places.

For further information on Unfolded WMMSE, please refer Pellaco *et al.* (2022).

CHAPTER 6

ARCHITECTURES

Irrespective of what architecture we choose, it is important that the architecture satisfies some basic requirements.

- The input will be L processed pilots (each of length M) obtained during the uplink pilot transmission phase (refer Sub Section 2.1.2).
- The output of the network will be the BF for each user and θ for each reflective element in the IRS. Therefore, the output will have a total length of $MK + N$.

NOTE: For convenience, we state that the network outputs θ. But, it actually outputs $e^{j\theta}$.

Modern Deep Learning libraries such as PyTorch and TensorFlow which are commonly used for building, training and testing neural networks don't have sufficient support for complex number calculations yet. Even though some basic support for complex numbers exist, the software tools haven't matured yet. So, all of our networks will only be performing real number calculations with real-valued inputs and real-valued outputs.

The input and the output we intend to have for the network are both complex-valued. We will therefore split the input into real and imaginary parts before feeding into the network and combine the real and imaginary parts outputted by the network to form the actual complex-valued outputs. So, the size of the input and output of the network will be double the sizes stated above.

Another important feature of any architecture for our task will be the *Normalization modules*. Since, we are trying to solve a constrained optimization problem (Equation 2.7), we need to ensure that the constraints are satisfied by the output generated.

- We enforce the total power constraint by forcing the outputted beamformers to have a Frobenius norm of $\sqrt{P}$.
- We also divide the outputted θ by their magnitudes to ensure that they are a purely phase term with magnitude 1.

NOTE: The total power constraint in Equation 2.7 uses $\leq P$ but we force the outputted beamformers of the network to satisfy equality. This is not sub-optimal. In fact, if we have beamformers with Frobenius norm $< \sqrt{P}$, it can be shown that the scalar multiple of these beamformers that has Frobenius norm $\sqrt{P}$ has a better sum rate. Therefore, the optimal beamformers will have Frobenius norm of $\sqrt{P}$ and will satisfy equality in the total power constraint.

6.1 MLP-A

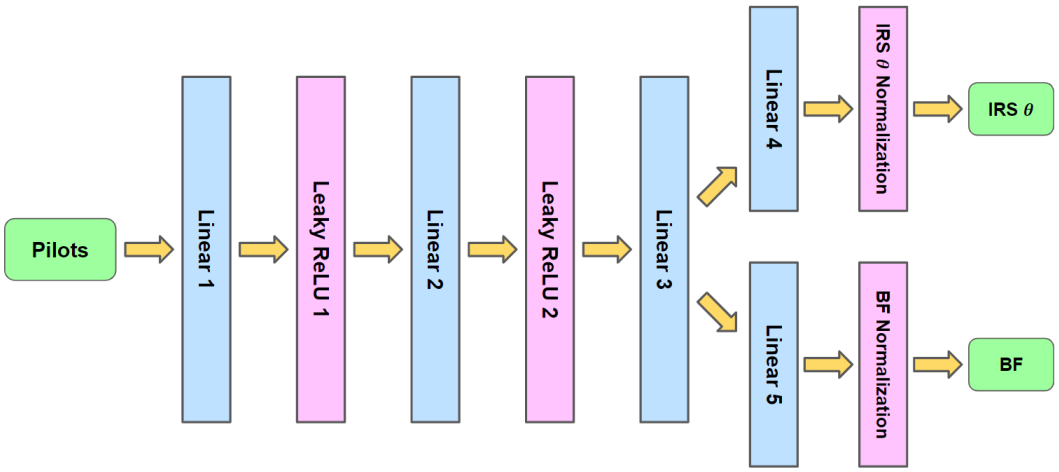


Figure 6.1: Architecture MLP-A

In this architecture, we have 3 main linear layers (`Linear 1`, `Linear 2` and `Linear 3`) with Leaky ReLU ($\alpha = 0.05$) activation between them. These 3 linear layers are taken to have the same number of neurons. `Linear 4` will have $2N$ neurons and `Linear 5` will have $2MK$ neurons. The 2 normalization blocks are present to impose the constraints. This network is just a basic MLP inspired architecture [Jiang *et al.* (2020)]. It is important to note that this model is pilot length dependent. The number of inputs this model takes changes with change in pilot length. As a result, a MLP-A model trained for one pilot length won't work for a different pilot length. This is a major drawback considering that the number of pilots we receive may change and we will have to omit or duplicate some of the received pilots to use this model. Therefore, it is better to have a model that is pilot length independent. The number of trainable parameters and model size increase with increase in pilot length.

6.2 RNN-A

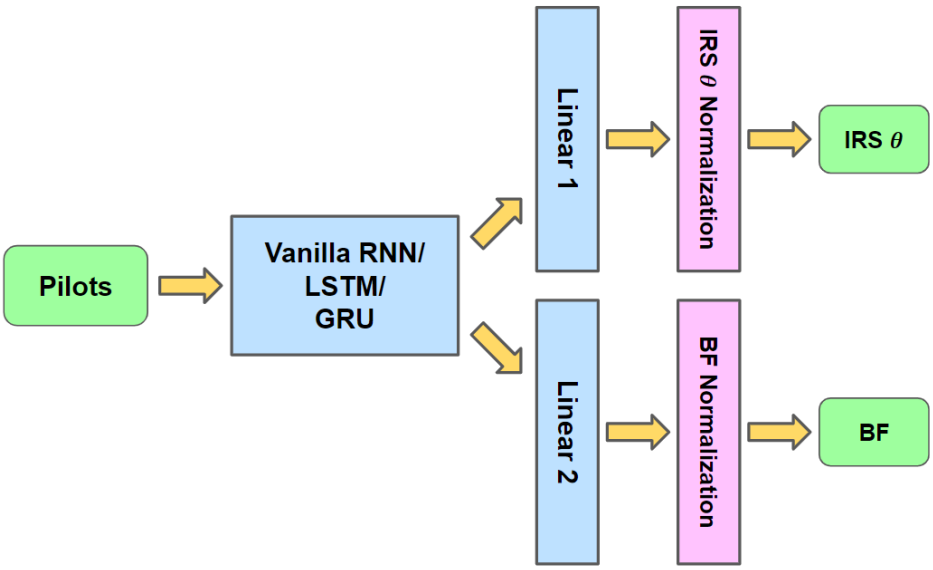


Figure 6.2: Architecture Vanilla RNN-A/LSTM-A/GRU-A

This is the exact same architecture as Figure 6.1 but we replace the first 3 linear layers and their activations with a RNN such as a Vanilla RNN, LSTM or GRU. Any one of the 3 will work. The introduction of a RNN into the architecture will now allow the model to accept input of any pilot length. We will now treat the input as a sequence of pilots (length $2M$) with L as the sequence length instead of looking at the input as a single vector of length $2ML$. Since the model is pilot length independent, the number of trainable parameters and model size remain same irrespective of the pilot length.

6.3 RNN-WMMSE-A

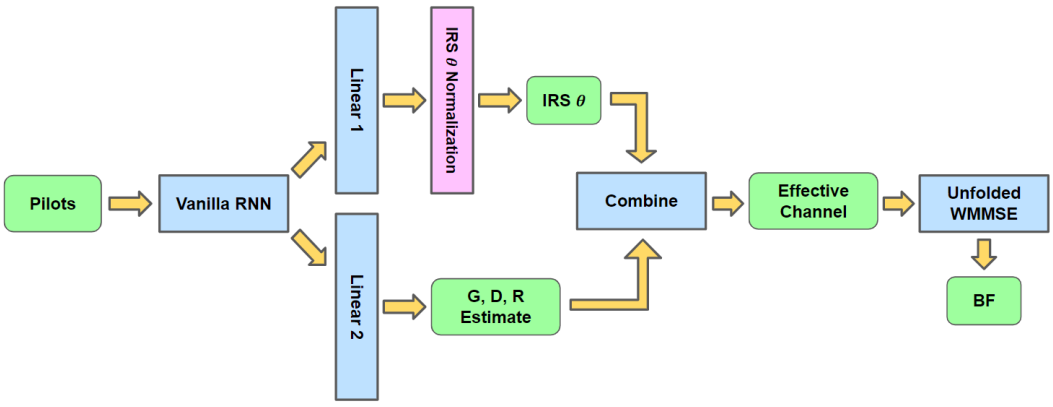


Figure 6.3: Architecture RNN-WMMSE-A

In this architecture, we carry forward the idea to use RNNs to make the architecture pilot length independent. We further modify the network to make it more suited for the task at hand. The Vanilla RNN is used to process the input pilots of any pilot length. The output of the Vanilla RNN contains the condensed information of all the pilots. This is then given to `Linear 1` to generate θ of the IRS and `Linear 2` to generate an estimate of the channels. Now that we have θ of the IRS and the channels, we can calculate the effective channel from the BS to the users using Equation 2.1.

We now employ Unfolded WMMSE to output the beamformers using the effective channel. The outputted beamformers need not be normalized as PGD already ensures that the total power constraint is followed (refer Section 5.2). But, it is important to note that PGD does not force equality in the constraint unlike the BF normalization module.

NOTE: Even though we call the output of `Linear 2` to be an estimate of the channels, it actually may be nothing close to the channels numerically. The network is only trained to give high sum rate and there are no conditions on what the output of `Linear 2` should be like. What `Linear 2` outputs is completely up to the network to choose.

In this architecture, the number of trainable parameters in the Unfolded WMMSE is very low and almost all trainable parameters are present in the linear layers and the RNN. Instead of the `Linear 2` layer, if we substitute an analytical way of estimating the channels, the number of trainable parameters will reduce even more. This is an advantage of using algorithm inspired models. With knowledge about the task at hand, we modify the network to precisely do our task and this leads to very few trainable parameters being needed. This might lead to lesser training times and smaller model sizes.

Algorithm inspired models are also explainable and reliable in their outputs unlike black box models which may badly misbehave with certain inputs [Rosebrock (2020)]. Explainability and reliability are necessary, particularly for models used in critical applications.

CHAPTER 7

BASELINES

In this chapter, we will look at some baselines that we will compare the performance of our trained models against.

7.1 Capacity

We calculate the capacity so that we have an upper bound on the sum rate. This acts as a simple sanity check to ensure that the trained model isn't showing sum rates higher than what the maximum possible sum rate is.

We calculate this baseline on the test set. Sufficient number of batches of test set are used to ensure that the capacity calculated is stable. We pass the test set batch through the trained model to obtain the θ for the IRS. Using the θ outputted by the trained model and the exact channels for the batch, we calculate the effective channel from the BS to the users (refer Equation 2.1). Once we have the effective channel, we calculate the capacity using the method described in Jindal *et al.* (2005). Note that the noise standard deviation in Jindal *et al.* (2005) is taken to be 1. But, we take downlink noise standard deviation to be σ_0. So, we divide the effective channel obtained by σ_0 before we proceed with capacity calculation as described in the paper.

For details on the water-filling algorithm required, refer Yu *et al.* (2004).

7.2 Random IRS phase, Random Beamformers

We choose the phase for the IRS, θ from $[0, 2\pi)$ uniformly randomly. We sample the beamformer from the complex normal distribution ($\mu = 0$, $\sigma = 1$). The beamformers are normalized to satisfy the total power constraint before the sum rate is calculated for this choice of θ and beamformers.

7.3 Random IRS phase, Channel Hermitian Beamformers

We choose the phase for the IRS, θ from $[0, 2\pi)$ uniformly randomly. Using the generated θ, we calculate the effective channel from the BS to the users (refer Equation 2.1). We take the beamforming matrix to be the hermitian of the downlink channel matrix. We then normalize the beamformers to satisfy the total power constraint before calculating the sum rate.

7.4 Random IRS phase, Zero Forcing Beamformers

We choose the phase for the IRS, θ from $[0, 2\pi)$ uniformly randomly. Using the generated θ, we calculate the effective channel from the BS to the users (refer Equation 2.1). We take the beamforming matrix to be the pseudo-inverse of the downlink channel matrix. We then normalize the beamformers to satisfy the total power constraint before calculating the sum rate.

NOTE 1: In the Hermitian and Zero Forcing baselines, we need to take the hermitian and pseudo-inverse of the downlink channel matrix and not the effective channel matrix. From Equation 2.2, we see that the downlink channel matrix will be the Hermitian of the effective channel matrix.

NOTE 2: Each of the last three baselines are averaged over all channels used during training and testing.

CHAPTER 8

EXPERIMENTS

Table 8.1 and Table 8.2 list the parameters and the model configurations used for the experiments respectively.

8.1 User Location Generation

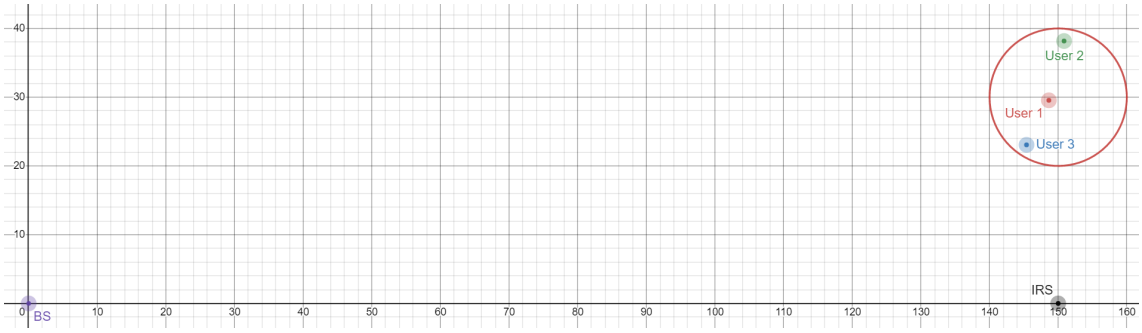


Figure 8.1: Setup used for experiments

The BS is located at $(0, 0)$ and the IRS is located at $(150, 0)$. The users are randomly uniformly chosen within a circle of radius 10 centered at $(150, 30)$. Figure 8.1 shows the exact user locations that will be used for all experiments.

8.2 Channel Generation

In order to ensure that the model performs well irrespective of channel, the model will be trained and tested with multiple channels. This ensures that the model does not overfit to one particular channel. We use a new channel for every batch of the train and test dataset. So, considering that the number of training and test samples are 2000 and the batch size is 20, we will use 100 channels each for training and testing.

The direct channel from the BS to the users is modelled as a Rayleigh Fading channel and the channels from BS to IRS and from IRS to users are modelled as Rician

Parameter	Value
Pilot lengths	[3, 15, 30, 45, 75, 105, 150]
Number of BS	1
Number of IRS	1
Number of Users	3
Number of Antennas in BS	4
Number of Reflectors in IRS	100
Uplink Noise Sigma (σ_1)	$\sqrt{1.0 * 10^{-13}}$
Downlink Noise Sigma (σ_0)	$\sqrt{3.162278 * 10^{-12}}$
Number of Train Samples	2000
Number of Test Samples	2000
Batch Size	20
New channel for each batch of Train & Test	TRUE
Learning Rate	$8 * 10^{-5}$
Number of Epochs	50(RNN-WMMSE-A), 30(else)
Optimizer	Adam
Loss Function	Negative Sum Rate
BS Location	0 + 0j
IRS Location	150 + 0j
Center around which Users are spread	150 + 30j
Radius of User spread	10
Total Downlink Power Constraint (dBm)	30
Total Uplink Power Constraint (dBm)	10
Path Loss for G (BS to IRS)	$30 + 22 * log_{10}(dist)$
Path Loss for D (BS to Users)	$32.6 + 36.7 * log_{10}(dist)$
Path Loss for R (IRS to Users)	$30 + 22 * log_{10}(dist)$

Table 8.1: Experiment Parameters

MLP-A	
Number of Neurons	100
Vanilla RNN-A, GRU-A, LSTM-A	
Number of Neurons	20
Bi-Directionality	TRUE
Non-Linearity	Tanh (Vanilla RNN-A)
RNN-WMMSE-A	
Number of Neurons	20
Bi-Directionality	TRUE
Non-Linearity	Tanh (Vanilla RNN)
WMMSE Iterations	4
PGD Iterations	4

Table 8.2: Model Parameters

Fading channels. Section 5, A of Jiang *et al.* (2021) explains in detail the steps to generate the channels.

8.3 Dataset Generation

We will use unsupervised learning to train our models. We give the pilots as input to a model and train the model to maximize sum rate calculated using the model output. We do not provide the model with a target as in supervised learning. Therefore, each sample in our dataset will only contain pilots and no target.

Suppose we want to generate a dataset for pilot length L (needs to be a multiple of K). For each sample, the L pilots are grouped into τ sub-frames of K pilots. The IRS keeps the phase shifts fixed within each sub-frame, but uses different phase shifts in different sub-frames so that both the users-to-IRS and the IRS-to-BS channels can be measured. The symbols sent by the users in each sub-frame will be exactly the same. Only IRS phase varies in different sub-frames.

In this project, in each sub-frame, we make the users send a symbol one-by-one while the other users remain quiet. In other words, only user 1 sends a symbol for pilot 1 and only user 2 sends a symbol for pilot 2 and so on. We then process the raw pilots obtained using match filtering to get the processed pilots that we will feed into the network. For our choice of user symbols, match filtering boils down to simple scaling. For further details on dataset generation, refer Section 3, A of Jiang *et al.* (2021).

NOTE: Once the entire dataset is created, we divide the dataset by the maximum absolute value in the dataset. This is done to force the values in the dataset to be between -1 and 1. This sort of normalization is necessary for effective DL model training.

8.4 Results

We conduct five independent runs where the user locations are kept same, but the channels and datasets are freshly generated. Table 8.3 contains all the results.

Model Type	Pilot Length	Train Time (s)	Test SR	Train SR	Capacity	B1 - Random	B2 - Hermitian	B3 - ZF	# Trainable Params	Model Size (MB)
MLP-A	3	36.64	6.949	6.958	9.283	1.066	3.348	2.886	45,324	0.18
	15	38.89	6.929	6.925	9.272	1.066	3.347	2.886	54,924	0.22
	30	39.26	7.255	7.204	9.307	1.065	3.347	2.884	66,924	0.27
	45	38.77	7.385	7.407	9.273	1.066	3.347	2.880	78,924	0.32
	75	38.87	6.825	6.773	9.276	1.062	3.347	2.888	102,924	0.42
	105	38.80	7.220	7.352	9.275	1.064	3.345	2.888	126,924	0.52
	150	39.53	7.379	7.556	9.259	1.066	3.347	2.888	162,924	0.66
Vanilla RNN-A	3	42.81	7.330	7.400	9.245	1.063	3.346	2.889	10,384	0.04
	15	44.02	7.366	7.351	9.282	1.063	3.346	2.888	10,384	0.04
	30	44.21	6.898	7.034	9.254	1.062	3.347	2.887	10,384	0.04
	45	45.66	6.887	6.915	9.280	1.063	3.346	2.886	10,384	0.04
	75	45.96	7.026	7.123	9.260	1.066	3.347	2.887	10,384	0.04
	105	46.85	6.772	6.644	9.283	1.062	3.346	2.886	10,384	0.04
	150	53.36	6.938	7.010	9.279	1.065	3.346	2.889	10,384	0.04
LSTM-A	3	43.01	6.897	6.959	9.298	1.064	3.345	2.889	13,984	0.06
	15	44.07	6.901	7.011	9.278	1.065	3.346	2.886	13,984	0.06
	30	45.41	7.192	7.049	9.267	1.064	3.344	2.888	13,984	0.06
	45	46.68	6.921	6.582	9.299	1.065	3.347	2.893	13,984	0.06
	75	48.73	7.053	7.052	9.267	1.063	3.348	2.888	13,984	0.06
	105	51.06	6.755	6.658	9.283	1.061	3.345	2.889	13,984	0.06
	150	52.48	7.066	7.181	9.292	1.063	3.346	2.891	13,984	0.06
GRU-A	3	43.49	6.870	6.973	9.265	1.065	3.346	2.886	12,784	0.05
	15	43.68	6.917	6.834	9.300	1.065	3.348	2.889	12,784	0.05
	30	44.88	7.047	7.139	9.254	1.064	3.348	2.890	12,784	0.05
	45	47.90	7.178	7.277	9.249	1.062	3.347	2.887	12,784	0.05
	75	48.25	6.887	6.969	9.251	1.061	3.347	2.885	12,784	0.05
	105	49.71	6.754	6.700	9.271	1.062	3.347	2.890	12,784	0.05
	150	53.65	7.203	7.306	9.278	1.061	3.347	2.888	12,784	0.05
RNN-WMMSE-A	3	313.44	5.950	6.099	8.533	1.064	3.345	2.885	67,800	0.27
	15	314.27	6.671	6.690	8.767	1.064	3.346	2.887	67,800	0.27
	30	315.63	6.167	6.094	8.731	1.064	3.345	2.886	67,800	0.27
	45	315.83	6.432	6.278	8.682	1.064	3.347	2.888	67,800	0.27
	75	321.02	6.744	6.854	8.848	1.061	3.344	2.890	67,800	0.27
	105	320.58	6.604	6.688	8.819	1.064	3.349	2.884	67,800	0.27
	150	324.56	6.885	6.911	8.956	1.064	3.345	2.889	67,800	0.27

Table 8.3: Results - Averaged over 5 runs

As expected, we see that the number of trainable parameters increases with pilot length in MLP-A while it remains constant in the other architectures. Supporting the theory discussed, we observe that the number of trainable parameters increases from Vanilla RNN-A to GRU-A to LSTM-A.

We also see that the training times for RNN-WMMSE-A is significantly longer compared to other architectures. This is perhaps a result of the architecture being completely custom for the Unfolded WMMSE module. All the other models use layers available readily as part of the PyTorch library and these will have been heavily optimized unlike our custom Unfolded WMMSE module. It may also be the case that some of the calculations used in Unfolded WMMSE can not be done in a GPU. This would force the model to use the much slower CPU to train. Unfolded WMMSE requiring 16 PGD iterations to be run sequentially may also be a major contributor to the higher training times.

We also see that the MLP-A has the lowest training times. This is probably a result of the model being capable of taking in all the pilots parallelly unlike the other RNN based models which have to take in the pilots one after another.

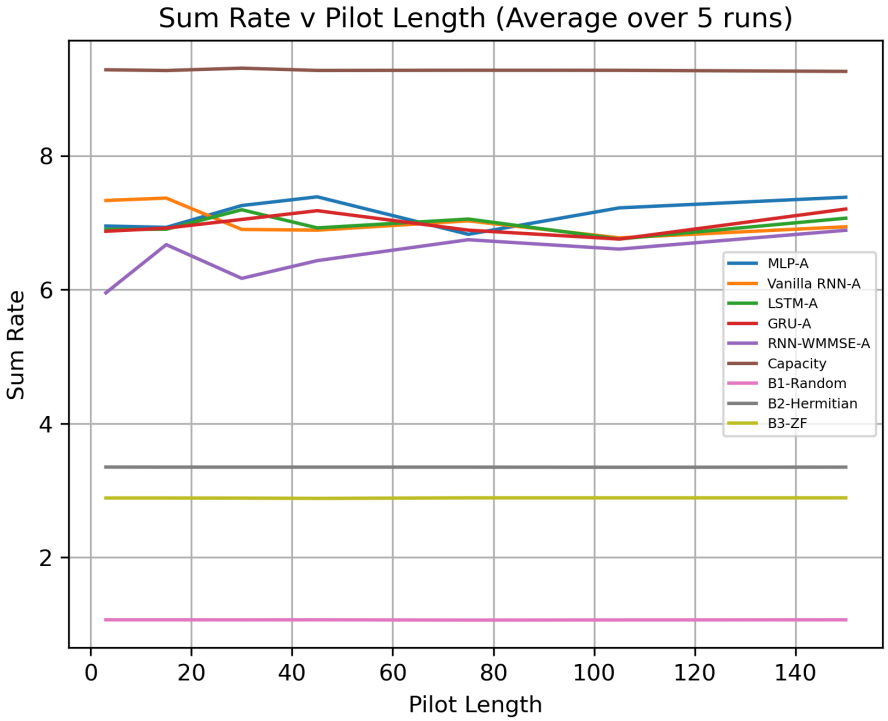


Figure 8.2: Average Test Sum Rate vs Pilot Length

Figure 8.2 plots the average test sum rate of the models for all pilot lengths along with the baselines. We see that the models perform better than all 3 baselines while

having test sum rates lower than the capacity (which is ideal). The performance of every architecture is similar. But, a rather unexpected sight is how test sum rate remains fairly constant irrespective of the pilot length. One would expect lower pilot lengths to yield lower sum rates as fewer pilots contain lesser information.

To further probe this anomaly, we tried training models with pilot length 3 and uplink transmission power of -250 dBm. The test sum rate achieved by this model was similar. This indicates an issue with some part of model training. At this uplink transmission power, the pilots will essentially just be noise. The model shouldn't be able to perform this well.

After this behaviour was observed, all parts of the code was meticulously checked by printing out values and no issue was found in the channel generation or the dataset generation or the loss function. The only possible cause of this issue could be the PyTorch library itself misbehaving. This misbehaviour maybe triggered by the usage of complex numbers in the loss function. Training models with a loss function that only uses real number calculations is the way to check this hypothesis.

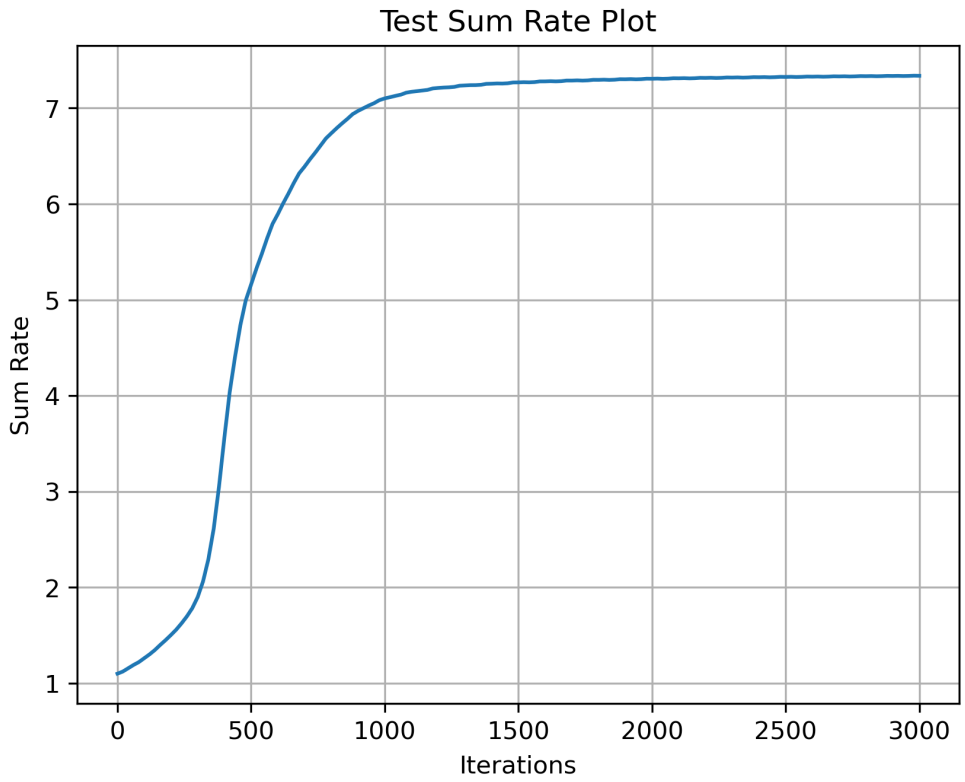


Figure 8.3: Test Sum Rate vs Iterations (Training plot) - GRU-A

NOTE: Figure 8.3 and Figure 8.4 are plotted for 3000 (30 epochs) and 5000 (50 epochs) iterations respectively.

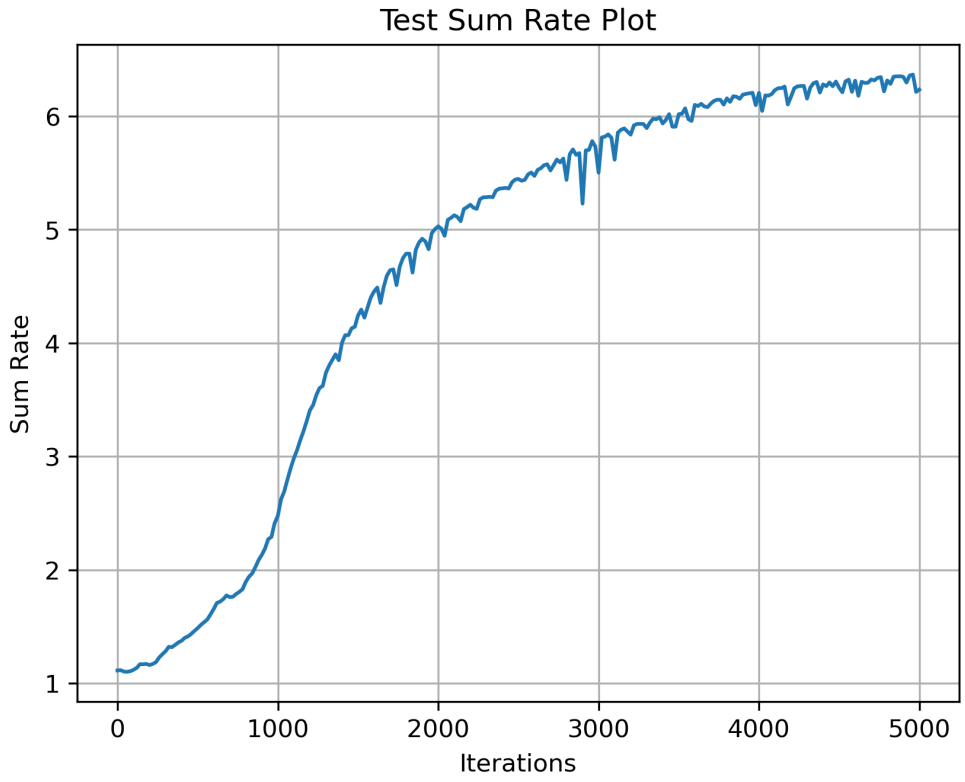


Figure 8.4: Test Sum Rate vs Iterations (Training plot) - RNN-WMMSE-A

Figure 8.3 and Figure 8.4 show the evolution of test sum rate as training progresses. We see that GRU-A learns much quicker, the test sum rate reached is higher and is overall much more stable (with no spikes in sum rate during training) throughout the training as compared to RNN-WMMSE-A. The other architectures too have training plots similar to GRU-A.

This noticeably worse training curve for RNN-WMMSE-A could be due to,

- Adam optimizer not being able to find the right step sizes for the custom trainable parameters that we added. This might be leading to the optimizer overshooting often and causing the spikes.

- The output BFs of RNN-WMMSE-A not being forced to satisfy the power constraint with equality. This may be giving the model unnecessary freedom that is making training noisier.

- Inaccuracy in the initial BF used by Unfolded WMMSE. The initial BF used by Unfolded WMMSE is chosen to satisfy the total power constraint with equality. But, numerical errors in storing the initial BF may lead to the initial BF not satisfying the required constraint. This could be tackled by creating an initial BF that satisfies the power constraint with inequality.

8.5 Future Work

The following are some possible avenues for future work,

- Implement the loss function with just real operations to check if that's causing the anomaly discussed.
- Implementing more baselines. We could add a baseline that implements the traditional approach to this problem. This will help highlight the difference in performance of learning based methods.
- Substitute the linear layer in RNN-WMMSE-A that estimates the channels with an analytical channel estimate such as the Linear MMSE (LMMSE) estimate. This will massively reduce the number of trainable parameters.
- If we really want to optimize model training and inference, we could try writing custom CUDA training and inference scripts to efficiently utilize GPUs. There also exist some dedicated libraries that try to better utilize compute resources than PyTorch.

APPENDIX A

CODE

All the code infrastructure used in this project from dataset generation to model training, testing and baselines are available at this GitHub Repository. The `README.md` file in the GitHub repository explains in detail the process of installing all the software requirements and guides the user on how to run the code.

Considering how computationally expensive model training can be, the entire code base has been implemented in a non-interactive manner to allow easy mass job deployments in compute clusters such as AQUA. Once a job is deployed, the code requires no further input from the user until completion.

REFERENCES

1. **Abdulwahab, S.**, **M. Jabreel**, and **D. Moreno** (2017). *Deep Learning Models for Paraphrases Identification*. Ph.D. thesis.

2. **Björnson, E.**, **Özdogan**, and **E. G. Larsson** (2020). Intelligent Reflecting Surface vs. Decode-and-Forward: How Large Surfaces Are Needed to Beat Relaying? *IEEE Wireless Communications Letters*, **9**(2), 244–248. ISSN 2162-2337, 2162-2345. URL http://arxiv.org/abs/1906.03949. ArXiv:1906.03949 [cs, math].

3. **Chen, J.**, **Y.-C. Liang**, **H. V. Cheng**, and **W. Yu** (2023). Channel Estimation for Reconfigurable Intelligent Surface Aided Multi-User mmWave MIMO Systems. URL http://arxiv.org/abs/1912.03619. ArXiv:1912.03619 [eess].

4. **Chung, J.**, **C. Gulcehre**, **K. Cho**, and **Y. Bengio** (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. URL http://arxiv.org/abs/1412.3555. ArXiv:1412.3555 [cs].

5. **Cui, W.**, **K. Shen**, and **W. Yu** (2019). Spatial Deep Learning for Wireless Scheduling. *IEEE Journal on Selected Areas in Communications*, **37**(6), 1248–1261. ISSN 0733-8716, 1558-0008. URL http://arxiv.org/abs/1808.01486. ArXiv:1808.01486 [cs, eess, math].

6. **Elbir, A. M.** and **K. V. Mishra** (2022). A Survey of Deep Learning Architectures for Intelligent Reflecting Surfaces. URL http://arxiv.org/abs/2009.02540. ArXiv:2009.02540 [cs, eess, math].

7. **Elbir, A. M.**, **A. Papazafeiropoulos**, **P. Kourtessis**, and **S. Chatzinotas** (2020). Deep Channel Learning For Large Intelligent Surfaces Aided mm-Wave Massive MIMO Systems. *IEEE Wireless Communications Letters*, **9**(9), 1447–1451. ISSN 2162-2337, 2162-2345. URL http://arxiv.org/abs/2001.11085. ArXiv:2001.11085 [cs, eess, math].

8. **Goodfellow, I.**, **Y. Bengio**, and **A. Courville**, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

9. **Guo, H.**, **Y.-C. Liang**, **J. Chen**, and **E. G. Larsson** (2020). Weighted Sum-Rate Maximization for Reconfigurable Intelligent Surface Aided Wireless Networks. *IEEE Transactions on Wireless Communications*, **19**(5), 3064–3076. ISSN 1558-2248. Conference Name: IEEE Transactions on Wireless Communications.

10. **Hochreiter, S.** and **J. Schmidhuber** (1997). Long Short-Term Memory. *Neural Computation*, **9**(8), 1735–1780. ISSN 0899-7667. URL https://doi.org/10.1162/neco.1997.9.8.1735.

11. **Huang, C.**, **G. C. Alexandropoulos**, **C. Yuen**, and **M. Debbah** (2019). Indoor Signal Focusing with Deep Learning Designed Reconfigurable Intelligent Surfaces. URL http://arxiv.org/abs/1905.07726. ArXiv:1905.07726 [cs, math].

12. **Ingolfsson, T. M.** (2021). Insights into LSTM architecture. URL https://thorirmar.com/post/insight_into_lstm/.

13. **Javatpoint** (2020). Multi-layer Perceptron in TensorFlow - Javatpoint. URL https://www.javatpoint.com/multi-layer-perceptron-in-tensorflow.

14. **Jiang, T.**, **H. V. Cheng**, and **W. Yu**, Learning to Beamform for Intelligent Reflecting Surface with Implicit Channel Estimate. *In GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. 2020. ISSN: 2576-6813.

15. **Jiang, T.**, **H. V. Cheng**, and **W. Yu** (2021). Learning to Reflect and to Beamform for Intelligent Reflecting Surface With Implicit Channel Estimation. *IEEE Journal on Selected Areas in Communications*, **39**(7), 1931–1945. ISSN 1558-0008. Conference Name: IEEE Journal on Selected Areas in Communications.

16. **Jindal, N.**, **W. Rhee**, **S. Vishwanath**, **S. Jafar**, and **A. Goldsmith** (2005). Sum power iterative water-filling for multi-antenna Gaussian broadcast channels. *IEEE Transactions on Information Theory*, **51**(4), 1570–1580. ISSN 1557-9654. Conference Name: IEEE Transactions on Information Theory.

17. **Kalita, D.** (2022). A Brief Overview of Recurrent Neural Networks (RNN). URL https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/.

18. **Krizhevsky, A.**, **I. Sutskever**, and **G. E. Hinton** (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, **60**(6), 84–90. ISSN 0001-0782. URL https://dl.acm.org/doi/10.1145/3065386.

19. **Liu, S.**, **Z. Gao**, **J. Zhang**, **M. Di Renzo**, and **M.-S. Alouini** (2020). Deep Denoising Neural Network Assisted Compressive Channel Estimation for mmWave Intelligent Reflecting Surfaces. URL http://arxiv.org/abs/2006.02201. ArXiv:2006.02201 [cs, eess, math].

20. **Liu, Y.-F.**, **Y.-H. Dai**, and **Z.-Q. Luo** (2011). Coordinated Beamforming for MISO Interference Channel: Complexity Analysis and Efficient Algorithms. *IEEE Transactions on Signal Processing*, **59**, 1142–1157.

21. **McCulloch, W. S.** and **W. Pitts** (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, **5**(4), 115–133. ISSN 1522-9602. URL https://doi.org/10.1007/BF02478259.

22. **Minsky, M.** and **S. A. Papert**, *Perceptrons, Reissue of the 1988 Expanded Edition with a new foreword by Léon Bottou: An Introduction to Computational Geometry*. MIT Press, 2017. ISBN 978-0-262-53477-2. Google-Books-ID: PLQ5DwAAQBAJ.

23. **Mishra, D.** and **H. Johansson**, Channel Estimation and Low-complexity Beamforming Design for Passive Intelligent Surface Assisted MISO Wireless Energy Transfer. *In ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019. ISSN: 2379-190X.

24. **Monga, V.**, **Y. Li**, and **Y. C. Eldar** (2021). Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing. *IEEE Signal Processing Magazine*, **38**(2), 18–44. ISSN 1558-0792. Conference Name: IEEE Signal Processing Magazine.

25. **Nadeem, Q.-U.-A.**, **H. Alwazani**, **A. Kammoun**, **A. Chaaban**, **M. Debbah**, and **M.-S. Alouini** (2020). Intelligent Reflecting Surface Assisted Multi-User MISO Communication: Channel Estimation and Beamforming Design. URL http://arxiv.org/abs/2005.01301. ArXiv:2005.01301 [cs, math].

26. **Pellaco, L.**, **M. Bengtsson**, and **J. Jaldén** (2022). Matrix-Inverse-Free Deep Unfolding of the Weighted MMSE Beamforming Algorithm. *IEEE Open Journal of the Communications Society*, **3**, 65–81. ISSN 2644-125X. Conference Name: IEEE Open Journal of the Communications Society.

27. **Pykes, K.** (2020). The Vanishing/Exploding Gradient Problem in Deep Neural Networks. URL https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-19135

28. **Rosebrock, A.** (2020). Adversarial images and attacks with Keras and TensorFlow. URL https://pyimagesearch.com/2020/10/19/adversarial-images-and-attacks-with-keras-and-tensorflow/.

29. **Rumelhart, D. E.**, **G. E. Hinton**, and **R. J. Williams** (1986). Learning representations by back-propagating errors. *Nature*, **323**(6088), 533–536. ISSN 1476-4687. URL https://www.nature.com/articles/323533a0. Number: 6088 Publisher: Nature Publishing Group.

30. **Wang, Z.**, **L. Liu**, and **S. Cui** (2020). Channel Estimation for Intelligent Reflecting Surface Assisted Multiuser Communications: Framework, Algorithms, and Analysis. URL http://arxiv.org/abs/1912.11783. ArXiv:1912.11783 [cs, eess, math].

31. **Wei, L.**, **C. Huang**, **G. C. Alexandropoulos**, **C. Yuen**, **Z. Zhang**, and **M. Debbah** (2021). Channel Estimation for RIS-Empowered Multi-User MISO Wireless Communications. *IEEE Transactions on Communications*, **69**(6), 4144–4157. ISSN 0090-6778, 1558-0857. URL http://arxiv.org/abs/2008.01459. ArXiv:2008.01459 [cs, eess, math].

32. **Wu, Q.** and **R. Zhang** (2019). Intelligent Reflecting Surface Enhanced Wireless Network via Joint Active and Passive Beamforming. *IEEE Transactions on Wireless Communications*, **18**(11), 5394–5409. ISSN 1558-2248. Conference Name: IEEE Transactions on Wireless Communications.

33. **Yu, W.**, **W. Rhee**, **S. Boyd**, and **J. Cioffi** (2004). Iterative Water-Filling for Gaussian Vector Multiple-Access Channels. *Information Theory, IEEE Transactions on*, **50**, 145–152.

34. **Zheng, B.** and **R. Zhang** (2019). Intelligent Reflecting Surface-Enhanced OFDM: Channel Estimation and Reflection Optimization. *IEEE Wireless Communications Letters*, **PP**, 1–1.

35. **Zhi-Quan Luo** and **Shuzhong Zhang** (2008). Dynamic Spectrum Management: Complexity and Duality. *IEEE Journal of Selected Topics in Signal Processing*, **2**(1), 57–73. ISSN 1932-4553, 1941-0484. URL http://ieeexplore.ieee.org/document/4453890/.