Tuning of Optical Beamforming Networks: A Deep Learning Approach

Herminarto Nugroho
Tuning of Optical Beamforming Networks: A Deep Learning Approach

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Systems and Control at Delft University of Technology

Herminarto Nugroho

October 13, 2015

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology
In aircraft-satellite communications, aircrafts receive signals from satellites. The conventional solution is to steer a dish antennas mechanically toward satellites. Optical Beamforming Networks (OBFNs) rely instead on many small and flat antennas called Phased Array Antennas (PAAs) that are coordinated in order to receive signals from specific angles. In this project, a new type of OBFN proposed by Meijerink et al. [1] will be considered. The tuning of these OBFNs such that the beam is steered correctly is a difficult nonlinear problem that, so far, has only been addressed with off-the-shelf solvers in very small setups.

The problem of tuning a large-scale OBFN is in many aspects similar to training a neural network with many layers. For the latter problem, many recent advances have been made under the umbrella term deep learning. The objective of this graduation project is to explore if and how advances in deep learning can be exploited to tune large-scale OBFNs. This can be achieved either by formulating the tuning problem as a learning problem, or by modifying existing algorithms in the area of deep learning. The tuning methodology developed in this project will be based on feedback that can be easily measured in real systems. The purpose is that in the future, online tuning, which needs feedbacks that can be measured, will be used. Training data, which consists of full signals, can be measured, and therefore will be used in this project. For pilot signals, the desired signal is known explicitly.

The content of this thesis is divided into two distinct parts. In the first part, we analyze the OBFN setup and tuning algorithm proposed by Blokpoel [2] and Meijerink et al. [1]. The analysis covers the whole system, including antenna elements (AEs), OBFNs based on binary tree topology, optical ring resonators (ORRs), as well as the ORR’s parameters and their influence to the group delay response of the signal. Then, a way to exploit that special structure of OBFNs into deep neural network representation will follow afterwards. The second part covers all breakthroughs of deep learning including all developments that are useful for tuning OBFNs, such as auto-encoders, momentum, and dropout regularization. Furthermore, the implementation of those breakthroughs and the results of tuning large-scale OBFNs using deep learning approach are presented in last chapters.

It is expected that deep learning offers new approaches for tuning large scale OBFNs. Non-linear Programming (NLP) which currently is the standard solution to tune OBFNs, can
only be applied to small scale OBFNs. Given the configuration of large-scale OBFNs and all nominal parameters required, it was verified in simulation experiments that the deep learning approach can be used to tune large-scale OBFNs for any desired delays.
Preface xv

1 Introduction 1
  1-1 Goals of the Thesis ................................. 2
  1-2 Research Approach .................................. 2
  1-3 Outline ............................................. 3
  1-4 Nomenclature ....................................... 3

2 Phased Array Antennas and Optical Ring Resonators: An Overview 5
  2-1 Phased Array Antennas ............................... 6
  2-2 Optical Ring Resonators ............................ 8
    2-2-1 Structure of the Optical Ring Resonator ....... 8
    2-2-2 Mathematical Model of the Optical Ring Resonator 9
    2-2-3 Cascade of Multiple ORRs ..................... 13

3 Principles of Ring Resonator-Based Optical Beamforming Networks 17
  3-1 Optical Beamforming Network Structure .............. 17
    3-1-1 Parallel-structured Optical Beamforming Networks 17
    3-1-2 Binary-tree-structured Optical Beamforming Networks 18
  3-2 Optical Phase Synchronization ...................... 20
  3-3 Derivation of the Optimality Criteria ............... 23
    3-3-1 Group Delay Criterion ........................... 23
    3-3-2 Phase Criterion ................................ 24
    3-3-3 Power Criterion ................................ 24
  3-4 Current Optimization Method ....................... 25
Table of Contents

4 Principles of Deep Neural Networks 27
4-1 Common Types of Neural Networks ........................................... 29
  4-1-1 Feedforward Neural Networks ........................................ 29
  4-1-2 Recurrent Neural Networks ........................................... 29
  4-1-3 Convolutional Networks ............................................ 30
4-2 Training Neural Networks using Gradient Descent ......................... 31
  4-2-1 The Concept of Backpropagation .................................... 31
  4-2-2 Some Breakthroughs in Deep Learning ............................... 35
4-3 Deep Belief Networks .......................................................... 36
  4-3-1 Restricted Boltzmann Machine ....................................... 36
4-4 Auto-encoder ................................................................. 38
4-5 Dropout Regularization .......................................................... 40

5 Deep Neural Network Representations of Optical Beamforming Networks 43
5-1 Neural Network Configurations for OBFN Systems .......................... 43
  5-1-1 Feed-forward Neural Network-based OBFN Systems ................ 43
  5-1-2 Generating Training Examples ....................................... 46
  5-1-3 The Formulation of Weight Matrices ................................ 47
  5-1-4 The Non-linear Optimization ........................................ 49
  5-1-5 Stochastic Gradient Projection Method ............................... 50
  5-1-6 The Backpropagation Algorithm and The Derivative of the Frequency
             Response with respect to Parameters ............................. 51
5-2 Separation of Complex Numbers .............................................. 53
  5-2-1 Separation of Complex Numbers in Training Examples ............... 54
  5-2-2 Complex Numbers Separation in Weight Matrices .................... 55
  5-2-3 The Effect of Complex Numbers Separation on Backpropagation .... 55
5-3 Numerical Results ............................................................... 57
  5-3-1 Simulation Setup ........................................................ 57
  5-3-2 Simulation Result of a $4 \times 1$ OBFN for a Certain Desired Delay .... 58
  5-3-3 Simulation Result of a $4 \times 1$ OBFN for Several Desired Delays .... 59
  5-3-4 Simulation Result of a $4 \times 1$ OBFN for Some Different Initializations .. 59

6 The Implementation of Some Breakthroughs in Deep Learning 63
6-1 Momentum ........................................................................ 63
  6-1-1 The Importance of Momentum ........................................ 64
  6-1-2 The Effect of Momentum for Tuning OBFNs .......................... 64
6-2 Auto-encoder ..................................................................... 66
  6-2-1 Pre-training OBFN Systems using Auto-encoder ..................... 66
  6-2-2 Identity Network Problem ............................................. 66
6-3 Dropout Regularization .......................................................... 69
  6-3-1 Dropping Procedure of OBFN Systems ............................... 69
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-3-3</td>
<td>Stochastic Gradient Projection Method and Random Picking Frequencies Function</td>
<td>119</td>
</tr>
<tr>
<td>B-3-4</td>
<td>Creating Mask Matrices Function</td>
<td>120</td>
</tr>
<tr>
<td>B-3-5</td>
<td>Creating Weight Matrices Function</td>
<td>121</td>
</tr>
<tr>
<td>B-3-6</td>
<td>Backpropagation Algorithm Function</td>
<td>121</td>
</tr>
<tr>
<td>B-3-7</td>
<td>Constrained Backpropagation Function</td>
<td>122</td>
</tr>
<tr>
<td>B-3-8</td>
<td>Derivative with respect to Parameters Function</td>
<td>123</td>
</tr>
<tr>
<td>B-3-9</td>
<td>Group Delay Error Function</td>
<td>125</td>
</tr>
<tr>
<td>B-3-10</td>
<td>Plot Delay Responses Function</td>
<td>126</td>
</tr>
<tr>
<td>B-4</td>
<td>Non-linear Programming (NLP) Scripts</td>
<td>127</td>
</tr>
<tr>
<td>B-4-1</td>
<td>Main Script for Tuning using NLP</td>
<td>127</td>
</tr>
<tr>
<td>B-4-2</td>
<td>Optimization Scripts for Different Paths</td>
<td>127</td>
</tr>
<tr>
<td>B-4-3</td>
<td>Cost Functions for Different Paths</td>
<td>129</td>
</tr>
</tbody>
</table>

**Glossary**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>List of Acronyms</td>
<td>137</td>
</tr>
</tbody>
</table>
# List of Figures

2-1 Illustration of the aeronautic communication system [3]. ........................................ 5  
2-2 A conventional solution for aeronautic communication systems: a mechanically steered antenna [4], [5]. ................................................................. 6  
2-3 Illustration of Phased Array Antenna [4]. ................................................................. 6  
2-5 Schematic of a rectangular $M \times N$ planar array antenna [3] ................................. 7  
2-6 Beamformer illustration with $N$ inputs using intensity modulation (IM) and direct optical detection. LNA = low-noise amplifier [1]. ................................. 8  
2-7 Structure of a $1 \times 1$ single-stage ORR [3] .............................................................. 9  
2-8 An ORR formed by a $2 \times 2$ coupler and a feedback waveguide and its $Z-$ transform schematic [3]. ................................................................. 9  
2-9 Group delay responses of a single ORR for different values of $\kappa$ [3] .................. 12  
2-10 Definition of delay bandwidth of a single ORR [3] ................................................ 12  
2-11 Cascade of multiple ORRs [3]. ............................................................................. 13  
2-12 Group delay response of a cascade of three ORRs [3] ........................................... 14  
2-13 Illustration of squeezing method to reduce delay ripple of a cascade of 2 ORRs [3]. 14  
2-14 The simulation results of the tradeoffs [3]. .......................................................... 15  
3-1 Schematic of an $4 \times 1$ OBFN with symmetric parallel structure [3] ...................... 18  
3-2 Schematic of an $4 \times 1$ OBFN with asymmetric parallel structure [3] ................... 18  
3-3 Schematic of an $4 \times 1$ OBFN with symmetric binary tree structure [3] .............. 19  
3-4 Schematic of an $4 \times 1$ OBFN with asymmetric binary tree structure [3] .......... 19  
3-5 Schematic of an $8 \times 1$ OBFN with asymmetric binary-tree structure [3] ............ 19
3-6 Spectrum of the modulated signal (solid line) in case of: (a) optical intensity modulation (IM), (b) optical single-sideband suppressed-carrier (SSB-SC) modulation. The suppressed sideband and optical carrier are represented by the doted line. Corresponding desired group delay of a delay elements in the OBFN is represented by a dashed line. $f_0$ specifies the optical carrier frequency, $f_{\text{min}}$ and $f_{\text{max}}$ specify the lower and upper bound of the frequency range of the modulating signal respectively [1].

3-7 $8 \times 1$ OBFN configuration with extra phase shifters inserted in the upper branch [6].

3-8 $8 \times 1$ OBFN configuration with extra phase shifters inserted in front of every path. Adapted from [6].

4-1 Non-linear model of a neuron. Adapted from [7].

4-2 Common types of activation function. Adapted from [8].

4-3 Simple neural network example.

4-4 Comparison of feedforward and recurrent neural network. Adapted from [9].

4-5 Sparse connectivity and shared weight illustration [10].


4-7 Deep neural network architecture and its forward and backward phase. Adapted from [8].

4-8 Example of simple feedforward neural network.

4-9 Graphical model of Deep Belief Network. Adapted from [12].

4-10 Architectural graph of Boltzmann machine and Restricted Boltzmann machine. Adapted from [12].

4-11 Illustration of auto-encoder. Adapted from [13].

4-12 Example of deep neural network [13].

4-13 Auto-encoder used for pre-training [13].

4-14 Dropout neural network model [14].

4-15 Simple example of dropout for feedforward neural network.

5-1 (a) A $4 \times 1$ OBFN system. (b) The neural network configuration of a $4 \times 1$ OBFN.

5-2 The neural network configuration of a $4 \times 1$ OBFN with combiner after every stage.

5-3 (a) An $8 \times 1$ OBFN system. (b) The neural network configuration of an $8 \times 1$ OBFN.

5-4 Illustration of the gradient projection method [15, p. 49f].

5-5 Simulation results of a $4 \times 1$ OBFN using stochastic gradient projection. (a) group delay response. (b) test error. Nominal parameters: Nfreq = 10, Nexamples = 20000, Nstochastic = 20, delay = $[0 \ 0.1 \ 0.2 \ 0.3]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, iteration $= 1000$.

5-6 Illustration of binary tree $4 \times 1$ OBFN.

6-1 The comparison of the test error result of the $4 \times 1$ OBFN simulation using different momentum coefficients ($m$): (a) $m = 0$, (b) $m = 0.4$, (c) $m = 0.8$, (d) $m = 1.0$. Nominal parameters: $\kappa_0 = [0.6, 0.6, 0.6, 0.6]$, $\phi_0 = [0, 2\pi - 0.4, 0.4, 0]$, Nfreq $= 20$, Nexamples $= 20000$, Nstochastic $= 20$, delay $= [0 \ 0.2 \ 0.4 \ 0.6]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, iteration $= 500$. 

Herminarto Nugroho

Master of Science Thesis
6-2 The comparison of the delay response of the $4 \times 1$ OBFN simulation using different momentum coefficients ($\phi$): (a) $\phi = 0.7$, (b) $\phi = 1.0$. Nominal parameters: $\phi_0 = [0.6, 0.6, 0.6, 0.6]$, $\phi = [0, 2\pi, 0.4, 0.4, 0]$, $Nfreq = 20$, $Nexamples = 20000$, $Nstochastic = 20$, delay $= [0.2, 0.4, 0.6]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, iteration $= 500$ .......................... 65

6-3 Unsupervised pre-train illustration for a $4 \times 1$ OBFN, (a) autoencoder for layer 1 (b) autoencoder for layer 2 (c) autoencoder for layer 3. .......................... 67

6-4 Unsupervised pre-train illustration for a $4 \times 1$ OBFN after deleting zero weights, (a) autoencoder for layer 1 (b) autoencoder for layer 2 (c) autoencoder for layer 3. .......................... 67

6-5 Group delay response of $4 \times 1$ OBFN using initialization from denoising auto-encoder with SNR $= 0.05$. Nominal parameters: $Nfreq = 10$, $Nexamples = 20000$, $Nstochastic = 20$, delay $= [0.1, 0.2, 0.3]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 1 \times 10^{-5}$, momentum coefficient $= 0.9$, iteration $= 1000$ .......................... 69

6-6 Dropping procedure of the neural network representation of an $M \times 1$ OBFN system (a) neurons in each layer are dropped with probability $p^{(i)} = 0.5$. Note that the gray lines represent zero weights, which are deleted in (b). (b) neural network representation after deleting zero weight. The only remaining outputs are the output of the surviving neurons after dropout. .......................... 70

6-7 The test error of dropout regularization implementation in a $4 \times 1$ OBFN with $p = 0.5$ for all layers except input layer whose $p = 0.9$. Nominal parameters: $Nfreq = 10$, $Nexamples = 20000$, $Nstochastic = 20$, delay $= [0.1, 0.2, 0.3]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.9$, iteration $= 1000$ .......................... 70

6-8 Illustration of the binary tree $2^3 \times 1$ OBFN. All $2^{n-1}$ ORRs in each stage form several mirror pairs of ORR pairs. The red dotted-line represents the mirror line. .......................... 72

6-9 Illustration of how to choose starting solution $\phi$ of ORR pairs. .......................... 73

6-10 Histogram plots of 100 group delay errors of initialization results for $4 \times 1$ OBFNs with delays $d = [0.1, 0.2, 0.3]$ ns, where (a) $d = 1$, (b) $d = 2$, (c) $d = 3$. .......................... 75

6-11 The histogram of (a) squared group delay error, (b) normalized squared group delay error for delay $= 2$ $[0.1, 0.2, 0.3]$ ns. Nominal parameters: $Nfreq = 10$, $Nexamples = 20000$, $Nstochastic = 20$, delay $= 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.9$, iteration $= 1000$. .......................... 76

6-12 The comparison of the delay responses of $8 \times 1$ OBFN simulations with initialization using: (a) squared group delay error approach, (b) normalized squared group delay error approach. Nominal parameters: $Nfreq = 10$, $Nexamples = 20000$, $Nstochastic = 20$, delay $= 2$ $[0.1, 0.2, 0.3]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 1 \times 10^{-5}$, momentum coefficient $= 0.7$, iteration $= 2000$. .......................... 77

6-13 Test error and group delay response from tuning a $4 \times 1$ OBFN with delay $3$ $[0.1, 0.2, 0.3]$ ns. Nominal parameters: $Nfreq = 3$, $Nexamples = 20000$, $Nstochastic = 20$, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.9$, iteration $= 1000$. .......................... 77

6-14 Test error and group delay response from tuning a $4 \times 1$ OBFN with delay $3$ $[0.1, 0.2, 0.3]$ ns. Nominal parameters: $Nfreq = 20$, $Nexamples = 20000$, $Nstochastic = 20$, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.9$, iteration $= 1000$. .......................... 78

6-15 Test error results of a $4 \times 1$ OBFN with (a) fixed 20 frequencies, (b) randomly pick 3 out of 20 frequencies. Nominal parameters: $Nfreq = 20000$, $Nstochastic = 20$, delay $= [0.1, 0.2, 0.3]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.7$, iteration $= 1000$. .......................... 80
7-1 The configuration of an $8 \times 1$ OBFN which consists of 3 stages and 12 ORRs. 82
7-2 Illustration of $\phi_c \pm 1$ can cover a whole 2 GHz bandwidth. 83
7-3 Histogram plots of 100 errors of initialization results for $8 \times 1$ OBFNs with delays $d[0 \ 0.1 \ 0.2 \ \cdots \ 0.7]$ns, where (a) $d = 1$, (b) $d = 2$, (c) $d = 3$. 84
7-4 Group delay responses of $8 \times 1$ OBFNs for delay $d[0, 0.1, 0.2, \cdots, 0.7]$ ns with $d = 1$, $d = 2$, and $d = 3$. (left) the best case, (right) the worst case. Nominal parameters: Nexamples = 20000, Nstochastic = 20, $f_c = 107.52$ THz, Nfreq = 20, random freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 2000. 85
7-5 Group delay responses of $8 \times 1$ OBFNs for delay $d[0, 0.1, 0.2, \cdots, 0.7]$ ns. (a) deep learning approach, (b) non-linear programming. Nominal parameters: Nexamples = 20000, Nstochastic = 20, $f_c = 107.52$ THz, Nfreq = 20, random freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 200. 87
7-6 The configuration of a $16 \times 1$ OBFN which consists of 4 stages and 32 ORRs. 88
7-7 Histogram plots of 100 errors of initialization results for $16 \times 1$ OBFNs with delays $d[0 \ 0.1 \ 0.2 \ \cdots \ 1.5]$ns, where (a) $d = 1$, (b) $d = 2$, (c) $d = 3$. 89
7-8 Group delay responses of a $16 \times 1$ OBFN for desired delay $d[0, 0.1, 0.2, \cdots, 1.5]$ ns with $d = 1$, $d = 2$, and $d = 3$. (left) best case, (right) worst case initialization. Nominal parameters: Nexamples = 20000, Nstochastic = 20, $f_c = 107.52$ THz, Nfreq = 20, random freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 5000. 90
7-9 Group delay responses of $16 \times 1$ OBFNs for delay $d[0, 0.1, 0.2, \cdots, 1.5]$ ns. (a) deep learning approach, (b) non-linear programming. Nominal parameters: Nexamples = 20000, Nstochastic = 20, $f_c = 107.52$ THz, Nfreq = 20, random freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 1000. 92
7-10 Histogram plots of 100 errors of initialization results for $32 \times 1$ OBFNs with delays $d[0 \ 0.1 \ 0.2 \ \cdots \ 3.1]$ns, where (a) $d = 1$, (b) $d = 2$, (c) $d = 3$. 94
7-11 Group delay responses of a $32 \times 1$ OBFN for desired delay $d[0, 0.1, 0.2, \cdots, 3.1]$ ns with $d = 1$, $d = 2$, and $d = 3$. (left) best case, (right) worst case initialization. Nominal parameters: Nexamples = 20000, Nstochastic = 20, $f_c = 107.52$ THz, Nfreq = 20, random freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 10000. 95
List of Tables

5-1 Tuning $4 \times 1$ OBFNs results in complex number $\kappa$ and $\phi$ ........................................ 54

5-2 Resulting $\kappa$ and $\phi$ of tuning a $4 \times 1$ OBFN using stochastic gradient projection. Nominal parameters: $\text{Nfreq} = 10$, $\text{Nexamples} = 20000$, $\text{Nstochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000. .......................................................... 58

5-3 Normalized squared group delay error for several delays, with $\kappa_0 = [0.9, 0.9, 0.9, 0.9]$ and $\phi_0 = [0, (2\pi - 0.4), 0.4, 0]$ using stochastic gradient projection. Nominal parameters: $\text{Nfreq} = 10$, $\text{Nexamples} = 20000$, $\text{Nstochastic} = 20$, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000. .......................................................... 59

5-4 Normalized squared group delay error for different $\kappa_0$ but the same $\phi_0 = [0, (2\pi - 0.4), 0.4, 0]$ using stochastic gradient projection. Nominal parameters: $\text{Nfreq} = 10$, $\text{Nexamples} = 20000$, $\text{Nstochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000. .......................................................... 60

5-5 Normalized squared group delay error for some $\phi_0$ but the same $\kappa_0 = [0.9, 0.9, 0.9, 0.9]$ using stochastic gradient projection. Nominal parameters: $\text{Nfreq} = 10$, $\text{Nexamples} = 20000$, $\text{Nstochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000. .......................................................... 60

6-1 Normalized squared group delay errors for different momentum coefficients $\mu$ based on simulation of $4 \times 1$ OBFNs. Nominal parameters: $\kappa_0 = [0.6, 0.6, 0.6, 0.6]$, $\phi_0 = [0, 2\pi - 0.4, 0.4, 0]$, $\text{Nfreq} = 20$, $\text{Nexamples} = 20000$, $\text{Nstochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 100. .......................................................... 66

6-2 Pre-training results of $4 \times 1$ OBFNs using auto-encoders with tight-weight constraint. Nominal parameters: $\text{Nfreq} = 10$, $\text{Nexamples} = 20000$, $\text{Nstochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $1 \times 10^{-5}$. .......................................................... 68

6-3 Pre-training results of $4 \times 1$ OBFNs using auto-encoders without tight-weight constraint. Nominal parameters: $\text{Nfreq} = 10$, $\text{Nexamples} = 20000$, $\text{Nstochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $1 \times 10^{-5}$. .......................................................... 68
6-4 Pre-training results of $4 \times 1$ OBFNs using denoising auto-encoders. Nominal parameters: Nfreq 10, Nexamples = 20000, Nstochastic = 20, delay = $[0, 0.1, 0.2, 0.3]$ ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $1 \times 10^{-5}$. 68

6-5 Initial guesses and run time comparison of two initialization methods: random initialization and group delay error approach for different desired delays $d[0, 0.1, 0.2, 0.3]$ ns, gradient projection iteration = 500. 75

6-6 Initialization results using group delay error approach and final normalized squared group delay error of a $4 \times 1$ OBFNs for delay = $d[0, 0.1, 0.2, 0.3]$, iteration = 500. 75

6-7 Error and running time comparison of fixed frequencies and randomly pick frequencies case of a $4 \times 1$ OBFN. Nominal parameters: Nexamples = 20000, Nstochastic = 20, desired delay = $d[0, 0.1, 0.2, 0.3]$ ns, frequency center ($f_0$) = 107.52 THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 1000. 80

7-1 Initial guesses for $8 \times 1$ OBFNs. The initialization based on group delay error approach will find the value of $\kappa_0, \phi_{a1}, \phi_{a2}, \phi_{b3}$, Nominal parameter $\phi_c = 0$ or equivalently $\phi_c = 2\pi$. 83

7-2 Initialization results, initial and final normalized squared group delay errors using group delay error approach of an $8 \times 1$ OBFN for delays $d[0, 0.1, 0.2, \cdots, 0.7]$. 1000 random candidate initial parameters are chosen. Final $E_{NSGD}$ is obtained after 2000 gradient projection iteration with learning rate = 2E-5 and momentum coefficient = 0.7. 84

7-3 Comparison of optimum parameters, final normalized squared group delay error, and total run time for 200 iterations of tuning $8 \times 1$ OBFN using deep learning approach and Non-linear programming (NLP). Both simulations used the same initial parameters. 87

7-4 Initial guesses for a $16 \times 1$ OBFN. Nominal parameter $\phi_c = 0$ or equivalently $\phi_c = 2\pi$. 88

7-5 Initialization results and normalized squared group delay errors of an $16 \times 1$ OBFN for delays $d[0, 0.1, \cdots, 1.5]$. 2000 random initial parameters are chosen. Final $E_{NSGD}$ is obtained after 5000 iteration with learning rate = 2E-5 and momentum coefficient = 0.7. 89

7-6 Normalized squared group delay error and total run time comparison of tuning $16 \times 1$ OBFNs using deep learning approach and non-linear programming with iteration = 1000. 91

7-7 Initial guesses for a $32 \times 1$ OBFN. Initialization using group delay error will find the value of $\kappa_0, \phi_{a1}, \phi_{a2}, \cdots, \phi_{a15}$. Nominal parameter $\phi_c = 0$ or $\phi_c = 2\pi$. 93

7-8 Initialization results and normalized squared group delay errors of a $32 \times 1$ OBFN for delays $d[0, 0.1, 0.2, \cdots, 1.5]$. 5000 random candidate initial parameters are chosen. Final $E_{NSGD}$ is obtained after 10000 gradient projection iteration with learning rate = 2E-5 and momentum coefficient = 0.7. 94

A-1 The resulting $\kappa^*_4$ and $\phi^*_4$ of $4 \times 1$ OBFNs for several delays, with $\kappa_0 = [0.9, 0.9, 0.9, 0.9]$ and $\phi_0 = [0, (2\pi - 0.4), 0.4, 0]$ using stochastic gradient projection. Nominal parameters: Nfreq = 10, Nexamples = 20000, Nstochastic = 20, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 500. 99

A-2 The resulting $\kappa^*_4$ and $\phi^*_4$ of $4 \times 1$ OBFNs for different $\kappa_0$ but the same $\phi_0 = [0, (2\pi - 0.4), 0.4, 0]$ using stochastic gradient projection. Nominal parameters: Nfreq = 10, Nexamples = 20000, Nstochastic = 20, delay = $[0, 0.1, 0.2, 0.3]$ ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 500. 100
A-3 The resulting $\kappa^*_i$ and $\phi^*_i$ of $4 \times 1$ OBFNs for different $\phi_0$ but the same $\kappa_0 = [0.9, 0.9, 0.9, 0.9]$ using stochastic gradient projection. Nominal parameters: $Nfreq = 10$, $Nexamples = 20000$, $Nstochastic = 20$, delay $= [0.0.1.0.2.0.3]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, iteration $= 500$.

A-4 Initial and optimum values of $\kappa$ and $\phi$ for some different desired delays obtained by tuning $8 \times 1$ OBFNs considering two initialization cases: the best and the worst case. Nominal parameters: $Nexamples = 20000$, $Nstochastic = 20$, desired delay $= d[0, 0.1, 0.2, \cdots, 0.7]$ ns, $f_c = 107.52$ THz, $Nfreq = 20$, random picked freq $= 3$, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.7$, iteration $= 2000$.

A-5 Initial and optimum values of $\kappa$ and $\phi$ for some different desired delays obtained by tuning $16 \times 1$ OBFNs considering two initialization cases: the best and the worst case. Nominal parameters: $Nexamples = 20000$, $Nstochastic = 20$, desired delay $= d[0, 0.1, 0.2, \cdots, 1.5]$ ns, $f_c = 107.52$ THz, $Nfreq = 20$, random picked freq $= 3$, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.7$, iteration $= 5000$.

A-6 Initial and optimum values of $\kappa$ and $\phi$ for some different desired delays obtained by tuning $16 \times 1$ OBFNs considering two initialization cases: the best and the worst case. Nominal parameters: $Nexamples = 20000$, $Nstochastic = 20$, desired delay $= d[0, 0.1, 0.2, \cdots, 1.5]$ ns, $f_c = 107.52$ THz, $Nfreq = 20$, random picked freq $= 3$, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.7$, iteration $= 5000$.

A-7 Initial and optimum values of $\kappa$ and $\phi$ for some different desired delays obtained by tuning $16 \times 1$ OBFNs considering two initialization cases: the best and the worst case. Nominal parameters: $Nexamples = 20000$, $Nstochastic = 20$, desired delay $= d[0, 0.1, 0.2, \cdots, 1.5]$ ns, $f_c = 107.52$ THz, $Nfreq = 20$, random picked freq $= 3$, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.7$, iteration $= 5000$.

A-8 Comparison of optimum parameters, final normalized squared group delay error, and total run time of tuning $16 \times 1$ OBFN using deep learning approach and non-linear programming. Both simulations use the same initial parameters and 1000 iterations.

A-9 Initial and optimum parameters for $32 \times 1$ OBFNs. $Nexamples = 20/20k$, $d = 1$, $Nfreq = 3/20$, learning rate $= 2 \times 10^{-5}$, momentum $= 0.7$, iteration $= 10000$.

A-10 Initial and optimum parameters for $32 \times 1$ OBFNs. $Nexamples = 20/20k$, $d = 2$, $Nfreq = 3/20$, learning rate $= 2 \times 10^{-5}$, momentum $= 0.7$, iteration $= 10000$.

A-11 Initial and optimum parameters for $32 \times 1$ OBFNs. $Nexamples = 20/20k$, $d = 3$, $Nfreq = 3/20$, learning rate $= 2 \times 10^{-5}$, momentum $= 0.7$, iteration $= 10000$. 

Master of Science Thesis

Herminarto Nugroho
Preface

The idea of doing my thesis on this subject came after my scholarship committee, The Ministry of Communication and Information of the Republic of Indonesia, required me to do research in the area of communication engineering. I remembered that Prof. dr. ir. Michel Verhaegen had once presented some of his current research projects, with one of them being control of airplane communications. I later contacted him to ask about this project, and he then introduced me to my thesis supervisor, Dr. Sander Wahls, whose expertise is in communications.

The most important technical acknowledgment is made to Dr. Sander Wahls for providing me with guidance and opportunities for learning telecommunication, a field which I am interested but not yet experienced in. I would also like to thank Laurens Bliik for his constant support and willingness to discuss the problems I have encountered in this project.

Moreover, I would like to thank all my friends with whom I traveled, shared incredible memories, laughter and joy. Finally, I would like to deeply thank my family who have supported me in all my endeavours and motivated me to aim higher.

An additional technical note is required: if any typo or incorrect information is found please contact me via herminarto.nugroho@gmail.com
The demand of being able to access internet and TV has been increasing recently, even on intercontinental flights. This has been the motivation to develop effective and efficient ways for planes to transmit/receive signals to/from satellites. In order to transmit/receive RF signals to/from a satellite, the plane should focus the transmission towards the satellite. The conventional solution for steering focused beams toward satellites is to steer dish antennas mechanically. However, mechanically steered antennas have some disadvantages, such as high maintenance cost, large dimension, and increased drag forces [3, p.10].

Phased array antenna (PAA) systems offer several advantages when compared to mechanically steered antennas, such as agile beam steering, relatively low maintenance cost, reduce drag forces when applied for instance in vehicles and aircrafts, and the possibility of supporting multiple antenna beams [16].

A PAA system consists of an array of multiple antenna elements (AEs), corresponding transmission and/or reception units, and a beamformer. The individual AE signals consist of a time-delayed version of some desired signal, some undesired signals (from different directions), and noise. The values of these delays are different for each AE, depending on the geometrical distribution of the AE locations and the direction of the incoming/outgoing desired signal. In reception mode, the beamformer consists of a delay-and-combine network that equalizes the delay values of the desired signal, such that the desired signal adds up in phase and is reinforced, whereas the undesired signals do not add up and are hence suppressed [1]. In transmission mode, it adds different delays to the transmitted signal such that the emitted signals from the AEs superpose as desired. It is desirable that the time-delays are tunable, in order to be able to alter the reception/transmission angle of the PAA.

This thesis will focus on tuning beamforming in reception mode. A seamlessly tunable optical beamformer concept proposed by Meijerink et al. [17] is used. It is based on coherent optical combining in an optical beamforming network (OBFN) with optical ring resonators (ORRs) [18, 19] as tunable delay elements. The ORRs are tuned thermally, which makes the tuning process highly accurate [20]. Non-linear programming (NLP) optimization is currently used to determine the parameters of the ORRs. One of the drawbacks of NLP is that it is a
black-box solver which cannot exploit the special structure of OBFNs. This drawback makes NLP not suitable for tuning large-scale OBFNs.

In order to be able to tune large scale OBFNs, a deep learning approach is proposed. The binary topology of the OBFN is in many aspects similar to the structure of neural networks. Moreover, deep neural networks with many hidden layers can have similar dimension as a large scale OBFN. The recent discoveries in neural networks made in the mid of last decade make deep learning a potential solution for tuning OBFNs.

1-1 Goals of the Thesis

The main goal of this thesis is to find data-driven methods which work efficiently for tuning large-scale OBFNs. These methods will address the basic problem of the current tuning method (Non-linear Programming Optimization), which cannot exploit the special structure of OBFNs.

1-2 Research Approach

Based on the main goal of the thesis, the research approach can be divided into two parts containing several topics that must be addressed. The first part concerns the creation of a new OBFNs tuning method using the deep learning approach. This approach exploits the special structure of OBFNs, such that it can be used to tune large-scale OBFNs. The second part is related to the way this approach can be effectively and efficiently performed by implementing major breakthroughs in the field of the deep learning.

Concerning the creation of a new tuning method using deep learning approach, it is apparent that the following topics need to be addressed:

1. A way to represent the special structure of OBFNs into neural network will be developed.

2. A way to generate training examples has to be addressed. They consist of several sets of input vector and their respective desired output vector.

3. Weight matrices will be formulated based on the neural network representation mentioned in point 1.

4. A non-linear optimization problem needs to be formulated. It consists of the cost function, design variables, constraints, and the suitable optimization algorithm.

5. The backpropagation algorithm will be derived.

6. The derivative of the frequency response of the ORRs to parameters/design variables needs to be derived.

Regarding the implementation of breakthroughs in the field of the deep learning, there are several issues which have to be addressed:
1. A good initialization method needs to be developed.
2. A method to avoid frequency overfitting has to be addressed.
3. A way to improve the speed of computation should be developed. This is essential for tuning large-scale OBFNs.
4. The implementation of a new tuning approach in large-scale OBFNs will be analyzed.

1-3 Outline

The theoretical background of this thesis consists of three chapters. In Chapter 2, a brief introduction to phased array antennas (PAAs) and optical ring resonators (ORRs) is presented, including mathematical derivations of frequency responses, group delays and phase delays. In Chapter 3, the principles of ring resonator-based optical beamforming networks (OBFNs) are discussed. The derivation of common optimality criteria is also presented in this chapter. In the last part of this chapter, the current method for tuning OBFNs and its disadvantages are discussed. In Chapter 4, principles of deep learning are presented. Recent developments in deep learning which seem to be potential solution to tune large-scale OBFNs will be explained briefly in that chapter. After theoretical background chapters, the development and results from the new method are presented in the following three chapters. In Chapter 5, a way to exploit the special structure of OBFNs into neural network structure representation will be presented, followed by how to generate training examples, weight matrices, and the backpropagation algorithm. The implementation of some breakthroughs in the field of deep learning is presented in Chapter 6, which mainly focus on how to speed up computation, obtain good initialization and avoid overfitting. In Chapter 7, the simulation of tuning larger-setup of OBFNs is presented. In that chapter, the implementation of all concept of the new method is being used to tune a larger OBFNs. Finally, some conclusion and recommendations are presented in Chapter 8.

1-4 Nomenclature

Row or column vectors are represented by boldface lower-case symbols such as $\mathbf{x}$. Boldface upper-case symbols, such as $\mathbf{A}$, are used for matrices. Regular font, such as $x_1$, denotes a scalar variable.

The symbol $(\circ)^*$ denotes the complex conjugate number of $(\circ)$. It can also be used to represent the optimal result from an optimization problem. The symbol $(\circ)^\top$ represents the transpose of $(\circ)$.

The operator $\mathcal{F}$ denotes the Fourier transform. The operator $\nabla$ will be used to represent the gradient. Given a certain scalar function $f : \mathbb{R}^N \to \mathbb{R}$, its gradient can be expressed as $\nabla f(\mathbf{x}) = \left[ \frac{\partial f}{\partial x_1} \ldots \frac{\partial f}{\partial x_N} \right]$.

The norm of vectors used during this thesis will be, by default, the 2-norm, which is denoted by $\|\circ\|_2$. For a vector $\mathbf{x} \in \mathbb{R}^N$, the 2-norm is defined as $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_N^2}$.
Chapter 2

Phased Array Antennas and Optical Ring Resonators: An Overview

In the last century, radio systems have become an essential way to communicate by means of electromagnetic waves with frequencies below that of visible light. In recent years, an increasing demand for fast information exchange has motivated the development of many new broadband radio applications. People wish to be able to connect to the internet, participate in teleconferences, or watch live TV no matter where they are, especially when traveling on a long intercontinental flight. This demand motivated the development of aeronautic communication systems between satellites and airplanes, as illustrated in Figure 2-1.

![Illustration of the aeronautic communication system](image)

Figure 2-1: Illustration of the aeronautic communication system [3].

In these applications, the radio links have low signal power densities. Therefore, high-gain and direction-sensitive antennas are essential for signal transmission and reception. Ordinary omni-directional antennas are not preferable because, although they are directional-sensitive, they have a low gain [21]. A conventional solution for a direction-sensitive antenna is a dish antenna, since its main beam can be steered mechanically to desired directions [22].

Even though being direction-sensitive and having high-gain, a dish antenna is slow because its speed is limited by mechanical movements. Mechanical movements also have negative effect
on the tuning precision. Other drawbacks are its large weight, large size, and the aerodynamic
drag effect when mounted on top of a plane [3, p.10]. They are also difficult to maintain [1].
Phased array antennas (PAAs) are an attractive alternative to dish antennas [23, 17]. The
explanation of PAA will be covered in the first section of this chapter.

2-1 Phased Array Antennas

A phased array antenna (PAA) (illustrated in Figure 2-3) consists of an array of antenna
elements (AEs). The PAA beam pattern is determined by the geometry of the array as well
as the signal amplitude and phase relation between the AEs [3, p. 5f.]. The main beam of
the PAA can be steered towards the desired direction without any mechanical movement.
This is possible by changing the amplitude and the phase relation of the AE signals [3,
p. 5f.]. To control both the amplitudes and phases of the AE signals, a so-called beamformer
is required. Conventionally a beamformer is realized in the electrical domain. However,
it can also be realized in the optical domain, i.e. photonically. Compared to its electrical
counterpart, the photonic beamformer has advantages such as compactness, light weight,
low loss, frequency independence, large instantaneous bandwidth, and inherent immunity
to electromagnetic interference [3, p. 4]. These advantages make photonic beamformer an
interesting solution for the communication between satellite and airplanes.

The simplest PAA is a one-dimensional linear array of antenna elements, which is based on
the geometry of multiple identical AEs equally spaced along a single line. Although there are
various array geometries as explained in [21, 23], to understand the basic principle of a PAA,
the one-dimensional linear array antenna is sufficient. Figure 2-4 shows the illustration of a simple 4-element linear array antenna.

\[ \Delta t_a(d, \theta) = \frac{d \sin(\theta)}{c_0}, \]  

where \( d \) is the spacing between the AEs in meters and \( c_0 \) is the speed of light in vacuum (in m/s).

The application of the PAA in the general case will not be as simple as for one-dimensional linear array antennas. More complex two-dimensional beam steering is sometimes required for PAA applications. A practical solution of this two-dimensional beam steering problem is to use a planar array antenna, as shown in Figure 2-5. Basically, it is a rectangular \( M \times N \) planar array which can be regarded as \( M \) columns of \( N \)-element linear arrays or \( N \) rows of \( M \)-element linear arrays. More formal analyses of PAAs can be found in [21, 23].
2-2 Optical Ring Resonators

A PAA can be controlled either electrically or optically [3, p. 3ff.]. The optical controller is called photonic beamforming system. Photonic beamforming systems use the same beamforming principles as their electrical counterparts, except AE signals are processed in the optical domain. In this case, AE signals of a PAA must first be converted into optical signals by means of optical modulation. Then, they are processed and combined by means of an OBFN. Eventually the output of the OBFN is converted back into the electrical domain by means of optical detection [3, p. 9f.]. The illustration of optical beamforming scheme is shown in Figure 2-6. Compared to its electrical counterpart, the photonic beamformer has advantages such as compactness, light weight, low loss, frequency independence, large instantaneous bandwidth, and inherent immunity to electromagnetic interference [3, p. 149].

![Figure 2-6: Beamformer illustration with N inputs using intensity modulation (IM) and direct optical detection. LNA = low-noise amplifier [1].](image)

The system which will be considered in this thesis is a photonic beamforming system in which RF signals received by AEs are modulated on optical carriers. Tunable optical delay lines synchronize the signals modulated on the optical carrier. Optical ring resonators (ORRs) are used to implement the tunable optical delay lines [19]. In this section, principles of ORRs will be covered including the structure, the transfer function and transfer matrix used to derive frequency response of ORRs, and delay properties of both single and multiple cascaded ORRs.

2-2-1 Structure of the Optical Ring Resonator

A simple one-input one-output single-stage ORR is illustrated in Figure 2-7. It consists of a ring-shaped waveguide and a straight waveguide. Those waveguides are able to couple light between each other. The parameter $\kappa$ is the power coupling coefficient, which has a value between 0 and 1, and $L_R$ is the round-trip length of the ring-shaped waveguide. Other parameters are $T$, which is the round-trip period, and $\phi$, which is the extra phase-shift due to the heater on the top of the ring.
2-2 Optical Ring Resonators

![Figure 2-7: Structure of a $1 \times 1$ single-stage ORR [3]](image)

**2-2-2 Mathematical Model of the Optical Ring Resonator**

The behavior of an optical component can be described by its frequency response, which relates the amplitude and phase of the field at the input to those at the output [3, p. 13]. Rabus [24] and Zhuang [3] have derived the frequency response of an ORR, which will be summarized in this subsection.

**Derivation of the ORR Frequency Response**

![Figure 2-8: An ORR formed by a $2 \times 2$ coupler and a feedback waveguide and its $Z$-transform schematic [3]](image)

To derive the frequency response of an ORR, we can first look at the behavior of its two basic building blocks, namely a waveguide feedback path and a $2 \times 2$ coupler [3, p. 14ff.], which are shown in Figure 2-8a. The $Z$-transform schematic of an ORR is given in Figure 2-8b. Let the signal at the right side of the ring be $E^r$ and the one on the left side be $E^l$, then one can derive the following relations:

$$E^r = -j\sqrt{\kappa}E^i + \sqrt{1-\kappa}E^l$$

$$E^l = rz^{-1}e^{-j\phi}E^r$$

$$E^l = rz^{-1}e^{-j\phi}(-j\sqrt{\kappa}E^i + \sqrt{1-\kappa}E^l)$$

$$E^l = \frac{-j\sqrt{\kappa}rz^{-1}e^{-j\phi}E^i}{1 - r\sqrt{1-\kappa}z^{-1}e^{-j\phi}}$$

\[ (2-2) \]
\[
E^o = \sqrt{1-\kappa}E^i - j\sqrt{\kappa}E^d
\]
\[
= \sqrt{1-\kappa}(1-r\sqrt{1-\kappa}e^{-j\phi})E^i - \kappa rz^{-1}e^{-j\phi}E^i
\]
\[
= \frac{\sqrt{1-\kappa} - rz^{-1}e^{-j\phi}}{1 - r\sqrt{1-\kappa}z^{-1}e^{-j\phi}}E^i
\] (2-3)

where \( r \) defines the power loss.

Substituting \( z^{-1} = e^{-2\pi j f T} \) with \( T \) being the round-trip time of the ring, results in the following frequency response:

\[
H(f) = \frac{\sqrt{1-\kappa} - re^{-2\pi j f T - j\phi}}{1 - r\sqrt{1-\kappa}e^{-2\pi j f T - j\phi}}
\] (2-4)

Equation (2-4) is the same as the Equation (2.18) in [3, p. 19] and Equation (2.52) in [24, p. 14].

**Derivation of the ORR Gain and Phase Response**

From the frequency response of an ORR in Equation (2-4), the gain \(|H|\) of this frequency response can be obtained as follows:

\[
H(f) = \frac{\sqrt{1-\kappa} - re^{-2\pi j f T - j\phi}}{1 - r\sqrt{1-\kappa}e^{-2\pi j f T - j\phi}}
\]
\[
= \frac{\sqrt{1-\kappa} - r (\cos(-2\pi f T - \phi) + j \sin(-2\pi f T - \phi))}{1 - r\sqrt{1-\kappa} \left( \cos(-2\pi f T - \phi) + j \sin(-2\pi f T - \phi) \right)}
\]
\[
= \frac{\sqrt{1-\kappa} - r \cos(-2\pi f T - \phi) - j r \sin(-2\pi f T - \phi)}{1 - \sqrt{1-\kappa r} \cos(-2\pi f T - \phi) - j \sqrt{1-\kappa r} \sin(-2\pi f T - \phi)}
\] (2-5)

\[
|H(f)|^2 = \frac{\left(\sqrt{1-\kappa} - r \cos(-2\pi f T - \phi)\right)^2 + \left(r \sin(-2\pi f T - \phi)\right)^2}{\left(1 - \sqrt{1-\kappa r} \cos(-2\pi f T - \phi)\right)^2 + \left(\sqrt{1-\kappa r} \sin(-2\pi f T - \phi)\right)^2}
\]
\[
= \frac{(1-\kappa) + r^2 - 2\sqrt{1-\kappa r} \cos(-2\pi f T - \phi)}{1 + (1-\kappa) r^2 - 2\sqrt{1-\kappa r} \cos(-2\pi f T - \phi)}
\] (2-6)

Therefore, the gain is

\[
|H(f)| = \sqrt{\frac{(1-\kappa) + r^2 - 2\sqrt{1-\kappa r} \cos(2\pi f T + \phi)}{1 + (1-\kappa) r^2 - 2\sqrt{1-\kappa r} \cos(2\pi f T + \phi)}}
\] (2-7)
The phase response $\varphi(f)$ is defined as the phase (or angle) of the frequency response $H(f)$ mentioned in Equation (2-4). It is given by

$$
\varphi(f) = \arctan \left( \frac{\text{Im}\{1 - r\sqrt{1 - \kappa}e^{-2\pi j f T - j \phi}\}}{\text{Re}\{1 - r\sqrt{1 - \kappa}e^{-2\pi j f T - j \phi}\}} \right) - \arctan \left( \frac{r \sin(2\pi f T + \phi)}{\sqrt{1 - \kappa - r \cos(2\pi f T + \phi)}} \right),
$$

(2-8)

### Derivation of the ORR Group Delay Response

To show the delay characteristics of ORRs, the concept of group delay is used, which is defined as the negative derivative of the phase of the transfer function with respect to the frequency [25]. For example, given simple system as follows:

$$
y(t) = \int_{-\infty}^{\infty} x(u)h(t-u)du \iff Y(s) = H(s)X(s)
$$

If $x(t) = e^{i\omega t}$, then $y(t) = |H(i\omega)|e^{i(\omega t + \varphi(\omega))}$, where $\varphi(\omega) = \arg\{H(i\omega)\}$. The group delay is then defined by

$$
\tau_g(\omega) = -\frac{d\varphi(\omega)}{d\omega}.
$$

The relation between phase response $\varphi$ and group delay $\tau$ is given by

$$
\tau(f) = -\frac{1}{2\pi} \frac{\partial \varphi}{\partial f}.
$$

(2-9)

The derivative of $\arctan \left( \frac{A(f)}{B(f)} \right)$ is given by

$$
\left[ \arctan \left( \frac{A(f)}{B(f)} \right) \right]' = \frac{B(f)A'(f) - A(f)B'(f)}{A(f)^2 + B(f)^2}.
$$

(2-10)

Letting $c = \sqrt{1 - \kappa}$ and $z = 2\pi f T + \phi$ gives the following expression for group delay:

$$
\tau(f) = -\frac{1}{2\pi} \frac{\partial \varphi}{\partial f} = -\frac{1}{2\pi} \frac{\partial \varphi}{\partial z} = -T \frac{\partial \varphi}{\partial z} \\
= -T \frac{\partial}{\partial z} \left[ \arctan \left( \frac{r \sin(z)}{c - r \cos(z)} \right) - \arctan \left( \frac{rc \sin(z)}{1 - rc \cos(z)} \right) \right] \\
= -T \left( \frac{(c - r \cos z)(r \cos z) - (r \sin z)(r \sin z)}{r^2 \sin^2 z + c^2 + r^2 \cos^2 z - 2rc \cos z} - \frac{(1 - rc \cos z)(rc \cos z) - (rc \sin z)(rc \sin z)}{r^2 c^2 \sin^2 z + 1 + r^2 c^2 \cos^2 z - 2rc \cos z} \right) \\
= -T \left( \frac{-r^2 + rc \cos z}{r^2 + c^2 - 2rc \cos z} - \frac{-r^2 c^2 + rc \cos z}{r^2 c^2 + 1 - 2rc \cos z} \right) \\
= T \left( \frac{r^2 - rc \cos z}{r^2 + c^2 - 2rc \cos z} + \frac{rc \cos z - r^2 c^2}{r^2 c^2 + 1 - 2rc \cos z} \right).
$$

(2-11)
Bandwidth-Delay Relation of Single ORR

Figure 2-9 shows that when the group delay $\tau$ increases, the width of the group delay curve decreases. This is due to the fact that the area under the group delay curve actually represents the phase shift of the ORR, which is a constant $2\pi$ for one free spectral range (FSR) [3, p. 27f.]. FSR is the spacing in optical frequency or wavelength between two successive reflected or transmitted optical intensity maxima or minima. It is the inverse of the round-trip time (round-trip group delay). Note that this observation reveals the tradeoff between the delay value and the bandwidth of an ORR.

![Figure 2-9: Group delay responses of a single ORR for different values of $\kappa$ [3]](image)

Figure 2-10 shows the definition of the bandwidth ($\Delta\Omega_{BW}$) of an ORR. The delay band is defined as the frequency band between the two vertical dash-lines. The $\bar{\tau}$ is the average delay over the delay band, and $\Delta\tau$ is the maximum delay ripple in the delay band. Note that all quantities shown in Figure 2-10 are normalized with respect to the FSR of the ORR [3, p. 27].

![Figure 2-10: Definition of delay bandwidth of a single ORR [3]](image)

When ORRs are used to generate group delays, the trade-off between the delay value and bandwidth will always be a consideration. In some cases the desired delay value can only be
reached by sacrificing bandwidth. This problem sometimes occurs when bandwidth and the delay are both large. A single ORR will not be able to cover large desired bandwidth and high desired delay value at the same time. In this particular problem, a cascade of multiple ORRs can be used.

2-2-3 Cascade of Multiple ORRs

As shown in the previous subsection, the trade-off between delay value and bandwidth needs to be tackled. The group delay response of an ORR becomes narrower as the peak delay value increases. When the required group delay value for the input signal is such that a single ORR cannot provide sufficient bandwidth, a cascade of multiple ORRs can be used to increase the delay bandwidth. The illustration of a cascade of multiple ORRs is shown in Figure 2-11.

![Figure 2-11: Cascade of multiple ORRs [3].](image)

The transfer function of N-stage cascade ORRs in normalized angular frequency is defined by the product of those of the individual single-stage ORRs [3, p. 31.]:

$$H_{\text{total}}(f) = \prod_{i=1}^{N} H_i(f), \quad (2-12)$$

where $H_i(f)$ is the transfer function of the $i$-th stage. The magnitude, phase and group delay are respectively defined by

$$|H_{\text{total}}(f)| = \prod_{i=1}^{N} |H_i(f)|, \quad (2-13)$$

$$\varphi_{\text{total}}(f) = \sum_{i=1}^{N} \varphi_i(f), \quad (2-14)$$

$$\tau_{\text{total}}(f) = \sum_{i=1}^{N} \tau_i(f). \quad (2-15)$$

**Group Delay of Cascade of Multiple ORRs**

By properly tuning the value of $\kappa_i$ and $\phi_i$ of each individual ORR in cascade, a flattened delay band can be generated for a desired delay value, which at the same time has an increased bandwidth as well (see Figure 2-12) [3, p. 31f.].

Figure 2-12 shows a cascade of three ORRs that generates a flat desired delay value. We can observe that the resulting delay band is not completely flat, but has a ripple. The ripple will result in unequal delays to different frequency components of the signals which fall in the
delay band. When ORRs are used in an OBFN as delay elements, the ripples will lead to variations in the beam angle of the antenna [19, p. 31.] [23].

In the case of a single ORR, one way to reduce the ripple is by reducing the bandwidth, as shown in Figure 2-10. In the case of a cascade of many ORRs, it can be reduced by shifting the resonance frequencies of the individual ORR closer to each other (squeezing) [3, p. 32f.]. This will reduce the bandwidth accordingly (see Figure 2-13). This phenomenon explains the trade-off between the ripple and bandwidth.

![Figure 2-12: Group delay response of a cascade of three ORRs [3].](image)

![Figure 2-13: Illustration of squeezing method to reduce delay ripple of a cascade of 2 ORRs [3].](image)

**Bandwidth-Delay Relation of Cascade of Multiple ORRs**

Zhuang [3] mentioned that for multiple cascaded ORRs, there are mutual tradeoffs between group delay, delay ripple, bandwidth and the number of ORR stages $N$. His simulation results of these tradeoffs are illustrated in Figure 2-14 for cascades of up to four ORRs.

Figure 2-14a shows that the bandwidth increases from 2 to 3 ORRs more than from 1 to 2 ORRs. Compared to a cascade of 2 ORRs, a cascade of 3 ORRs has the individual group
(a) Relation between bandwidth and ripple for different number of ORR (with average delay value is 5)

(b) Relation between bandwidth and average group delay for different number of stages and ripples

Figure 2-14: The simulation results of the tradeoffs [3].

delay response of the third ORR in the middle of the delay band [3, p. 33f.]. The third ORR has relatively lower peak delay value compared to the other two at the side in order to achieve a flattened delay band (see Figure 2-12). For a cascade of more ORRs, the delay bandwidth increases proportional to the number of ORRs [3, p. 33f.].

One may conclude that to get a wider bandwidth, it is possible to increase the number of ORRs used. However, this will obviously increase the system complexity [3, p. 37f.]. Other aspects such as scalability and system cost should be considered as well. Another structure which minimizes the required number of ORRs, has a low cost and is scalable while able to cover a wide range of desired bandwidths would be preferable. In the next chapter, the most suitable structure of OBFNs in terms of system complexity, cost and scalability will be presented.
Principles of Ring Resonator-Based Optical Beamforming Networks

Optical ring resonators (ORRs), which have been explained in the previous chapter, can be used to form optical beamforming networks. In ring resonator-based optical beamforming networks, the ORR is used as the delay element. Besides the delay element, OBFNs consist of other tunable optical signal processing circuitry as well. Through an OBFN, the signals received by the individual antenna elements (AEs) can be synchronized and combined to achieve the desired signal. In the first section of this chapter, a combining structure which will be used for OBFNs in this thesis will be explained. This structure is designed to be low-cost, scalable, composed by minimum number of ORRs, and able to cover wide range of desired bandwidths, which is better than a conventional cascade of multiple ORRs.

3-1 Optical Beamforming Network Structure

As has been mentioned in the last part of the previous chapter, the system complexity is directly related to the number of ORRs. Beside that, other aspects such as scalability and system cost should also be taken into account. In this section, an explanation about which structure of OBFN is the most suitable in terms of system complexity, cost, and scalability will be given.

Zhuang [3] mentioned several structures of the OBFN, which can be grouped into two main structures: parallel-structured OBFNs and binary-tree-structured OBFNs.

3-1-1 Parallel-structured Optical Beamforming Networks

Parallel-structured OBFNs are the most straight forward structure of OBFNs. They use an architecture with only two stages; a parallel-structured delay stage followed by a combining...
stage [3, p. 38f.]. In this parallel structure, each OBFN channel uses an independent ORR-based delay element (DE) [3, p. 38ff.]. Those DEs can be arranged in both symmetrical (see Figure 3-1) and asymmetrical manner (see Figure 3-2).

The difference between symmetrical and asymmetrical parallel-structured OBFNs is the addition of fixed delay in asymmetrical parallel-structured OBFNs [3, p. 38ff.]. Fixed delays can be added to the signals before they enter the ORRs as shown in Figure 3-2. The added delays should be such that they compensate the delayed-time offsets for the maximum negative signal receiving angle. The additional fixed delays can be achieved by introducing additional signal path length differences either in the RF paths or in the optical paths [3, p. 40f.].

3-1-2 Binary-tree-structured Optical Beamforming Networks

Binary-tree-structured OBFNs are OBFNs with multiple binary-combining stages, where signals of each two neighboring channels are synchronized and combined separately [3, p. 42ff.]. Unlike the parallel structure, the binary-tree structure divides the signal combining process into multiple steps. In the first stage, the entire AE array is divided into multiple AE pairs formed by two neighboring AEs, and signals of each AE pair are synchronized and combined into new imaginary AE array [3, p. 42f.]. In the second step, this new imaginary AE array is again divided into multiple AE pairs, and each AE pair is synchronized and combined separately. This process will continue until all AE signals are combined [3, p. 42f.]. Like parallel-structured OBFNs, binary-tree-structured OBFNs can be arranged in both symmetrical (see Figure 3-3) and asymmetrical manner with addition of fixed delay (see Figure 3-4).
Zhuang [3, p. 45f.] found in numerical experiments that, for the same antenna specifications and signal bandwidth, asymmetrical binary-tree-structured OBFNs lead to the least number of necessary ORRs. The asymmetric structure also benefits the scalability of the system. Each time when a stage is added to the OBFN, the number of input ports will be doubled [3, p. 46]. An 8 × 1 OBFN with three stages, which is an extension of the 4 × 1 OBFN in Figure 3-4, is shown in Figure 3-5.

Figure 3-3: Schematic of an 4 × 1 OBFN with symmetric binary tree structure [3]

Figure 3-4: Schematic of an 4 × 1 OBFN with asymmetric binary tree structure [3]

Figure 3-5: Schematic of an 8 × 1 OBFN with asymmetric binary-tree structure [3]
3-2 Optical Phase Synchronization

The required complexity of OBFNs can be reduced by lowering the bandwidth of the modulated optical signals at OBFN’s inputs. Performing optical single-sideband suppressed-carrier (SSB-SC) modulation minimizes the required optical bandwidth [1, p. 7]. This section explains advantages and challenges of SSB-SC modulation.

![Figure 3-6](image)

**Figure 3-6:** Spectrum of the modulated signal (solid line) in case of: (a) optical intensity modulation (IM), (b) optical single-sideband suppressed-carrier (SSB-SC) modulation. The suppressed sideband and optical carrier are represented by the dotted line. Corresponding desired group delay of a delay elements in the OBFN is represented by a dashed line. $f_0$ specifies the optical carrier frequency, $f_{min}$ and $f_{max}$ specify the lower and upper bound of the frequency range of the modulating signal respectively [1].

The most straightforward modulation choice would be the intensity modulation (IM) shown in Figure 3-6a. In this modulation, OBFNs should be designed to provide a flat group delay response covering the whole optical carrier frequency and two sidebands [1, p. 6f.] (see the dashed line in Figure 3-6a). As a result, the minimum optical bandwidth required to perform optical beamforming is equal to twice the maximum frequency of the modulated signal [1, p. 7]. This large optical bandwidth requires a large FSR of the ORRs, which corresponds to a low round-trip time and hence a small physical size, which may introduce problems with the realizability of the corresponding optical chips [1, p. 7].

Observing Figure 3-6a, it should be noted that the two sidebands of the intensity-modulated signal in fact carry the same information, while the optical carrier is only required for O/E conversion [1, p. 7]. Therefore, the required optical bandwidth of the OBFN can be minimized by performing optical single-sideband suppressed-carrier (SSB-SC) modulation (depicted in Figure 3-6b) instead of optical IM [1, p. 7].

Note that removing the optical carrier is essential here. This is due to the fact that the phase relation between the optical carrier and the remaining sideband is lost when the group delay response is only optimized for the sideband frequency range [1, p. 7]. The problem is when the delayed optical signals are combined such that they add up in phase, the optical carriers from the different paths might not add up in phase [1, p. 7]. Correcting this would significantly increase the required bandwidth, and hence the number of ORR as well.

Optical phase synchronization is used to correct this issue by adding extra phase shifters. Before each coupler in the chip, an extra phase shifter is inserted in the upper branch which
is implemented in the configuration in Figure 3-7. It can also be added in front of every path, as shown in Figure 3-8. The function of these phase shifters is to adjust the optical phases in the beamformer paths in order to maximize the output signal [26, p. 17].

![Figure 3-7: 8 × 1 OBFN configuration with extra phase shifters inserted in the upper branch [6].](image1)

![Figure 3-8: 8 × 1 OBFN configuration with extra phase shifters inserted in front of every path. Adapted from [6].](image2)

One approach is to use a feedback loop from the output signal to the phase shifters in the OBFN paths. This could be based on the total power of the desired signal, which should be maximized [1, p. 9]. To obtain maximum total power of the desired signal, the expression

\[
\sum_{n \neq m} \frac{1}{2} \left( \bar{\phi}_n - \bar{\phi}_m + \phi_n(f) - \phi_m(f) - 2\pi f (\Delta t_n - \Delta t_m) \right)^2
\]

should be minimized for a specific frequency \(f\) by changing the value of \(\bar{\phi}_n\) and \(\bar{\phi}_m\). The terms \(\bar{\phi}_n\) and \(\bar{\phi}_m\) specify the phase shift for path \(n\) and \(m\) respectively provided by the extra
phase shifters, which cannot depend on frequency. The term \( \varphi_n(f) \) and \( \varphi_m(f) \) specify the desired phase shift obtained by tuning the delay elements (ORRs). Note that the desired phase shift depends on the frequency.

It is assumed that extra phase shifters have been correctly tuned for the frequency center \( (f_c) \) which is located in the center of the bandwidth of interest. Equation (3-1) should be minimized for that specific \( f_c \). That is the extra phase shifters should be chosen such that

\[
\hat{\varphi}_n - \hat{\varphi}_m + \varphi_n(f_c) - \varphi_m(f_c) - 2\pi f_c(\Delta t_n - \Delta t_m) = 0, \forall n, m. \tag{3-2}
\]

If we observe Figure 3-7 carefully, we can see that in that figure, we use the last path as a reference, since there is not any extra phase shifter there. In this project we will use the top-most path as a reference, because of its simplicity. Choosing this reference will simplify the implementation of Equation (3-2). Adding extra phase shifters in front of every path also simplifies the computation, since we only need to compute one extra phase shifter in each path. Therefore, in this project we will use the implementation of extra phase shifter as shown in Figure 3-8.

**Example:**

Consider an \( 8 \times 1 \) OBFN configuration depicted in Figure 3-8. The extra phase shifter \( \hat{\varphi}_n \) is inserted in front of the \( n \)-th path. Using Equation (3-2) and setting the first path as a reference (i.e., \( m = 1 \)) we get

\[
\hat{\varphi}_n + \varphi_n(f_c) - 2\pi f_c \Delta t_n = 0. \tag{3-3}
\]

It means that the extra phase shifter for the \( n \)-th path is defined by

\[
\hat{\varphi}_n = -\varphi_n(f_c) + 2\pi f_c \Delta t_n. \tag{3-4}
\]

The phase shift \( \varphi_n \) is obtained by tuning the ORRs in path \( n \), which is defined by

\[
\varphi_n = \sum_k \phi_k, \tag{3-5}
\]

where \( k \) specifies the index of the ORR in path \( n \). For example from Figure 3-8, we obtain

\[
\begin{align*}
\varphi_2 &= \phi_1, \\
\varphi_3 &= \phi_2 + \phi_3, \\
\varphi_4 &= \phi_2 + \phi_3 + \phi_4, \\
\varphi_5 &= \phi_5 + \phi_9 + \phi_{10} + \phi_{11} + \phi_{12}, \\
\varphi_6 &= \phi_5 + \phi_9 + \phi_{10} + \phi_{11} + \phi_{12}, \\
\varphi_7 &= \phi_6 + \phi_7 + \phi_9 + \phi_{10} + \phi_{11} + \phi_{12}, \\
\varphi_8 &= \phi_6 + \phi_7 + \phi_8 + \phi_9 + \phi_{10} + \phi_{11} + \phi_{12}.
\end{align*}
\tag{3-6}
\]

Based on the simulation using numerical parameters mentioned in [2], when the value of \( \kappa \) and \( \phi \) approach the optimum value \( \kappa^* \) and \( \phi^* \) the phase shift obtained by tuning ORR is defined by

\[
\phi_k \approx \pi, \quad \forall k. \tag{3-7}
\]

For the sake of simplicity and fast computation, we will use this approximation instead of continuously giving output feedback to determine extra phase shifters. Note that different numerical parameters may result in different approximation. It is better to simulate using small setups to get the approximation of \( \phi_k \), then use it for larger setups.
3-3 Derivation of the Optimal Criteria

Tuning the OBFN means looking for the optimum parameters of each ORR such that every path of the AE will have its desired delay. Before looking for the optimum parameters through optimization algorithms, one first needs to know how optimality is defined. This section will explain and derive the formula for several optimization criteria (or cost functions), which can be used in the tuning process. There are three criteria which have been used as a cost function in the literature: delay, phase and power criterion [2, p. 21ff.], [3, p. 155f.].

3-3-1 Group Delay Criterion

The first criterion is based on the delay spectrum formula for one ORR

\[ \tau(f) = \frac{\kappa T}{2 - \kappa - 2\sqrt{1 - \kappa \cos(2\pi f T + \phi)}}, \quad (3-8) \]

where \( \tau(f) \) is the group delay for a specific frequency, \( \kappa \) is the coupling coefficient, \( \phi \) is an extra phase shift, and \( T \) is the round trip time. This equation (3-8) is equal to equation (2-11) with \( r = 1 \).

The index \( i \) then is added to specify the group delay for different ORRs. For every path of the OBFN, the delay responses should be summed to get the total delay response of one path. The total time delay (\( \tau_{\text{total}} \)) for one path with \( N \) ORRs is defined by

\[ \tau_{\text{total}}(f) = \frac{\kappa_i T}{2 - \kappa_i - 2\sqrt{1 - \kappa_i \cos(2\pi f T + \phi_i)}}, \quad (3-9) \]

This delay spectrum can now be compared to the desired target group delay \( D \). It is important to note that the value of the delay is only considered in the part of the spectrum where the modulated optical signal is located [2, p. 22]. It means that the delay will only be important over some specific range of frequency. Therefore the comparison between the group delay and target delay should be carried out in a certain band, defined by a starting frequency \( f_{\text{min}} \) and an end frequency \( f_{\text{max}} \).

In this thesis, it is assumed that the frequency range from \( f_{\text{min}} \) to \( f_{\text{max}} \) is given. After comparing the delay for a specific frequency range, we get the cost function for a particular path. The total cost function with respect to the delay criterion is obtained by summing the costs of all paths. Then, the cost function we want to minimize is

\[ \mu(\phi_1, \kappa_1, \phi_2, \kappa_2, \ldots) = \sum_{h=1}^{M} \left( \sum_{k=1}^{P} (\tau_{\text{total}}(f_k) - D_h)^2 \right), \quad (3-10) \]

where \( \mu \) is the cost function, \( h \) is the path index of the OBFN, \( M \) is the number of paths of the OBFN, \( k \) is the \( k \)-th frequency in the given frequency range \( P \).
3-3-2 Phase Criterion

Another way to define a criterion is by looking at the phase response instead of the group delay. The phase response for a single ORR is

\[ \psi(f) = \arctan \left( \frac{\sin(2\pi f T + \phi_i)}{\sqrt{1 - \kappa_i - \cos(2\pi f T + \phi_i)}} \right) - \arctan \left( \frac{\sqrt{1 - \kappa_i} \sin(2\pi f T + \phi_i)}{1 - \sqrt{1 - \kappa_i} \cos(2\pi f T + \phi_i)} \right). \]  

(3-11)

This equation (3-11) is equal to the equation (2-8) with \( r = 1 \).

The total phase response for one path is the sum of the \( \psi(f) \) for every ORR in that respective path. The total phase \( \psi_{total} \) for one path consisting of \( N \) ORRs is defined by

\[ \psi_{total}(f) = \sum_{i=1}^{N} \left[ \arctan \left( \frac{\sin(2\pi f T + \phi_i)}{\sqrt{1 - \kappa_i - \cos(2\pi f T + \phi_i)}} \right) - \arctan \left( \frac{\sqrt{1 - \kappa_i} \sin(2\pi f T + \phi_i)}{1 - \sqrt{1 - \kappa_i} \cos(2\pi f T + \phi_i)} \right) \right]. \]  

(3-12)

As has been shown in equation (2-9), that the relation between phase response \( \varphi \) and group delay \( \tau \) is given by

\[ \tau(f) = -\frac{1}{2\pi} \frac{\partial \varphi}{\partial f} \]

so we know that

\[ \varphi = -2\pi f \tau. \]  

(3-13)

Given the desired group delay for one specific path \( D_h \) and the frequency range \( f_k \), applying equation (3-13) to determine the desired phase delay for that specific path results in

\[ \psi_h = -2\pi f_k D_h. \]  

(3-14)

Then, the cost function we want to minimize is

\[ \mu(\varphi_1, \kappa_1, \varphi_2, \kappa_2, \cdots) = \sum_{h=1}^{M} \left( \sum_{k=1}^{P} \left( \psi_{total,h}(f_k) + 2\pi f_k D_h \right) \right)^2 \]  

(3-15)

3-3-3 Power Criterion

The power criterion is based on the fact that the ideal (desired) power would be reached when all the phases of different signals received by antenna elements are equal [2, p. 25]. The cost function now measures the loss of power due to the differences in phases, and can be expressed by subtracting the actual output power from the ideal one.

The ideal and actual power for one path are defined by

\[ P_{ideal} = \sum_{i=1}^{N} \sum_{k=1}^{P} a_i |H_i(f_k)| \]  

(3-16)

\[ P_{act} = \sum_{i=1}^{N} \sum_{k=1}^{P} a_i |H_i(f_k)| e^{j(\psi_{total,h}(f_k) + 2\pi f_k D_h)} \]  

(3-17)
where $i$ specifies the $i$-th ORR in one path consisting of $N$ ORRs, $k$ specifies the $k$-th frequency in the given frequency range $P$, and $a_i$ is a weighting factor for each different ORR that ranges from $(0, \cdots, 1)$. In this thesis, $a_i$ is assumed to be equal to one [2].

Then, the cost function we want to minimize is

$$
\mu(\phi_1, \kappa_1, \phi_2, \kappa_2, \cdots) = \sum_{h=1}^{M} [P_{\text{ideal}_h} - P_{\text{act}_h}]^2
$$

(3-18)

where $h$ is the path index and $M$ is the number of paths of the OBFN.

Comparing these three criteria, the delay criterion is the most straightforward. However, the delay error does not play a direct role in the systems output power because it cannot guarantee the maximum output power of the desired signal. Moreover, as far as the antenna pattern is concerned, the rise of side lobes which determines the normalized residual null level (output signal level for an undesired direction) also does not directly depend on the delay error but on the phase error [17, 4]. Therefore, by using the phase criterion, the negative effect of antenna side lobes can be minimized. Even so, the phase criterion still cannot guarantee the maximum output power of the desired signal. Theoretically, the power criterion leads to the best system performance [3, p. 155f.]. It is also measurable in real systems.

### 3-4 Current Optimization Method

Now, after the optimization criteria have been introduced, we can discuss the method of optimization. Non-linear programming (NLP) is the method which is currently being used to tune OBFNs [2]. That is, non-linear programming [27] is used to find the minimum value of a cost function subject to several constraints. The general form of the optimization problem is

$$
\min_{x} \, f(x) \\
\text{s.t.} \quad l \leq x \leq u, \quad g_i(x) = 0, \quad h_i(x) \leq 0.
$$

(3-19)

The vector $x$ contains the parameters which need to be tuned to minimize the function $f(x)$. The vector $l$ and $u$ (called constraint vectors) contain the lower bound and upper bound for the parameter vector $x$. The functions $g_i(x) = 0$ and $h_i(x) \leq 0$ are the constraint functions.

In MATLAB, this algorithm is implemented in the function called **fmincon** [28].

The specification of the constraints in the OBFN system is quite trivial. The value of $\kappa$ must be in the range between 0 and 1, and the $\phi$ are limited from 0 to $2\pi$. This is due to the fact that the phase shift of more than $2\pi$ is theoretically not meaningful [2, p. 27f.].

One of the drawbacks of NLP is that it is a black-box solver which cannot exploit the special structure of OBFNs. This drawback makes NLP not suitable for tuning large-scale OBFNs. The optimization process with a typical non-linear programming (NLP) solving method such as semi-quadratic programming method [29] is computationally expensive. Therefore, a new method for tuning is needed to handle large-scale OBFNs. In the next chapter, concepts from deep learning are discussed as a starting point for a more efficient method.
Chapter 4

Principles of Deep Neural Networks

An artificial neural network, commonly referred to as “neural network”, is a model which consists of a set of computational cells/units/neurons and a set of one way data connections joining units. Each connection has a number called weight. The connections and their weights are important parameters in any neural network model. They determine the behavior of the model, and can be compared to instructions in a conventional computer program [30, p. 3]. Figure 4-1 illustrates a single neuron which first sums the inputs and then applies a non-linear activation function.

Figure 4-1: Non-linear model of a neuron. Adapted from [7].

Let the neuron have $N$ inputs. The inputs to the neuron are denoted by $x_i$ and the weights for each input are denoted by $w_i$ where $i = 1, \cdots, N$. The output of the neuron is

$$y = f\left(\sum_{i=1}^{N} w_i x_i \right), \quad (4-1)$$

Typical examples of non-linear activation functions are the threshold function and the sigmoid function [8, p. 13f.]. The threshold function (see Figure 4-2a) is described as

$$f(s) = \begin{cases} 
1 & \text{if } s \geq 0 \\
0 & \text{if } s < 0
\end{cases} \quad (4-2)$$
The Sigmoid function (see Figure 4-2b), whose graph is “S”-shaped, is by far the most common form of activation function used in the construction of neural networks [8, p. 14]. It is defined by

\[ f(s) = \frac{1}{1 + e^{-as}} , \]  

where \( a \) is the slope parameter of the sigmoid function. By varying \( a \), different sigmoid functions with different slopes will be obtained. In the limit of \( a \) approaching infinity, the sigmoid function becomes the threshold function. Another function which can be used as activation function is the tangent hyperbolic function, \( f(s) = \tanh(s) \).

Neurons are grouped into layers, called input layer, output layer and hidden layers. Input layers consist of some neurons which are directly connected to the input signal. Output layers are sets of neurons which compute the output of the network. Between input layer and output layer, there are hidden layer(s), which connect the input layer to the output layer. There may be only one hidden layer, in which case the network is called shallow, or multiple layers, in which case the network is called deep [8, p. 21]. Figure 4-3 illustrates a simple neural network structure.
Neural networks work very well on many current problems. Problems in pattern recognition, voice recognition, hand writing recognition, function approximation and control have been solved well by means of neural networks [8, p. 38ff.]. To achieve a good performance, suitable weights which attain a desired design objective must be found [8, p. 34ff.].

4-1 Common Types of Neural Networks

Neural networks can be grouped by their structure (architecture). The most common types of neural networks are feedforward neural networks, recurrent neural networks and convolutional networks [8, p. 21ff.].

4-1-1 Feedforward Neural Networks

In feedforward neural networks, the input vector goes into the input layer, is then passed through the hidden layers, up until the output layer, without any connection to previous layers. The illustration of feedforward neural network is shown in Figure 4-4a.

Given a vector input $x^{(n-1)}$ going into a specific layer $n$ and the weights matrix of the layer $W^{(n)}$, then the output of the layer is equal to:

$$x^{(n)} = F(W^{(n)}x^{(n-1)}), \quad (4-4)$$

where $F$ applies the non-linear activation function used in the neural network to each element of its input vector. The equation (4-4) also shows that the output of the layer $n$ is then used as an input to the next layer $n+1$.

4-1-2 Recurrent Neural Networks

Recurrent neural networks distinguish themselves from feedforward neural networks in that they have at least one feedback loop [8, p. 23]. The illustration is shown in Figure 4-4b. In that figure, there are feedback loops from output layer to hidden layers or from one hidden layer to another hidden layer before. The formula (4-1) and the types of activation function used in feedforward neural networks are applicable also in recurrent neural networks.

![Figure 4-4](image-url)
4-1-3 Convolutional Networks

Convolutional networks are a special type of feedforward neural network designed specifically to recognize two dimensional shapes with high degrees of invariance to translation, scaling, skewing, and other forms of distortions [8, p. 201]. Since the introduction of convolutional networks by LeCun et al. [31] in the early 1990’s, they have demonstrated excellent performance at difficult tasks such as hand-written digit classification, face detection [32, p. 818] and high resolution image recognition and classification [33, 34].

A convolutional neural network consists of several layers. These layers can be of three types:

**Convolutional**

The convolutional layer is just a convolution of the previous layer, where the weights specify the convolution filter. Convolutional layers consist of neurons which take the inputs from local (nearby) receptive field (neurons) in the previous layer [8, p. 201f.]. See Figure 4-5a for the illustration of nearby neurons. This kind of connection is called sparse connectivity (see Figure 4-5b). The layer is composed of multiple feature maps, with each feature map being in the form of a plane within which the individual neurons are constrained to share the same weights [8, p. 201f.] (see Figure 4-5c).

![Figure 4-5: Sparse connectivity and shared weight illustration [10]](image)

**Max-Pooling**

After each convolutional layer, there may be a pooling layer. The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block. There are several ways to do this pooling, such as taking the average or the maximum, or a learned linear combination of the neurons in the block. Max-pooling layers take the maximum of the block they are pooling in order to reduce variance. This will ensure that the same result will be obtained, even when features have small translations. This is an important operation for object classification and detection.

**Fully-Connected**

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. A fully connected layer takes all neurons in
4-2 Training Neural Networks using Gradient Descent

The two most popular methods of training neural networks are supervised and unsupervised training. Supervised training is the process to train a neural network with knowledge of specific desired input-output pairs. Typical tasks in supervised training are classification and regression [35, p. 1]. Unsupervised training, on the other hand, is self-organized learning in which there are still inputs but without the desired outputs. Unsupervised training is used typically in tasks such as clustering data into similarity groupings [30, p. 133]. The training discussed in this section refers to supervised learning.

In supervised learning, the desired response (value) is provided to the network, which will be compared to the actual response of the network from the inputs provided. The error, as measured by a cost function, is minimized by adjusting the weights of the network. The adjustments are carried out iteratively in many cases based on the gradient of the cost function with respect to the weights. There are actually many ways to minimize the cost function. However, due to the large dimension of many neural networks, usually some form of gradient descent is used because of its simplicity.

4-2-1 The Concept of Backpropagation

Backpropagation is an algorithm for computing the gradient of the cost function with respect to the weights of the network efficiently. It employs a nice simplification that makes it easy to find the network's weight gradient, which makes the problem computationally feasible. The term “back-propagation” appears to have evolved after 1985, when the term error correcting rule was popularized through the publication of *Parallel Distributed Processing* (Rumelhart et al. [36, p. 63]). At the heart of backpropagation is an expression for the partial derivative $\frac{\partial C}{\partial w}$ of the cost function $C$ with respect to any weight $w$ in the network. The calculation of the derivative is done in a backward pass (as shown in Figure 4-7) by applying the chain rule.
The algorithm cycles through the training samples as indicated in Algorithm 1:

**Algorithm 1** Backpropagation algorithm

1. **Initialization:** Initialize all weights in the network
2. repeat
   - For every data (pairs of input and desired output) in the training set
     - Present the input to the network
   - **Propagate the input forward through the network to get the actual output:**
     - For each layer in the network
       - For every neuron in the layer
         - Calculate the sum of the inputs to the neuron
         - Calculate the activation function for the neuron
     - End
   - **Propagate the errors backward through the network:**
     - Calculate the cost function (difference between the desired and actual output)
     - For all hidden layers
       - For every neuron in the layer
         - Calculate the partial derivative of the cost function with respect to the weights of each neuron
         - Update each neuron’s weight in the network via gradient descent
     - End
   - Calculate Global Error
     - Calculate the Error
3. until (maximum number of iterations > than specified) OR (Error is < than specified)

**Example: Backpropagation for a feedforward neural network**

Consider a simple feedforward neural network with *N* layers and two neurons per layer, as illustrated in Figure 4-8.

Let the scalar non-linear activation function for each neuron be defined as

\[ f : \mathbb{R} \rightarrow \mathbb{R}, \quad \text{e.g.,} \quad f(s) = \tanh(s). \]  \hspace{1cm} (4-5)

Its derivative is denoted by

\[ f' := \frac{\partial f}{\partial s}. \]
We define
\[
F \left( \begin{bmatrix} u_1 \\ \vdots \\ u_d \end{bmatrix} \right) := \begin{bmatrix} f(u_1) \\ \vdots \\ f(u_d) \end{bmatrix}
\]  
(4-6)

The input to the neural network is defined as \( x^{(0)} \in \mathbb{R}^2 \).

The layers propagate the input using weight matrices
\[
W^{(n)} = \begin{bmatrix} w_{1,1}^{(n)} & w_{1,2}^{(n)} \\ w_{2,1}^{(n)} & w_{2,2}^{(n)} \end{bmatrix}
\]  
as follows:
\[
y^{(n)} = \begin{bmatrix} y_1^{(n)} \\ y_2^{(n)} \end{bmatrix} = W^{(n)} x^{(n-1)},
\]  
(4-7)
\[
x^{(n)} = \begin{bmatrix} x_1^{(n)} \\ x_2^{(n)} \end{bmatrix} = F(y^{(n)}).
\]  
(4-8)

The output of the neural network is \( x^{(N)} \).

Let the desired output \( d = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}^T \in \mathbb{R}^2 \) and the input \( x^{(0)} \in \mathbb{R}^2 \) of the network be given.

We aim to minimize the cost function:
\[
C(W^{(1)}, \ldots, W^{(N)}) = \frac{1}{2} \left( (d_1 - x_1^{(N)})^2 + (d_2 - x_2^{(N)})^2 \right)
\]  
(4-9)
via gradient descent, i.e.
\[
w_{i,j}^{(n)} \leftarrow w_{i,j}^{(n)} - \lambda \frac{\partial C}{\partial w_{i,j}^{(n)}}
\]  
(4-10)
where \( \lambda > 0 \) is a sufficiently small step-size. Therefore we need to find the partial derivatives \( \frac{\partial C}{\partial w_{i,j}^{(n)}} \).

**Remark:** In the case of multiple training pairs, the cost function will be
\[
C(W^{(1)}, \ldots, W^{(N)}) = \frac{1}{2} \sum_{\alpha=1}^{S} \left\| d[\alpha] - x^{(N)}[\alpha] \right\|^2,
\]  
(4-11)
Gradient for the last layer (layer $N$)

From equation (4-9), we can determine the partial derivative of the cost function with respect to $x_k^{(N)}$ as follows:

$$\frac{\partial C}{\partial x_k^{(N)}} = -(d_k - x_k^{(N)})$$  \hspace{1cm} (4-13)

Using the chain rule, the partial derivative of the cost function with respect to $w_{i,j}^{(N)}$ is given by

$$\frac{\partial C}{\partial w_{i,j}^{(N)}} = \frac{\partial C}{\partial x_1^{(N)}} \frac{\partial x_1^{(N)}}{\partial w_{i,j}^{(N)}} + \frac{\partial C}{\partial x_2^{(N)}} \frac{\partial x_2^{(N)}}{\partial w_{i,j}^{(N)}}$$

$$= \frac{\partial C}{\partial x_1^{(N)}} f'(w_{i,1}^{(N)} x_{1}^{(N-1)} + w_{i,2}^{(N)} x_{2}^{(N-1)}) x_{j}^{(N-1)}$$

$$+ \frac{\partial C}{\partial x_2^{(N)}} f'(w_{i,1}^{(N)} x_{1}^{(N-1)} + w_{i,2}^{(N)} x_{2}^{(N-1)}) x_{j}^{(N-1)}$$

$$= \frac{\partial C}{\partial x_{x_i}^{(N)}} f'(y_i^{(N)}) x_{j}^{(N-1)}.$$  \hspace{1cm} (4-14)

Gradient for layer $n - 1$

Let $n$ be any number in $\{2, \cdots, N\}$. The partial derivative of the cost function $C$ with respect to $x_k^{(n-1)}$ is

$$\frac{\partial C}{\partial x_k^{(n-1)}} = \frac{\partial C}{\partial x_1^{(n)}} \frac{\partial x_1^{(n)}}{\partial x_k^{(n-1)}} + \frac{\partial C}{\partial x_2^{(n)}} \frac{\partial x_2^{(n)}}{\partial x_k^{(n-1)}}$$

$$= \frac{\partial C}{\partial x_1^{(n)}} f'(w_{1,1}^{(n)} x_{1}^{(n-1)} + w_{1,2}^{(n)} x_{2}^{(n-1)}) w_{1,k}^{(n)}$$

$$+ \frac{\partial C}{\partial x_2^{(n)}} f'(w_{2,1}^{(n)} x_{1}^{(n-1)} + w_{2,2}^{(n)} x_{2}^{(n-1)}) w_{2,k}^{(n)}$$

$$= \frac{\partial C}{\partial x_{x_1}^{(n)}} f'(y_1^{(n)}) w_{1,k}^{(n)} + \frac{\partial C}{\partial x_{x_2}^{(n)}} f'(y_2^{(n)}) w_{2,k}^{(n)}.$$  \hspace{1cm} (4-15)
The partial derivative of cost function with respect to $w^{(n-1)}_{i,j}$ is

$$
\frac{\partial C}{\partial w^{(n-1)}_{i,j}} = \frac{\partial C}{\partial x^{(n-1)}_1} \frac{\partial x^{(n-1)}_1}{\partial w^{(n-1)}_{i,j}} + \frac{\partial C}{\partial x^{(n-1)}_2} \frac{\partial x^{(n-1)}_2}{\partial w^{(n-1)}_{i,j}}
$$

$$
= \frac{\partial C}{\partial x^{(n-1)}_1} \frac{\partial f}{\partial w^{(n-1)}_{i,j}} \left( w^{(n-1)}_{i,1} x_1^{(n-2)} + w^{(n-1)}_{i,2} x_2^{(n-2)} \right)
$$

$$
= \frac{\partial C}{\partial x^{(n-1)}_i} \frac{\partial f}{\partial y^{(n-1)}_i} y^{(n-1)}_i x_j^{(n-2)}.
$$

(4-16)

**Backpropagation algorithm**

The inputs to the algorithm are the input to the network $x^{(0)}$ and the desired output $d$. The outputs of the algorithm are the partial derivatives of the cost function with respect to the weights $\frac{\partial C}{\partial w^{(n)}_{i,j}}$ for all $i, j$ and $n$. The backpropagation algorithm runs as follows:

1. Compute $y^{(n)}$ and $x^{(n)}$ for $n = 1, \cdots, N$ in a forward pass using equation (4-7) and (4-8).

2. Compute $\frac{\partial C}{\partial x^{(n)}_k}$ for $k \in \{1, 2\}$ using equation (4-13).

3. Compute $\frac{\partial C}{\partial w^{(n)}_{i,j}}$ for $i, j \in \{1, 2\}$ using equation (4-14).

4. For $n = N, N-1, \cdots, 2$:
   
   (a) Compute $\frac{\partial C}{\partial x^{(n-1)}_k}$ for $k \in \{1, 2\}$ using equation (4-15).

   (b) Compute $\frac{\partial C}{\partial w^{(n-1)}_{i,j}}$ for $i, j \in \{1, 2\}$ using equation (4-16).

**4-2-2 Some Breakthroughs in Deep Learning**

Neural networks with huge numbers of weights and multiple layers are very powerful machine learning systems. However, large networks with multiple hidden layers and highly non-linear activation functions are computationally expensive to train. Moreover, the optimization algorithms can be trapped in local optima. Deep neural networks also have serious problems with overfitting, which occurs when the network is too closely fit to a limited set of data points so that it will make poor predictions on new data. All of those challenges made the training of deep neural networks used to not work well before 2006 [37, 38].

One essential reason why deep neural networks used to not perform well in the past was because the learning process initialized the weights randomly. Random initializations work
poorly with deep neural networks because gradient descent suffers from vanishing/exploding
gradients [37, p. 249].

One breakthrough in deep neural networks was to initialize the weights close to a good
solution using unsupervised learning. Each layer is pretrained with an unsupervised learning
algorithm, learning a nonlinear transformation of its input (the output of the previous layer)
that captures the main variations in its input [39, p. 626f.]. Hinton et al. [40] in 2006 proposed
a method for initialization of Deep Belief Networks (DBNs) using a model called Restricted
Boltzmann Machine (RBM) [41, 42]. Bengio et al. [38] and LeCun et al. [43] in 2007 proposed
another initialization method based on auto-encoders. Another breakthrough is dropout
regularization [14], which reduces overfitting. Each breakthrough will be explained in more
detail in the following sections.

4-3 Deep Belief Networks

The idea of initialization of the weights of deep neural networks is called pre-taining, and Deep
Belief Networks (DBNs) are major example of this initialization [40]. A DBN is an unsupervised
generative model that mixes undirected and directed interactions between variables or
neurons.

Note: Ng and Jordan [44, p. 1f.] explains the difference between generative and discriminative
models. Generative models are models of the joint probability $p(a, b)$ of the state $a$ and the
state $b$. It refers to the probability of the intersection of state $a$ and state $b$ occurring.
Discriminative models are models of the conditional probability $p(b|a)$ of state $a$ and state
$b$. It refers to the probability of state $b$ is occurring given the state $a$ is occurring.

The top two layers, which have undirected interaction is a Restricted Boltzmann Machine
(RBM), while the other layers are directed networks called Logistic Belief Networks (LBNs).
Figure 4-9 illustrates the graphical model of a DBN and briefly explains RBM, LBN, undi-
rected and directed connections. $x$ represents the input layer, while $h^{(n)}$ represents the $n$-th
hidden layer. DBNs are build on a neural network structure that was first described by
Smolensky [45] in 1986. At that time this structure was referred to as a “harmonium”. It was
later renamed as RBM by Hinton et al. [40].

4-3.1 Restricted Boltzmann Machine

A Boltzmann machine is a stochastic binary machine whose composition consists of stochastic
neurons, as mentioned by Aarts and Korst [46]. A stochastic neuron resides in one of two
possible states in a probabilistic manner. These two states may be designated as 1 for the
ON state and 0 for the OFF state (binary units).

The stochastic neurons of the Boltzmann machine are partitioned into two functional groups,
called visible and hidden neurons (see Figure 4-10a). The visible neurons provide an interface
between the network and the environment in which it operates. The hidden neurons on the
other hand, always operate freely, they are used to explain underlying constraints contained
in the environmental input vectors [8, p. 598ff.].

The restricted Boltzmann machine is a special type of Boltzmann machine in which there are
no connections between neurons in the same hidden or visible layers, as shown in Figure 4-10b.
Since there are no connections between neurons in the same hidden and visible layers and the connection between visible neurons and hidden neurons is undirected, it follows that the states of the hidden neurons are conditionally independent of each other. Therefore, RBM is able to extract unbiased samples from the posterior distribution, given a data vector is clamped (connected) onto the visible neurons, according to Salakhutdinov and Hinton [42].

The weights obtained from training the DBN can be used as an initial guess of the weights. According to Hinton et al. [40], this initialization is close to a good solution. However, with the binary units introduced for RBM and DBN, some tricks are needed to implement this method for continuous inputs which are more generally used. One can “cheat” and handle the continuous inputs by scaling them to the (0,1) interval and considering each continuous input as the probability for a binary random variable to take the value 1 [38]. Previous work on continuous inputs in RBM as mentioned in Chen and Murray [47] may be useful as well.

Regardless of being powerful in initializing the weights, RBMs are considered difficult because they are probabilistic models. Recent studies have found that a simple deterministic model using auto-encoders also works well to pre-train deep neural network, as mentioned in [38, 43]. Therefore, further explanations of RBM and training of DBN will not be covered in this thesis.
4-4 Auto-encoder

Bengio et al. [38] and LeCun et al. [43] verified that the unsupervised pretraining principle can be applied using auto-encoders instead of RBMs as a layer building block. The concept is similar to DBN as explained in section 4-3, only that the RBM is replaced by an auto-encoder. Auto-encoders are artificial neural networks whose function is to reconstruct their inputs on the output layer, as illustrated in Figure 4-11.

![Illustration of auto-encoder. Adapted from [13]](image)

In order to understand how auto-encoders can be used for pre-training neural networks, one simple case will be discussed. Given a feedforward neural network with \( N \) layers, the layers propagate the input as follows:

\[
x^{(n)} = F(W^{(n)}x^{(n-1)}), \quad \text{where} \quad n = 1, \ldots, N; \quad W^{(n)} \in \mathbb{R}^{n_1^{(n)} \times n_2^{(n)}}
\]

We want to train layer \( k \) in order to get the weight \( W^{(k)} \). The input to layer \( k \) is \( x^{(k-1)} \). We introduce artificial weights \( \tilde{W}^{(k+1)} \in \mathbb{R}^{n_2^{(k)} \times n_1^{(k)}} \).

We construct the artificial neural network as follows:

\[
\tilde{x}^{(k+1)} = F(\tilde{W}^{(k+1)}x^{(k)}) = F(\tilde{W}^{(k+1)}F(W^{(k)}x^{(k-1)}))
\]

The artificial neural network can be trained like a conventional feedforward neural network, e.g. by minimizing the cost function

\[
C(W^{(k)}, \tilde{W}^{(k+1)}) = \frac{1}{2} \sum_{\alpha=1}^{S} \left\| \tilde{x}^{(k+1)[\alpha]} - x^{(k-1)[\alpha]} \right\|^2,
\]

with backpropagation. Here the index \( \alpha \) specifies the individual training sample, \( S \) denotes the total number of training samples and \( x^{(k)[\alpha]} \) is the input to the \( k \)-th layer generated by the training sample \( x^{(0)}[\alpha] \).

The artificial weights \( \tilde{W}^{(k+1)} \) will be discarded after the pretraining. The auto-encoder is said to have tied weights if in addition the constraint \( \tilde{W}^{(k+1)} = W^{(k)T} \) is enforced during pretraining [48]. Figure 4-12 and 4-13 illustrate how the auto-encoder is used for pre-training.

One question that arises with auto-encoders in comparison with RBMs is whether the auto-encoder will fail to learn useful representations when the number of units is not strictly decreasing from one layer to the next. The network could just be identity and perfectly...
reconstruct the inputs. However, Bengio et al. [38] found that the network with non-decreasing layer size works well. This is due to the fact that the weight decay and stochastic gradient descent prevent large weights: the optimization falls in a local minimum which corresponds to a good transformation of the input.

![Diagram](image1.png)

**Figure 4-12:** Example of deep neural network [13]

![Diagram](image2.png)

**Figure 4-13:** Auto-encoder used for pre-training [13]

There are some special cases of auto-encoders: denoising auto-encoders and sparse auto-encoders. Denoising auto-encoders are a type of auto-encoders with the addition of artificial noise on the inputs [48]. Sparse auto-encoders are a type of auto-encoders where sparsity (forcing many of the hidden units to be zero or near-zero) is enforced [49, 50].
4-5 Dropout Regularization

Dropout is a technique that prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently [14, 51]. Deep neural networks with multiple non-linear hidden layers are, as has been explained before, prone to overfitting. Many methods have been proposed to reduce overfitting, including stopping the training as soon as the performance on a validation set starts to get worse [52], introducing weight penalties of various kinds such as L1 and L2 regularization [53] and soft weight sharing [54]. With unlimited computational resources, the best way to regularize a model is to average the predictions of all possible settings of parameters [14, p. 1929]. This can sometimes be quite easy for simple models. However, for deep neural networks with a large number of hidden layers, the idea of averaging becomes computationally expensive. Dropout provides a more efficient way for this issue.

![Standard neural network](a) Standard neural network ![After applying dropout](b) After applying dropout

Figure 4-14: Dropout neural network model [14].

The term “dropout” itself refers to temporarily dropping out units (can be hidden or visible) in a neural network, along with all the incoming and outgoing connections, as shown in Figure 4-14. The choice of which units are dropped is random with probability $p$. The value of $p = 0.5$ seems to be close to optimal for a wide range of networks and tasks [14, p. 1930]. Input units can be dropped as well, but with different choice of $p$. The optimal probability of retention $p$ for input units usually closer to 1 than to 0.5 [14, p. 1930].

After applying dropout, the remaining survived network is called thinned network. With unlimited computational resources and training data, a neural network of $n$ units can form $2^n$ possible thinned neural networks which all share weights. Training neural network with dropout can be seen as training $2^n$ thinned networks with extensive weight sharing [14, p. 1931]. Combining all $2^n$ training is also effectively done in dropout regularization. The idea is that the neural network with scaled down weights (half) approximates the geometric areas of these models [14, p. 1931]. This ensures that for any hidden unit the expected output (under the distribution used to drop units at training time) is the same as the actual output at test time.

Herminarto Nugroho Master of Science Thesis
Dropout neural networks can be trained with stochastic gradient descent [55] [51, p. 4]. Each of the thinned networks is updated using only a small part of the training data (minibatches) which is chosen randomly. This will make the training of neural networks computationally cheap [56, p. 2ff.].

Training networks with dropout regularization and using the approximate averaging method explained before leads to a lower generalization error on a wide variety of problems compared to other methods [51, 14].

**Example: Dropout regularization for a feedforward neural network**

Consider a simple feedforward neural network with \( N \) layers and four neurons per layer except for the output layer which has only one neuron, as illustrated in Figure 4-15. Let the scalar non-linear activation function \( f \), its derivative \( f' \) and \( F \) be defined as in the example in subsection 4-2-1.

![Simple example of dropout for feedforward neural network](image)

The input to the neural network is defined as \( x^{(0)} \in \mathbb{R}^4 \).

The first \( N - 1 \) layers propagate the input as follows:

\[
\begin{align*}
x^{(n)} &= \begin{bmatrix} x_1^{(n)} \\ x_2^{(n)} \\ x_3^{(n)} \\ x_4^{(n)} \end{bmatrix} = F(\mathbf{W}^{(n)} x^{(n-1)}), \quad \text{where} \quad \mathbf{W}^{(n)} = \begin{bmatrix} w_{1,1}^{(n)} & w_{1,2}^{(n)} & w_{1,3}^{(n)} & w_{1,4}^{(n)} \\ w_{2,1}^{(n)} & w_{2,2}^{(n)} & w_{2,3}^{(n)} & w_{2,4}^{(n)} \\ w_{3,1}^{(n)} & w_{3,2}^{(n)} & w_{3,3}^{(n)} & w_{3,4}^{(n)} \\ w_{4,1}^{(n)} & w_{4,2}^{(n)} & w_{4,3}^{(n)} & w_{4,4}^{(n)} \end{bmatrix} \\
\end{align*}
\]

(4-20)

where \( n \) is any number in \( \{1, \cdots, N - 1\} \).

The last (output) layer computes the output as follows:

\[
y = x^{(N)} = F(\mathbf{W}^{(N)} x^{(N-1)}), \quad \text{where} \quad \mathbf{W}^{(N)} = \begin{bmatrix} w_{1,1}^{(N)} & w_{1,2}^{(N)} & w_{1,3}^{(N)} & w_{1,4}^{(N)} \end{bmatrix}
\]

(4-21)
Applying dropout for the first $N-1$ layers

Introduce random binary masks $m^{(n)} \in \mathbb{R}^4$ in which the elements are either 0 or 1. The probability of their elements having value 1 is $p = 0.5$. By implementing the Hadamard product $A \circ B$, where the $(i, j)$ element of a matrix $A \in \mathbb{R}^{m \times n}$ is multiplied with the $(i, j)$ element of a matrix $B \in \mathbb{R}^{m \times n}$, the dropping procedure of the neurons for layer $n$ is

$$x^{(n)} = F\left(W^{(n)} (x^{(n-1)} \circ m^{(n)})\right).$$

(4-22)

**Note:** The random binary mask for the input layer $m^{(1)} \in \mathbb{R}^4$ is different. The probability of their elements being 1 is closer to 1 than to 0.5.

Given the training inputs, weights $\tilde{W}^{(1)}, \ldots, \tilde{W}^{(N)}$ can be found by minimizing the cost function similar as described in example in subsection 4-2-1 using stochastic gradient descent. Note that for each iteration of the dropout training, different random binary masks are used.

The combined weights used in the end will be

$$W^{(n)} := p^{(n)} \tilde{W}^{(n)},$$

(4-23)

where $p^{(n)}$ is the probability of the retention of the neurons in layer $n$. 

---

Herminarto Nugroho

Master of Science Thesis
Chapter 5

Deep Neural Network Representations of Optical Beamforming Networks

Ideas from deep learning, such as backpropagation, auto-encoders, and dropout regularization, seem to be promising for tuning large-scale OBFNs. In this chapter, therefore a deep neural network representation of the OBFN system will be presented.

In the first section of this chapter, neural network configurations which represent the special structure of OBFNs will be presented. How to generate training examples which consist of input vectors and their respective desired output vectors will also be explained. The choice of the activation function and the derivation of weight matrices will be discussed afterwards. The backpropagation and complex parameters problem will complete the first section. The complex parameters problem is that the optimization will likely result in complex parameters $\kappa$ and $\phi$. In the second section, a new approach based on the separation of complex numbers will be introduced to solve the complex parameters problem, followed by the modifications needed to implement this approach.

5-1 Neural Network Configurations for OBFN Systems

5-1-1 Feed-forward Neural Network-based OBFN Systems

The first important thing to consider in order to be able to implement ideas from deep learning is to determine the suitable form of the deep neural network which represents the OBFN correctly.

Figure 5-1 shows the binary-tree configuration of a $4 \times 1$ OBFN system and the neural network representation used in this thesis. The dotted lines represent the layer separation. The topmost path of the OBFN without any ORR is called the reference path. The signal coming to this path is not delayed. Therefore, it will later be used to determine the desired output of the network. Given $D_3 > D_2 > D_1 > 0$ as time differences between paths, it means that
the signal of bottom most path arrives the earliest compared to others and signal of reference path arrives the latest.

There are three types of weights that can be observed from Figure 5-1b. They are the identity weight ($I$), the frequency response ($H_i$), and zero weight which is illustrated by gray lines. Only frequency response weights will be updated via gradient descent, while the other two remain the same. Note that one ORR can be used in more than one path, as we can see in path 3 and 4 in which there are two ORRs being used together.

Based on numerical simulations, this neural network representation is able to learn to obtain reasonable values $\kappa^*$ and $\phi^*$ for all ORRs. Another neural network representation in which we include the combiner after every OBFN stage (see Figure 5-2) did not work well and is therefore not discussed in this thesis.

Figure 5-2: The neural network configuration of a $4 \times 1$ OBFN with combiner after every stage.

A way of constructing neural network similar to Figure 5-1 can be applied for larger OBFNs.
Figure 5-3: (a) An $8 \times 1$ OBFN system. (b) The neural network configuration of an $8 \times 1$ OBFN.
Figure 5-3 shows the binary-tree configuration of an $8 \times 1$ OBFN and its neural network representation. By following the same way of constructing the neural network as illustrated in Figure 5-1 and Figure 5-3, we can create a neural network representation of $2^\beta \times 1$ OBFN where $\beta$ specifies any real positive number.

5-1-2 Generating Training Examples

Training examples are sets of input vectors of a certain neural network and their respective desired output vectors. From Figure 5-1 and Figure 5-3, the input to the network is the signal received by each AE. The signal received by the reference path will be used as the desired output, the other inputs are the time-delayed versions of it.

In this project, it is assumed that the input signal is a signal that comes from a very far source (satellite) without any aberrations from the atmosphere. Noise signals and other signals coming from different directions are assumed to be very small and are omitted. The wavefront of the incoming signal is flat and therefore the input signals arriving at each antenna element are parallel to each other.

The input in time domain is shown in Figure 5-1a and Figure 5-3a. The Fourier transformation is used to transform the time domain input into frequency domain, as follows:

\[
\begin{align*}
    x_1^{(0)}(f) &= \mathcal{F}\{s(t)\}, \\
    x_2^{(0)}(f) &= \mathcal{F}\{s(t + D_1)\} = x_1^{(0)}(f)e^{i2\pi f D_1}, \\
    x_3^{(0)}(f) &= \mathcal{F}\{s(t + D_2)\} = x_1^{(0)}(f)e^{i2\pi f D_2}, \\
    x_4^{(0)}(f) &= \mathcal{F}\{s(t + D_3)\} = x_1^{(0)}(f)e^{i2\pi f D_3}, \\
    \vdots \\
    x_M^{(0)}(f) &= \mathcal{F}\{s(t + D_{M-1})\} = x_1^{(0)}(f)e^{i2\pi f D_{M-1}}. 
\end{align*}
\]

(5-1)

From Equation (5-1), the input is frequency dependent, i.e., different frequencies result in different inputs. OBFN systems in this project are aimed to maximize the output signal strength by obtaining desired group delay responses over a certain frequency range. Therefore, an array of signals for a set of frequencies is needed to simulate the system.

Given a frequency range $\mathbf{f} \in \mathbb{R}^N = [f_1\ f_2\ f_3\ \cdots\ f_N]$, the input array of the reference path is defined by

\[
x_1^{(0)} = \begin{bmatrix} x_1^{(0)}(f_1) \\ x_1^{(0)}(f_2) \\ \vdots \\ x_1^{(0)}(f_N) \end{bmatrix} \in \mathbb{C}^N, 
\]

(5-2)

while for the $m$-th path, the array input is defined by

\[
x_m^{(0)} = \begin{bmatrix} x_m^{(0)}(f_1) \\ x_m^{(0)}(f_2) \\ \vdots \\ x_m^{(0)}(f_N) \end{bmatrix} = \begin{bmatrix} x_1^{(0)}(f_1)e^{i2\pi f_1 D_{m-1}} \\ x_1^{(0)}(f_2)e^{i2\pi f_2 D_{m-1}} \\ \vdots \\ x_1^{(0)}(f_N)e^{i2\pi f_N D_{m-1}} \end{bmatrix} \in \mathbb{C}^N. 
\]

(5-3)
In summary, consider a neural network structure in Figure 5-1b and Figure 5-3b consisting of \(M\) antenna elements (paths). The input to the neural network is defined by

\[
x^{(0)} = \begin{bmatrix}
x^{(0)}_1 \\
x^{(0)}_2 \\
\vdots \\
x^{(0)}_M
\end{bmatrix} \in \mathbb{C}^{MN} \quad \text{with} \quad x^{(0)}_i = \begin{bmatrix}
x^{(0)}_i(f_1) \\
x^{(0)}_i(f_2) \\
\vdots \\
x^{(0)}_i(f_N)
\end{bmatrix} \in \mathbb{C}^N, \quad (5-4)
\]

where \(N\) specifies the number of frequencies. The input \(x^{(0)}_m(f_n)\) specifies the signal which is received by the \(m\)-th antenna element, for frequency \(f_n\).

Consider a lossless system, i.e., power loss \(r = 1\) and gain response \(|H(f)| = 1\), then the desired output of the network becomes

\[
d = \begin{bmatrix}
x^{(0)}_1 \\
x^{(0)}_2 \\
\vdots \\
x^{(0)}_M
\end{bmatrix} \in \mathbb{C}^{MN} \quad \text{with} \quad x^{(0)}_1 = \begin{bmatrix}
x^{(0)}_1(f_1) \\
x^{(0)}_1(f_2) \\
\vdots \\
x^{(0)}_1(f_N)
\end{bmatrix} \in \mathbb{C}^N. \quad (5-5)
\]

For a lossy system, the magnitude (gain) responses of ORRs will be included into the calculation of the desired output as follows:

\[
d = \begin{bmatrix}
x^{(0)}_1 \\
|H|_{tot,2}x^{(0)}_1 \\
\vdots \\
|H|_{tot,M}x^{(0)}_1
\end{bmatrix} \in \mathbb{C}^{MN} \quad \text{with} \quad x^{(0)}_1 = \begin{bmatrix}
x^{(0)}_1(f_1) \\
x^{(0)}_1(f_2) \\
\vdots \\
x^{(0)}_1(f_N)
\end{bmatrix} \in \mathbb{C}^N, \quad (5-6)
\]

where \(|H|_{tot,m}\) specifies the total gain response of ORR for the \(m\)-th path, which is formulated by \(|H|_{tot,m} = |H|_{1,m} \times |H|_{2,m} \times \cdots \times |H|_{r,m}\) where \(r\) being the number of ORRs in that respective path. The formula for \(|H|\) is presented in Equation (2-7).

### 5-1-3 The Formulation of Weight Matrices

Consider an \(M \times 1\) binary tree OBFN, which is represented by a neural network that consisting of \(M \times N\) neurons and \(L\) layers, where \(N\) specifies the number of frequencies of interest. Input vectors are defined by Equation (5-4). The layers propagate the input using weight matrices \(W^{(l)} \in \mathbb{C}^{MN \times MN}\) as follows:

\[
y^{(l)} = W^{(l)}x^{(l-1)}, \quad (5-7)
\]

\[
x^{(l)} = F(y^{(l)}). \quad (5-8)
\]

where \(l\) specifies the layer index and \(F\) specifies the activation function, which is defined by

\[
F : \mathbb{R}^N \rightarrow \mathbb{R}^N, \quad F(s) = s. \quad (5-9)
\]

The structure of the weight matrices depends on the neural network representation of a certain OBFN. This is due to the fact that different neural network configurations lead to different

---

Master of Science Thesis
Herminarto Nugroho
locations of the frequency responses $H_j$ inside the weight matrices. However, determining the weight matrices from a given neural network configuration is actually straightforward.

For instance, from Figure 5-1b the weight matrices of a $4 \times 1$ OBFN are defined as follows:

$$W^{(1)} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & H_1 & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & H_4 \end{bmatrix} \in \mathbb{C}^{4N \times 4N},$$  

$$W^{(2)} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & H_2 & 0 \\ 0 & 0 & 0 & H_2 \end{bmatrix} \in \mathbb{C}^{4N \times 4N},$$  

$$W^{(3)} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & H_3 & 0 \\ 0 & 0 & 0 & H_3 \end{bmatrix} \in \mathbb{C}^{4N \times 4N},$$

where $0 \in \mathbb{R}^{N \times N}$ specifies the zero matrix, $I \in \mathbb{R}^{N \times N}$ specifies the identity matrix and $H_j$ specifies the frequency response matrix of ORR number $j$ which is defined by

$$H_j = \begin{bmatrix} H_j(f_1) & 0 & \cdots & 0 \\ 0 & H_j(f_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & H_j(f_N) \end{bmatrix} \in \mathbb{C}^{N \times N},$$

and the frequency response $H_j$ is presented in Equation (2-4).

For an $8 \times 1$ OBFN depicted in Figure 5-3, the weight matrices are defined as follows:

$$W^{(1)} = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & H_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & H_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & H_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_{12} \end{bmatrix} \in \mathbb{C}^{8N \times 8N},$$

$$W^{(2)} = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & H_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & H_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & H_{10} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_{10} \end{bmatrix} \in \mathbb{C}^{8N \times 8N},$$
5-1 Neural Network Configurations for OBFN Systems

The Non-linear Optimization

Tuning OBFNs with a number of ROs ORRs is a non-linear optimization problem. Given \( S \) number of training examples, the deep learning algorithm trains the \( L \) layers of the neural
network to obtain the optimum value of all ORRs’ parameters (both \( \kappa \) and \( \phi \)) by minimizing a certain cost function. The layers propagate the input to get the actual output \( \mathbf{x}^{(L)} \). We aim to minimize the cost function

\[
C(\kappa_1, \phi_1, \kappa_2, \phi_2, \ldots, \kappa_R, \phi_R) = \frac{1}{2} \frac{1}{U} \sum_{\alpha=1}^{U} \| \mathbf{d}[\alpha] - \mathbf{x}^{(L)}[\alpha] \|^2,
\]

subject to constraints

\[
\epsilon \leq \kappa_j^{(i)} \leq 1 - \epsilon, \\
0 \leq \phi_j^{(i)} \leq 2\pi.
\]

via stochastic gradient projection, with \( \epsilon > 0 \) is a small number, \( \alpha \) is the index of training examples, and \( U \) is number of stochastic training examples randomly chosen from \( S \) training examples. In the constraints, \( \epsilon > 0 \) is needed because \( \kappa_j^{(i)} = 0 \) and \( \kappa_j^{(i)} = 1 \) are not implementable for frequency response (Equation (2-4)) and group delay response (Equation (2-11)).

Note that stochastic gradient descent cannot be used to solve this optimization problem because of the constraints. A non-linear optimization method with linear inequality constraints is needed. The stochastic gradient projection method is a good choice because of its simplicity which results in faster computation [15, p. 49f].

### 5-1-5 Stochastic Gradient Projection Method

We aim to minimize the cost function stated in Equation (5-21) and (5-22) via the stochastic gradient projection, i.e.

\[
\begin{bmatrix}
\kappa_1 \\
\vdots \\
\kappa_R
\end{bmatrix}
\leftarrow
\begin{bmatrix}
\kappa_1 \\
\vdots \\
\kappa_R
\end{bmatrix}
- \lambda \mathbf{P}
\begin{bmatrix}
\frac{\partial C}{\partial \kappa_1} \\
\vdots \\
\frac{\partial C}{\partial \kappa_R}
\end{bmatrix},
\]

\[
\begin{bmatrix}
\phi_1 \\
\vdots \\
\phi_R
\end{bmatrix}
\leftarrow
\begin{bmatrix}
\phi_1 \\
\vdots \\
\phi_R
\end{bmatrix}
- \lambda \mathbf{P}
\begin{bmatrix}
\frac{\partial C}{\partial \phi_1} \\
\vdots \\
\frac{\partial C}{\partial \phi_R}
\end{bmatrix},
\]

where \( \lambda \) is a sufficiently small learning rate and \( R \) is number of ORRs. \( \mathbf{P} \) is the projection matrices that projects the negative gradient on the linear subspace that is tangent to the active constraints [15, p. 49] (see Figure 5-4). The term \( \frac{\partial C}{\partial \kappa_n} \) and \( \frac{\partial C}{\partial \phi_n} \) are the derivatives of the cost function \( C \) with respect to \( \kappa \) and \( \phi \) respectively, which are formulated by

\[
\frac{\partial C}{\partial \kappa_n} = \frac{\partial C}{\partial w_{i,j}} \frac{\partial w_{i,j}}{\partial \kappa_n},
\]

\[
\frac{\partial C}{\partial \phi_n} = \frac{\partial C}{\partial w_{i,j}} \frac{\partial w_{i,j}}{\partial \phi_n},
\]

where \( w_{i,j} \) specifies the \((i,j)\)-th element of the weight matrix. In the stochastic gradient projection method, we randomly choose several example sets from the whole training examples every iteration as mentioned in Equation 5-21 where we randomly choose \( U \) examples from a whole \( S \) training examples.
Note that the gradient projection method is actually similar to the gradient descent method except for the case of obtained optimum parameters being outside the feasible area. In that case, parameters obtained from the optimization algorithm will be projected onto the constraint boundary [15, p. 49f].

![Figure 5-4: Illustration of the gradient projection method [15, p. 49f].](image)

Figure 5-4 shows the illustration of the gradient projection method where the gradient $\nabla f(x_i)$ is used as a search direction which results in the design variable being outside the feasible area. The projection matrix $P$ will project that respective variable onto the constraint boundary.

### 5-1-6 The Backpropagation Algorithm and The Derivative of the Frequency Response with respect to Parameters

The optimization process using the stochastic gradient projection method needs the information of the gradient of the cost function with respect to parameters $\kappa_n$ and $\phi_n$. Many methods can be used to find the gradient, but the most efficient way is by using the backpropagation algorithm.

The backpropagation algorithm is used to find the derivative of the cost function with respect to the weight ($\frac{\partial C}{\partial w_{i,j}}$) in Equation (5-25) and (5-26) efficiently by applying the chain rule in a reverse order. The backpropagation is formulated in Section 4-2-1. Note that in this case, the weights refer to frequency responses $H_j$ mentioned in Equation (5-13). However, we have to carefully consider coupling weights, in which some weights represent the same frequency response of a certain ORR.

In order to understand the concept of coupling weights, recall the weight matrices of layer 2 and 3 of $4 \times 1$ OBFN presented in Equation (5-11) and (5-12):

$$W^{(2)} = \begin{bmatrix} w^{(2)}_{1,1} & w^{(2)}_{1,2} & w^{(2)}_{1,3} & w^{(2)}_{1,4} \\ w^{(2)}_{2,1} & w^{(2)}_{2,2} & w^{(2)}_{2,3} & w^{(2)}_{2,4} \\ w^{(2)}_{3,1} & w^{(2)}_{3,2} & w^{(2)}_{3,3} & w^{(2)}_{3,4} \\ w^{(2)}_{4,1} & w^{(2)}_{4,2} & w^{(2)}_{4,3} & w^{(2)}_{4,4} \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & H_2 & 0 \\ 0 & 0 & 0 & H_2 \end{bmatrix} \in \mathbb{C}^{4N \times 4N}, \quad (5-27)$$
where

\[
W^{(3)} = \begin{bmatrix}
w_{1,1}^{(3)} & w_{1,2}^{(3)} & w_{1,3}^{(3)} & \cdots & w_{1,N}^{(3)} \\
w_{2,1}^{(3)} & w_{2,2}^{(3)} & w_{2,3}^{(3)} & \cdots & w_{2,N}^{(3)} \\
w_{3,1}^{(3)} & w_{3,2}^{(3)} & w_{3,3}^{(3)} & \cdots & w_{3,N}^{(3)} \\
w_{4,1}^{(3)} & w_{4,2}^{(3)} & w_{4,3}^{(3)} & \cdots & w_{4,N}^{(3)} \\
\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & H_3 & 0 \\
0 & 0 & 0 & H_3 \\
\end{bmatrix} \in \mathbb{C}^{4N \times 4N}.
\]

and \( N \) specifies the number of frequencies.

We can observe in Equation (5-28) that we have \( W_{3,3}^{(3)} = W_{4,4} = H_3 \). Using Equation (5-13) and (5-29), we get the relation of the weight with frequency response of a certain ORR as follows:

\[
w_{3,3}^{(3)} = H_3
\]

\[
\begin{bmatrix}
w_{1,1}^{(3)} & \cdots & w_{1,N}^{(3)} \\
\vdots & \ddots & \vdots \\
w_{N,1}^{(3)} & \cdots & w_{N,N}^{(3)} \\
\end{bmatrix} = \begin{bmatrix}H_3(f_1) & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & H_3(f_N) \\
\end{bmatrix}.
\]

Note that the frequency response \( H_n(f_1), H_n(f_2), \ldots, H_n(f_N) \) share the same parameter \( \kappa_3 \) and \( \phi_3 \). This needs to be properly considered when updating \( \kappa_n \) and \( \phi_n \) via stochastic gradient projection. This means that Equation (5-25) and (5-26) should be modified such that

\[
\frac{\partial C}{\partial \kappa_n} = \sum_{w_{i,j}=H_n} \frac{\partial C}{\partial w_{i,j}} \frac{\partial w_{i,j}}{\partial \kappa_n},
\]

\[
\frac{\partial C}{\partial \phi_n} = \sum_{w_{i,j}=H_n} \frac{\partial C}{\partial w_{i,j}} \frac{\partial w_{i,j}}{\partial \phi_n}.
\]

The remaining terms from Equation (5-31) and (5-32) are the derivatives of weights with respect to parameters \( \kappa \) and \( \phi \). Recall the frequency response of an ORR defined by Equation (2-4):

\[
H(f) = \frac{\sqrt{1 - \kappa - re^{-2\pi j ft-j \phi}}}{1 - r \sqrt{1 - \kappa e^{-2\pi j ft-j \phi}}}
\]

Then, the partial derivative of \( H \) with respect to \( \kappa \) is

\[
\frac{\partial H(\kappa, \phi)}{\partial \kappa} = \left( \frac{1 - \sqrt{1 - \kappa e^{-i(2\pi ft+\phi)}}}{1 - r \sqrt{1 - \kappa e^{-i(2\pi ft+\phi)}}} \right)^2
\]

\[
- \left( \frac{\sqrt{1 - \kappa - re^{-i(2\pi ft+\phi)}}}{\sqrt{1 - \kappa e^{-i(2\pi ft+\phi)}}} \right)^2
\]

\[
\frac{1}{2r^2 \sqrt{1 - \kappa e^{-i(2\pi ft+\phi)}}} - 1
\]

\[
\frac{1}{2 \sqrt{1 - \kappa} \left( 1 - r \sqrt{1 - \kappa e^{-i(2\pi ft+\phi)}} \right)^2}.
\]
The partial derivative of $H$ with respect to $\phi$ is

$$
\frac{\partial H(\kappa, \phi)}{\partial \phi} = \left(1 - r\sqrt{1 - \kappa e^{-i(2\pi fT + \phi)}}\right) \frac{\left(i r e^{-i(2\pi fT + \phi)}\right)}{\left(1 - r\sqrt{1 - \kappa e^{-i(2\pi fT + \phi)}}\right)^2} - \left(\sqrt{1 - \kappa} - r e^{-i(2\pi fT + \phi)}\right) \frac{\left(i r e^{-i(2\pi fT + \phi)}\right)}{\left(1 - r\sqrt{1 - \kappa e^{-i(2\pi fT + \phi)}}\right)^2}
$$

$$= \left(\kappa - r e^{-i(2\pi fT + \phi)}\right) \frac{\left(i r e^{-i(2\pi fT + \phi)}\right)}{\left(1 - r\sqrt{1 - \kappa e^{-i(2\pi fT + \phi)}}\right)^2} - \left(i r(1 - \kappa) e^{-i(2\pi fT + \phi)} - i r^2\sqrt{1 - \kappa} e^{-2i(2\pi fT + \phi)}\right) \frac{\left(1 - r\sqrt{1 - \kappa e^{-i(2\pi fT + \phi)}}\right)^2}{\left(1 - r\sqrt{1 - \kappa e^{-i(2\pi fT + \phi)}}\right)^2}
$$

$$= \frac{i r\kappa e^{-i(2\pi fT + \phi)}}{\left(1 - r\sqrt{1 - \kappa e^{-i(2\pi fT + \phi)}}\right)^2}.
$$

(5-34)

For summary, the derivative of the cost function with respect to the weight ($\frac{\partial C}{\partial w_{i,j}}$) is computed via backpropagation. Note that the coupling weights need to be carefully considered. Then, the derivative of the weight with respect to parameters mentioned in Equation (5-31) and (5-32) is defined by

$$
\frac{\partial w_{i,j}}{\partial \kappa_n} = \frac{r^2 e^{-2i(2\pi fT + \phi_n)} - 1}{2\sqrt{1 - \kappa_n} \left(1 - r\sqrt{1 - \kappa_n e^{-i(2\pi fT + \phi_n)}}\right)^2},
$$

$$\frac{\partial w_{i,j}}{\partial \phi_n} = \frac{i r\kappa_n e^{-i(2\pi fT + \phi_n)}}{\left(1 - r\sqrt{1 - \kappa_n e^{-i(2\pi fT + \phi_n)}}\right)^2}.
$$

(5-35)

This completes the formula to find the gradient (derivative) of the cost function with respect to the parameters $\kappa$ and $\phi$, which enables us to implement the stochastic gradient projection method to solve the optimization problem.

### 5-2 Separation of Complex Numbers

Table 5-1 shows the tuning results using deep learning approach. We can see that the resulting parameters are complex. These complex parameters are not desirable because they cannot be applied in real system applications.

In order to solve this complex parameters problem, a different approach in designing neural network representations is proposed. The basic idea of this approach is that training examples of the neural network network are separated into real and imaginary numbers. Both of them will be treated as new real training examples. The same case is also applied to the weight matrices. This approach will most likely enables the optimization algorithm to obtain real optimum values of $\kappa$ and $\phi$. In the following subsection, the modification of neural networks based on this approach will be presented.
Table 5-1: Tuning $4 \times 1$ OBFNs results in complex number $\kappa$ and $\phi$

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>Result</th>
<th>$\phi$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\kappa_1$</td>
<td>0.7608 - 0.2193i</td>
<td>$\phi_1$</td>
<td>0.0712 - 0.0019i</td>
</tr>
<tr>
<td>$\kappa_2$</td>
<td>0.5329 + 0.3498i</td>
<td>$\phi_2$</td>
<td>0.3378 + 0.0142i</td>
</tr>
<tr>
<td>$\kappa_3$</td>
<td>0.8162 - 0.0039i</td>
<td>$\phi_3$</td>
<td>4.4379 + 1.9450i</td>
</tr>
<tr>
<td>$\kappa_4$</td>
<td>0.6539 - 0.2501i</td>
<td>$\phi_4$</td>
<td>0.0933 + 0.0117i</td>
</tr>
</tbody>
</table>

5-2-1 Separation of Complex Numbers in Training Examples

Consider the neural network structures in Figure 5-1 and 5-3 consisting of $M$ antenna elements. The input vector of the neural network stated in Equation (5-4) is modified into separate real and imaginary numbers as follows:

$$
\mathbf{x}^{(0)} = \left[ \begin{array}{c} \mathbf{x}_1^{(0)} \\ \mathbf{x}_2^{(0)} \\ \vdots \\ \mathbf{x}_M^{(0)} \end{array} \right] \in \mathbb{R}^{2MN} \text{ with } \mathbf{x}_i^{(0)} = \left[ \begin{array}{c} \text{Re}\{x_i^{(0)}(f_1)\} \\ \text{Im}\{x_i^{(0)}(f_1)\} \\ \vdots \\ \text{Re}\{x_i^{(0)}(f_N)\} \\ \text{Im}\{x_i^{(0)}(f_N)\} \end{array} \right] \in \mathbb{R}^{2N}, \quad (5-36)
$$

where $N$ specifies the number of frequencies.

Consider a lossless system, then the desired output vector of the network becomes

$$
\mathbf{d} = \left[ \begin{array}{c} \mathbf{x}_1^{(0)} \\ \mathbf{x}_2^{(0)} \\ \vdots \\ \mathbf{x}_1^{(0)} \end{array} \right] \in \mathbb{R}^{2MN} \text{ with } \mathbf{x}_i^{(0)} = \left[ \begin{array}{c} \text{Re}\{x_i^{(0)}(f_1)\} \\ \text{Im}\{x_i^{(0)}(f_1)\} \\ \vdots \\ \text{Re}\{x_i^{(0)}(f_N)\} \\ \text{Im}\{x_i^{(0)}(f_N)\} \end{array} \right] \in \mathbb{R}^{2N}. \quad (5-37)
$$

For a lossy system, the magnitude responses will be included into the calculation as follows:

$$
\mathbf{d} = \left[ \begin{array}{c} \mathbf{x}_1^{(0)} \\ |H|_{tot,2} \mathbf{x}_1^{(0)} \\ \vdots \\ |H|_{tot,M} \mathbf{x}_1^{(0)} \end{array} \right] \in \mathbb{R}^{2MN} \text{ with } \mathbf{x}_i^{(0)} = \left[ \begin{array}{c} \text{Re}\{x_i^{(0)}(f_1)\} \\ \text{Im}\{x_i^{(0)}(f_1)\} \\ \vdots \\ \text{Re}\{x_i^{(0)}(f_N)\} \\ \text{Im}\{x_i^{(0)}(f_N)\} \end{array} \right] \in \mathbb{R}^{2N}, \quad (5-38)
$$

where $|H|_{tot,m}$ specifies the total magnitude response of ORRs for the $m$-th path, which is formulated by $|H|_{tot,m} = |H|_{1,m} \times |H|_{2,m} \times \cdots \times |H|_{r,m}$ where $r$ being the number of ORRs in that respective path.
5-2-2 Complex Numbers Separation in Weight Matrices

The layers propagate the input using weight matrices $W(l) \in \mathbb{R}^{2MN \times 2MN}$ as follows:

$$y(l) = W(l)x^{(l-1)},$$

$$x(l) = F(y(l)).$$

(5-39)

(5-40)

where $l$ specifies the layer index and $F$ specifies the activation function which is defined by

$$F : \mathbb{R}^N \rightarrow \mathbb{R}^N, \quad F(s) = s.$$ 

(5-41)

The basic configuration of the weight matrices is still the same as mentioned in equation (5-10) to (5-20), except that $H_j$, which specifies the frequency response matrix of ORR number $j$, and are now defined by

$$H_j \in \mathbb{R}^{2N \times 2N} = \begin{bmatrix} \text{Re}(H_j(f_1)) & -\text{Im}(H_j(f_1)) & \cdots & 0 & 0 \\ \text{Im}(H_j(f_1)) & \text{Re}(H_j(f_1)) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \text{Re}(H_j(f_N)) & -\text{Im}(H_j(f_N)) \\ 0 & 0 & \cdots & \text{Im}(H_j(f_N)) & \text{Re}(H_j(f_N)) \end{bmatrix}.$$ 

(5-42)

5-2-3 The Effect of Complex Numbers Separation on Backpropagation

The separation of complex numbers idea treats the real and imaginary parts of an input vector as two different real inputs. The same thing goes for the desired output vector. The weight matrices, which are formed by frequency responses $H_j$, have a special structure as observed in Equation (5-42). One can observe that there are relations (or constraints) between some elements of the $H_j$. The (1,1)-element of the $H_j$ is equal to the (2,2)-element, while the (2,1)-element is equal to minus the (1,2)-element.

To derive the formula for backpropagation with constraints of real and imaginary separation, we first consider the simplest case of one layer, one ORR and one frequency as follows:

$$x^{(1)} = H^{(1)}x^{(0)} \quad \text{with} \quad x^{(i)} \in \mathbb{C}^1, \quad H^{(i)} \in \mathbb{C}^{1 \times 1}$$

(5-43)

where $H$ is the frequency response of the ORR.

We separate the real and imaginary part such that the equation becomes

$$\begin{bmatrix} \text{Re}\{x^{(1)}\} \\ \text{Im}\{x^{(1)}\} \end{bmatrix} = \begin{bmatrix} \text{Re}\{H^{(1)}\} & -\text{Im}\{H^{(1)}\} \\ \text{Im}\{H^{(1)}\} & \text{Re}\{H^{(1)}\} \end{bmatrix} \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} \text{Re}\{x^{(0)}\} \\ \text{Im}\{x^{(0)}\} \end{bmatrix}.$$ 

(5-44)

From Equation (5-44) we know that constraints of the weight matrix $W$ is defined by

$$w_{1,1} = w_{2,2}, \quad w_{2,1} = -w_{1,2}.$$ 

(5-45)
We aim to minimize the cost function of that simplest case:

\[ C(\kappa, \phi) = \frac{1}{2} \| d - x^{(1)} \|^2 \quad (5-46) \]

Backpropagation is used to efficiently find the derivative of the cost function in Equation (5-46) with respect to the weight as follows:

\[
\frac{\partial C}{\partial w_{1,i}} = \frac{\partial C}{\partial x^{(1)}_1} \frac{\partial x^{(1)}_1}{\partial w_{1,i}} + \frac{\partial C}{\partial x^{(1)}_2} \frac{\partial x^{(1)}_2}{\partial w_{1,i}} \quad (5-47)
\]

Therefore, we obtain

\[
\frac{\partial C}{\partial w_{1,1}} = \frac{\partial C}{\partial x^{(1)}_1} x^{(0)}_{1,1},
\]

\[
\frac{\partial C}{\partial w_{1,2}} = \frac{\partial C}{\partial x^{(1)}_2} x^{(0)}_{1,2},
\]

\[
\frac{\partial C}{\partial w_{2,1}} = \frac{\partial C}{\partial x^{(1)}_1} x^{(0)}_{2,1},
\]

\[
\frac{\partial C}{\partial w_{2,2}} = \frac{\partial C}{\partial x^{(1)}_2} x^{(0)}_{2,2}.
\]

Using the chain rule and the implementation of constraints stated in Equation (5-45), we can find the derivative of the cost function with respect to parameters \( \kappa \) and \( \phi \) as follows:

\[
\frac{\partial C}{\partial \kappa} = \frac{\partial C}{\partial w_{1,1}} \frac{\partial w_{1,1}}{\partial \kappa} + \frac{\partial C}{\partial w_{2,1}} \frac{\partial w_{2,1}}{\partial \kappa} + \frac{\partial C}{\partial w_{1,2}} \frac{\partial w_{1,2}}{\partial \kappa} + \frac{\partial C}{\partial w_{2,2}} \frac{\partial w_{2,2}}{\partial \kappa}
\]

\[
= \frac{\partial C}{\partial w_{1,1}} \frac{\partial w_{1,1}}{\partial \kappa} + \frac{\partial C}{\partial w_{2,1}} \frac{\partial w_{2,1}}{\partial \kappa} + \frac{\partial C}{\partial w_{1,2}} \frac{\partial w_{1,2}}{\partial \kappa} - \frac{\partial C}{\partial w_{2,2}} \frac{\partial w_{2,2}}{\partial \kappa} + \frac{\partial C}{\partial w_{1,1}} \frac{\partial w_{1,1}}{\partial \kappa}
\]

\[
= \left( \frac{\partial C}{\partial w_{1,1}} + \frac{\partial C}{\partial w_{2,2}} \right) \frac{\partial w_{1,1}}{\partial \kappa} + \left( \frac{\partial C}{\partial w_{2,1}} - \frac{\partial C}{\partial w_{1,2}} \right) \frac{\partial w_{2,1}}{\partial \kappa}
\]

\[
= \left( \frac{\partial C}{\partial x^{(1)}_1} x^{(0)}_{1,1} + \frac{\partial C}{\partial x^{(1)}_2} x^{(0)}_{1,2} \right) \frac{\partial \text{Re} \left\{ H^{(1)} \right\}}{\partial \kappa} + \left( \frac{\partial C}{\partial x^{(1)}_1} x^{(0)}_{2,1} - \frac{\partial C}{\partial x^{(1)}_2} x^{(0)}_{2,2} \right) \frac{\partial \text{Im} \left\{ H^{(1)} \right\}}{\partial \kappa},
\]\n
\[
\frac{\partial C}{\partial \phi} = \frac{\partial C}{\partial w_{1,1}} \frac{\partial w_{1,1}}{\partial \phi} + \frac{\partial C}{\partial w_{2,1}} \frac{\partial w_{2,1}}{\partial \phi} + \frac{\partial C}{\partial w_{1,2}} \frac{\partial w_{1,2}}{\partial \phi} + \frac{\partial C}{\partial w_{2,2}} \frac{\partial w_{2,2}}{\partial \phi}
\]

\[
= \frac{\partial C}{\partial w_{1,1}} \frac{\partial w_{1,1}}{\partial \phi} + \frac{\partial C}{\partial w_{2,1}} \frac{\partial w_{2,1}}{\partial \phi} + \frac{\partial C}{\partial w_{1,2}} \frac{\partial w_{1,2}}{\partial \phi} - \frac{\partial C}{\partial w_{2,2}} \frac{\partial w_{2,2}}{\partial \phi} + \frac{\partial C}{\partial w_{1,1}} \frac{\partial w_{1,1}}{\partial \phi}
\]

\[
= \left( \frac{\partial C}{\partial w_{1,1}} + \frac{\partial C}{\partial w_{2,2}} \right) \frac{\partial w_{1,1}}{\partial \phi} + \left( \frac{\partial C}{\partial w_{2,1}} - \frac{\partial C}{\partial w_{1,2}} \right) \frac{\partial w_{2,1}}{\partial \phi}
\]

\[
= \left( \frac{\partial C}{\partial x^{(1)}_1} x^{(0)}_{1,1} + \frac{\partial C}{\partial x^{(1)}_2} x^{(0)}_{1,2} \right) \frac{\partial \text{Re} \left\{ H^{(1)} \right\}}{\partial \phi} + \left( \frac{\partial C}{\partial x^{(1)}_1} x^{(0)}_{2,1} - \frac{\partial C}{\partial x^{(1)}_2} x^{(0)}_{2,2} \right) \frac{\partial \text{Im} \left\{ H^{(1)} \right\}}{\partial \phi},
\]\n
\[
(5-52)
\]

\[
(5-53)
\]
where $\frac{\partial \text{Re}\{H^{(1)}\}}{\partial \kappa}$ and $\frac{\partial \text{Im}\{H^{(1)}\}}{\partial \kappa}$ specify the real and imaginary part of the derivative of the frequency response with respect to $\kappa$ mentioned in Equation (5-33). The term $\frac{\partial \text{Re}\{H^{(1)}\}}{\partial \kappa}$ and $\frac{\partial \text{Im}\{H^{(1)}\}}{\partial \kappa}$ specify the same case for $\phi$.

From this simple case, we can generate similar concepts for more complicated neural networks with cascades of multiple ORRs and multiple frequencies. Consider the frequency response presented in Equation (5-42). It can be considered as a matrix that consists of some simple weights mentioned in Equation (5-44). Therefore, we can implement similar concept discussed in this section to more complicated frequency response in Equation (5-42).

### 5-3 Numerical Results

#### 5-3-1 Simulation Setup

The nominal parameters of the OBFN setup simulated in this project are similar to the ones presented in [2]. A round-trip time ($T$) of 0.08 ns was used. The wavelength ($\lambda$) is 1550 nm. The frequency center ($f_c$) is 107.5249817 THz. The bandwidth of interest is 2 GHz. The complex numbers separation is implemented to ensure the resulting $\kappa$ and $\phi$ are real numbers. Therefore, training examples and weight matrices are formulated as mentioned in Section 5-2. The ORRs are considered to be lossy with power loss equal to 0.95.

In the simulation analysis, different errors will be used. The error which is stated in the cost function in Equation (5-21) (see subsection 5-1-4) is called the test error. This test error refers to the difference between the desired output signal and the actual output signal of the neural network for given training examples.

Another term of error is squared group delay error which is defined by

$$E_{SGD} = \frac{1}{2} \sum_{m=1}^{M} \sum_{n=1}^{F} \|D_{\text{des},m}(f_n) - D_{\text{act},m}(f_n)\|^2,$$

(5-54)

where $M$ specifies the number of AE. $D_{\text{des},m}$ specifies the desired delay response of the $m$-th path, which is known and given as a reference. $D_{\text{act},m}$ specifies the actual delay response of the $m$-th path, which is obtained by feeding $\kappa$ and $\phi$ into the group delay response formula presented in Equation (2-11). The bandwidth is discretized into $F$ frequencies. Note that this $F$ frequencies is different to $N$ frequency for tuning OBFN. $F$ should be chosen sufficiently big, i.e., $F = 100$, such that the $E_{SGD}$ is accurate.

It is often necessary to compute normalized squared group delay error which is defined by

$$E_{NSGD} = \frac{1}{2} \sum_{m=1}^{M} \sum_{n=1}^{F} \frac{\|D_{\text{des},m}(f_n) - D_{\text{act},m}(f_n)\|^2}{\|D_{\text{des},m}(f_n)\|^2},$$

(5-55)

since it gives the comparison how big the error is compared to the desired group delay response.
5-3-2 Simulation Result of a $4 \times 1$ OBFN for a Certain Desired Delay

Figure 5-5 shows the simulation result of a $4 \times 1$ OBFN with desired delay $[0 0.1 0.2 0.3]$ ns. The optimization algorithm used is the stochastic gradient projection, in which we randomly choose 20 out of 20000 training examples every iteration. The frequency band of interest is discretized into 10 frequencies. Initial guesses $\kappa_0$ and $\phi_0$ are chosen to be close to a good solution mentioned in [2], which are $\kappa_0 = [0.9 \ 0.9 \ 0.9 \ 0.9]$ and $\phi_0 = [0 \ 0.4 \ (2\pi - 0.4) \ 0]$.

![Figure 5-5: Simulation results of a $4 \times 1$ OBFN using stochastic gradient projection. (a) group delay response. (b) test error. Nominal parameters: $N_{freq} = 10$, $N_{examples} = 20000$, $N_{stochastic} = 20$, delay $= [0 0.1 0.2 0.3]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, iteration $= 1000$.](image)

Table 5-2 shows the resulting $\kappa^*$, $\phi^*$, and the initial and final normalized squared group delay error from the simulation in Figure 5-5. As we can observe, all $\kappa^*$ and $\phi^*$ are real numbers and inside the feasible area. These parameters are similar to the result found in [2].

<table>
<thead>
<tr>
<th>ORR</th>
<th>$\kappa_0$</th>
<th>$\phi_0$</th>
<th>Initial $E_{NSGD}$</th>
<th>$\kappa^*$</th>
<th>$\phi^*$</th>
<th>Final $E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>0</td>
<td>0.2493</td>
<td>0.9859</td>
<td>0.0018</td>
<td>1.9079E-04</td>
</tr>
<tr>
<td>2</td>
<td>0.9</td>
<td>5.8832 ($2\pi - 0.4$)</td>
<td></td>
<td>0.9766</td>
<td>5.9191</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.9</td>
<td>0.4</td>
<td></td>
<td>0.9726</td>
<td>0.3713</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.9</td>
<td>0</td>
<td></td>
<td>0.9865</td>
<td>0.0027</td>
<td></td>
</tr>
</tbody>
</table>
5-3-3 Simulation Result of a $4 \times 1$ OBFN for Several Desired Delays

It is now interesting to simulate some different desired delays. Let the desired delays be $d[0, 0.1, 0.2, 0.3]$ ns where $d$ is a positive real number. Table 5-3 shows the normalized squared group delay error of the simulation of $4 \times 1$ OBFNs where $d$ is varied from 1 to 3. More detailed results including all $\kappa^*$ and $\phi^*$ are presented in Table A-1 in the Appendix A.

Table 5-3: Normalized squared group delay error for several delays, with $\kappa_0 = [0.9, 0.9, 0.9, 0.9]$ and $\phi_0 = [0, (2\pi - 0.4), 0.4, 0]$ using stochastic gradient projection. Nominal parameters: $\text{Nfreq} = 10$, $\text{Nexamples} = 20000$, $\text{Nstochastic} = 20$, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000.

<table>
<thead>
<tr>
<th>$d$</th>
<th>$E_{NSGD}$</th>
<th>$d$</th>
<th>$E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2.0235 \times 10^{-4}$</td>
<td>1.1</td>
<td>$4.5599 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.1</td>
<td>$2.1$</td>
<td>1.2</td>
<td>$8.2673 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.2</td>
<td>$2.2$</td>
<td>1.3</td>
<td>$1.3115 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.3</td>
<td>$2.3$</td>
<td>1.4</td>
<td>$1.9003 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.4</td>
<td>$2.4$</td>
<td>1.5</td>
<td>$2.5756 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.5</td>
<td>$2.5$</td>
<td>1.6</td>
<td>$3.3153 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.6</td>
<td>$2.6$</td>
<td>1.7</td>
<td>$4.0978 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.7</td>
<td>$2.7$</td>
<td>1.8</td>
<td>$4.9126 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.8</td>
<td>$2.8$</td>
<td>1.9</td>
<td>$5.7753 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.9</td>
<td>$2.9$</td>
<td>2</td>
<td>$6.7455 \times 10^{-4}$</td>
</tr>
<tr>
<td>2</td>
<td>$3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As we can see from Table 5-3 the deep learning works well for any desired delays from $d = 1$ up to $d = 3$. If we observe normalized squared group delay errors in Table 5-3, they seem reasonable compared to the group delay plot shown in Figure 5-5a. Another interesting thing is that the error increases as the desired delay becomes bigger. This is expected because of the tradeoff between desired delays and the ripples as mentioned in Section 2-2-3. Resulting optimum parameters presented in Table A-1 are as expected. The optimum value $\kappa^*$ becomes smaller as the delay becomes bigger, which is consistent with the manual tuning principles mentioned in [2, p.11f].

5-3-4 Simulation Result of a $4 \times 1$ OBFN for Some Different Initializations

We simulated different initializations to see how significant the initialization is to the optimization result. Table 5-4 shows the normalized squared group delay error for different $\kappa_0$. The simulation uses the same $\phi_0 = [0, (2\pi - 0.4), 0.4, 0]$. More detailed results are presented in Table A-2 in the Appendix A.

From Table 5-4 we can see that changing $\kappa_0$ can significantly increase the group delay error if we choose it very far from the optimum values. Taking the same value of $\kappa_0$ is preferable when using random initialization since it will make the computation more effective. This is due to the fact that the desired group delay response is flat. Moreover, choosing the same $\kappa_0$ for all ORRs will reduce the required number of random initial parameters. Randomly choosing one value for all $\kappa_0$ is computationally cheaper than randomly choosing $n$ values, especially when $n$ is a very large number.
Table 5-4: Normalized squared group delay error for different $\kappa_0$ but the same $\phi_0 = [0, (2\pi - 0.4), 0.4, 0]$ using stochastic gradient projection. Nominal parameters: $N_{\text{freq}} = 10$, $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, delay = $[0.1 0.2 0.3]$ ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000.

<table>
<thead>
<tr>
<th>$\kappa_0$</th>
<th>$E_{\text{NSGD}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[0.9, 0.9, 0.9, 0.9]$</td>
<td>$2.0235 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0.7, 0.7, 0.7, 0.7]$</td>
<td>$1.9079 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0.5, 0.5, 0.5, 0.5]$</td>
<td>$1.8535 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0.3, 0.3, 0.3, 0.3]$</td>
<td>$2.2898$</td>
</tr>
<tr>
<td>$[0.9, 0.8, 0.7, 0.6]$</td>
<td>$2.2239 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

Now we try to simulate different $\phi_0$ using the same $\kappa_0 = [0.9, 0.9, 0.9, 0.9]$. The purpose is similar, we want to know how significant $\phi_0$ is to the result, which later can be used as a rule of thumb for determining good initial parameters. Normalized squared group delay errors for some $\phi_0$ are shown in Table 5-5, and more detailed results are in Table A-3 in the Appendix.

Table 5-5: Normalized squared group delay error for some $\phi_0$ but the same $\kappa_0 = [0.9, 0.9, 0.9, 0.9]$ using stochastic gradient projection. Nominal parameters: $N_{\text{freq}} = 10$, $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, delay = $[0.1 0.2 0.3]$ ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000.

<table>
<thead>
<tr>
<th>$\phi_0$</th>
<th>$E_{\text{NSGD}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[0 (2\pi - 0.4) 0.4 0]$</td>
<td>$2.0235 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 0.6) 0.6 0]$</td>
<td>$1.6706 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 0.8) 0.8 0]$</td>
<td>$1.1615 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 1.0) 1.0 0]$</td>
<td>$7.3265 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 1.2) 1.2 0]$</td>
<td>$9.3128 \times 10^{-3}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 0.4) 0.4 0]$</td>
<td>$1.5972 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 0.6) 0.6 0]$</td>
<td>$1.7089 \times 10^{-4}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 0.8) 0.8 0]$</td>
<td>$2.2239 \times 10^{-3}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 1.0) 1.0 0]$</td>
<td>$1.5347 \times 10^{-2}$</td>
</tr>
<tr>
<td>$[0 (2\pi - 1.2) 1.2 0]$</td>
<td>$1.5210 \times 10^{-1}$</td>
</tr>
</tbody>
</table>

Table 5-5 indicates that we need to carefully choose $\phi_0$ to obtain a good result. First, we consider the simplest case. Figure 5-6 shows the illustration of a binary tree $4 \times 1$ OBFN with 2 stages. The best choice of $\phi_0$ is different for every stage. For stage 1, with one ORR, the best $\phi_0$ is

$$\phi_0^{\text{stage 1}} = \phi_c$$

(5-56)

where $\phi_c$ denotes the so-called phase center. It is the phase shift from which the group delay plot is exactly located in the center of frequency of interest. Nominal parameters of OBFN setups considered in this thesis correspond to a phase center $\phi_c = 0$ (or equally $\phi_c = 2\pi$).

Figure 5-6: Illustration of binary tree $4 \times 1$ OBFN.
are defined as

\[ \phi_{0}^{\text{stage 2}} = \phi_c - \phi_a \]  
\[ \phi_{0}^{\text{stage 2}} = \phi_c + \phi_a \]

where \( \phi_a \) specifies any random phase shift from a feasible set of \( \phi \).

Note that for larger OBFN setups, the rule of thumb for determining good values for \( \phi_0 \) is similar. For stage \( n \) with \( r \) ORRs, we need to specify \( r/2 \) pairs of ORRs. The best initial guess for those \( r/2 \) pairs of ORRs is similar to Equations (5-57) and (5-58).

A more detailed explanation on how to obtain good initial values for \( \kappa \) and \( \phi \) will be presented in the next chapter and in Chapter 7. The implementation of breakthroughs in deep learning to avoid overfitting and decrease the computational time will be presented as well in the next chapter.
As has been presented in Chapter 4, neural networks with a huge numbers of weights and multiple layers are very powerful machine learning systems. With that potential, deep neural networks can represent large-scale OBFNs and be used to tune them. However, tuning large-scale OBFNs is computationally expensive. Moreover, deep learning can get trapped in local optima and may have serious problem with overfitting. Some breakthroughs in deep learning have been proposed in recent years to overcome these challenges. The momentum method, unsupervised pre-training methods, and dropout regularization are some breakthroughs which can be used to help deep learning solve complex non-linear problems as was presented in Chapter 4.

The first section of this chapter will discuss the importance of the momentum. Then, the implementation of auto-encoders and dropout regularization will be discussed including the problems that arise because of the special structure of OBFNs. Two methods of initialization: random initialization and group delay error approach will be discussed afterwards. The frequency overfitting problem including the approach to address this problem will close this chapter. Note that in this chapter, all simulations will be based on the numerical setup mentioned in Section 5-3-1.

6-1 Momentum

The momentum method introduced by Polyak [57] is a technique for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations [58]. The importance of momentum and its effect for tuning OBFN systems will be discussed in the following subsections.
6-1-1 The Importance of Momentum

Given the cost function $C(x)$ to be minimized, the momentum update is given by

$$v_{t+1} = \mu v_t + \epsilon \nabla C(x_t),$$

(6-1)

$$x_{t+1} = x_t + v_{t+1},$$

(6-2)

where $\epsilon$ specifies the learning rate, $\mu$ specifies the momentum coefficient valued between 0 and 1, and $\nabla C(x_t)$ is the gradient of the cost function $C(x)$ at $x_t$.

From Equation (6-1) to (6-2), we see that the momentum computes the new velocity ($v_{t+1}$) by applying a gradient-based correction to the previous velocity vector ($v_t$), and then adds the velocity to $x_t$ [58].

Polyak [57] showed that momentum improves local convergence rates. In his publication, he showed that momentum can considerably accelerate convergence to a local minimum, requiring $\sqrt{R}$ times fewer iterations than steepest descent to reach the same level of accuracy, where $R$ is the condition number of the curvature at minimum and the momentum coefficient $\mu$ is set to

$$\mu = \frac{\sqrt{R} - 1}{\sqrt{R} + 1}.$$  

(6-3)

6-1-2 The Effect of Momentum for Tuning OBFNs

Figure 6-1 shows the comparison of test errors of the implementation for different momentum coefficients. The momentum coefficient ($\mu$) has a value between 0 and 1. Figure 6-1a and Figure 6-1d show the plot of the test error of simulation of the case without momentum ($\mu = 0$) and full momentum ($\mu = 1$) respectively.

Numerical results shown in Figure 6-1 are as expected. Momentum can accelerate gradient descent, which will result in faster convergence. We can observe it from Figure 6-1a to Figure 6-1c. However, wrongly tuned momentum can be a disadvantage, as we can see in Figure 6-1d. In Figure 6-1d, the momentum coefficient is set too large which makes the test error oscillate wildly. As a consequence the optimization can not achieve the expected group delay response, as we can observe in Figure 6-2.

Table 6-1 shows the comparison of normalized squared group delay errors for different momentum coefficients. We can observe that bigger momentum coefficient results in smaller normalized squared group delay error. However, it should be noted that excessively large momentum coefficient will make the error become very big, which is not desirable, as shown in the case of $\mu = 1$ in Table 6-1.

Beside momentum, a good initialization is also crucial for training deep neural networks. Methods to find good initializations, called pre-training method, have been presented in Chapter 4. Recent studies have found that auto-encoders, which have been briefly explained in Section 4-4, work well to pre-train deep neural networks [38, 43]. On the next section, the implementation of auto-encoders to pre-train deep neural network representation of OBFN systems will be presented.
Figure 6-1: The comparison of the test error result of the $4 \times 1$ OBFN simulation using different momentum coefficients ($m$): (a) $m = 0$, (b) $m = 0.4$, (c) $m = 0.8$, (d) $m = 1.0$. Nominal parameters: $\kappa_0 = [0.6, 0.6, 0.6, 0.6]$, $\phi_0 = [0, 2\pi - 0.4, 0.4, 0]$, $N_{freq} = 20$, $N_{examples} = 20000$, $N_{stochastic} = 20$, delay = [0 0.2 0.4 0.6] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 500.

Figure 6-2: The comparison of the delay response of the $4 \times 1$ OBFN simulation using different momentum coefficients ($m$): (a) $m = 0.7$, (b) $m = 1.0$. Nominal parameters: $\kappa_0 = [0.6, 0.6, 0.6, 0.6]$, $\phi_0 = [0, 2\pi - 0.4, 0.4, 0]$, $N_{freq} = 20$, $N_{examples} = 20000$, $N_{stochastic} = 20$, delay = [0 0.2 0.4 0.6] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 500.
The Implementation of Some Breakthroughs in Deep Learning

6-2 Auto-encoder

Auto-encoders are artificial neural networks whose function is to reconstruct their inputs on the output layer. This section will cover the implementation of auto-encoders to pre-train OBFN systems and the problem which comes up because of the special structure of OBFNs.

6-2-1 Pre-training OBFN Systems using Auto-encoder

To understand the implementation of auto-encoders, let us consider the simplest case of a 4 × 1 OBFN. Larger OBFN setups will have similar concept except for the fact that a different weight configuration should be considered. Consider a 4 × 1 binary tree OBFN, which is represented by a neural network consisting of \( 2 \cdot 4 \cdot N \) neurons and \( L \) layers, where \( N \) specifies the number of frequencies. Note that the number of neurons is multiplied by 2 because of complex number separation. Let the input and layers propagation be defined as mentioned in Section 5-2.

Recall the neural network representation of the OBFN systems, e.g., as depicted in Figure 5-1 for 4 × 1 OBFNs. For Figure 5-1b, the weight matrices are defined as mentioned in Section 5-1-3. Note that the separation of complex numbers is implemented, so the frequency response matrix for particular ORR is defined in Equation (5-42).

The procedure of pre-training using auto-encoders has been explained in Section 4-4. The illustration of how we pre-train neural network representation of 4 × 1 OBFNs by training auto-encoders layer-by-layer is shown in Figure 6-3.

We can observe from Figure 6-3 that there are two types of weights: \( W^{(k+1)} \) which will be used as initial weights and artificial weights \( \tilde{W}^{(k+1)} \). Artificial weights \( \tilde{W}^{(k+1)} \) will be discarded after pre-training. We will consider both with and without tight-weight case explained in Section 4-4. The resulting \( \kappa \) and \( \phi \) from pre-training using auto-encoders will be used as initial parameters.

6-2-2 Identity Network Problem

As has been explained in Section 4-4, there is a concern about auto-encoders’ failure to learn useful representation when the number of neurons is not strictly decreasing from one layer to
the next. The network could just be identity and still perfectly reconstruct the input. This is not desirable because we can not get any initial parameters from identity networks. Even though Bengio et al. [38] found that networks with non-decreasing layer size still work well, there are some conditions which have high probability of causing identity networks, such as the structure of the weights and the linear activation function. Those two conditions exist in the OBFN tuning problems.

We can observe from Figure 6-3 that the gray line actually represents zero weights, which can be omitted (see Figure 6-4). From Figure 6-4 we can see that neurons are connected to neurons of the same path. It means that neurons from other paths do not influence the process of reconstructing input of certain path. This condition increases the possibility of networks being identity.

Tables 6-2 and 6-3 show results of pre-training the neural network representation of $4 \times 1$ OBFNs using auto-encoders with and without tight-weight respectively. As we can see from Tables 6-2 and 6-3, auto-encoders cannot produce the desired initial guesses for $\kappa$ and $\phi$ as mentioned in [2]. All $\kappa$ obtained from the auto-encoder were located in the lower limit constraint, which indicate that auto-encoders failed to extract good initial parameters.

One possible way to ensure non-identity networks is by using non-linear activation function.
Table 6-2: Pre-training results of $4 \times 1$ OB FnS using auto-encoders with tight-weight constraint. Nominal parameters: $N_{freq} = 10$, $N_{examples} = 20000$, $N_{stochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $1 \times 10^{-5}$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>Autoencoder result</th>
<th>$\phi$</th>
<th>Autoencoder result</th>
<th>$H$ at $f_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\kappa_1$</td>
<td>0.01</td>
<td>$\phi_1$</td>
<td>0</td>
<td>0.8261 - 0.0284i</td>
</tr>
<tr>
<td>$\kappa_2$</td>
<td>0.01</td>
<td>$\phi_2$</td>
<td>5.6632</td>
<td>0.9986 - 0.0153i</td>
</tr>
<tr>
<td>$\kappa_3$</td>
<td>0.01</td>
<td>$\phi_3$</td>
<td>1.6118</td>
<td>0.9997 + 0.0047i</td>
</tr>
<tr>
<td>$\kappa_4$</td>
<td>0.01</td>
<td>$\phi_4$</td>
<td>0</td>
<td>0.8261 - 0.0284i</td>
</tr>
</tbody>
</table>

Table 6-3: Pre-training results of $4 \times 1$ OB FnS using auto-encoders without tight-weight constraint. Nominal parameters: $N_{freq} = 10$, $N_{examples} = 20000$, $N_{stochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $1 \times 10^{-5}$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>Autoencoder result</th>
<th>$\phi$</th>
<th>Autoencoder result</th>
<th>$H$ at $f_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\kappa_1$</td>
<td>0.01</td>
<td>$\phi_1$</td>
<td>0</td>
<td>0.8261 - 0.0284i</td>
</tr>
<tr>
<td>$\kappa_2$</td>
<td>0.01</td>
<td>$\phi_2$</td>
<td>5.1983</td>
<td>0.9995 - 0.0082i</td>
</tr>
<tr>
<td>$\kappa_3$</td>
<td>0.01</td>
<td>$\phi_3$</td>
<td>1.8102</td>
<td>0.9998 + 0.0040i</td>
</tr>
<tr>
<td>$\kappa_4$</td>
<td>0.01</td>
<td>$\phi_4$</td>
<td>0</td>
<td>0.8261 - 0.0284i</td>
</tr>
</tbody>
</table>

Table 6-4: Pre-training results of $4 \times 1$ OB FnS using denoising auto-encoders. Nominal parameters: $N_{freq} = 10$, $N_{examples} = 20000$, $N_{stochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $1 \times 10^{-5}$.

<table>
<thead>
<tr>
<th>SNR</th>
<th>$\kappa$</th>
<th>Autoencoder result</th>
<th>$\phi$</th>
<th>Autoencoder result</th>
<th>$H$ at $f_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>$\kappa_1$</td>
<td>0.01</td>
<td>$\phi_1$</td>
<td>0.0922</td>
<td>0.8261 - 0.0284i</td>
</tr>
<tr>
<td></td>
<td>$\kappa_2$</td>
<td>0.01</td>
<td>$\phi_2$</td>
<td>5.5720</td>
<td>0.9985 - 0.0153i</td>
</tr>
<tr>
<td></td>
<td>$\kappa_3$</td>
<td>0.2872</td>
<td>$\phi_3$</td>
<td>0.7116</td>
<td>0.9035 + 0.3567i</td>
</tr>
<tr>
<td></td>
<td>$\kappa_4$</td>
<td>0.9950</td>
<td>$\phi_4$</td>
<td>0.0554</td>
<td>-0.9416 + 0.0447i</td>
</tr>
<tr>
<td>0.05</td>
<td>$\kappa_1$</td>
<td>0.0310</td>
<td>$\phi_1$</td>
<td>0.0922</td>
<td>0.8142 + 0.2296i</td>
</tr>
<tr>
<td></td>
<td>$\kappa_2$</td>
<td>0.0143</td>
<td>$\phi_2$</td>
<td>5.6721</td>
<td>0.9978 - 0.0222i</td>
</tr>
<tr>
<td></td>
<td>$\kappa_3$</td>
<td>0.4915</td>
<td>$\phi_3$</td>
<td>0.6793</td>
<td>0.5881 + 0.7304i</td>
</tr>
<tr>
<td></td>
<td>$\kappa_4$</td>
<td>0.9950</td>
<td>$\phi_4$</td>
<td>0.0755</td>
<td>-0.9399 + 0.0720i</td>
</tr>
</tbody>
</table>

However, the activation function used in this project is linear, so it cannot be implemented. Another way is to make sure the parameters do not hit the boundaries by using denoising auto-encoders, where we introduce noise to the training data. However, we found out that we need a very low signal to noise ratio (SNR) to make sure the parameters not hitting the boundary as presented in Table 6-4. This is useless since the signal will be badly corrupted by the noise. Pre-training using badly corrupted signal cannot extract good initial parameters as we can see from auto-encoders results presented in Table 6-4 which are very different compared to good initial parameters in [2]. The group delay response using that initial parameters is also not reasonable, as we can see in Figure 6-5.

Table 6-4: Pre-training results of $4 \times 1$ OB FnS using denoising auto-encoders. Nominal parameters: $N_{freq} = 10$, $N_{examples} = 20000$, $N_{stochastic} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $1 \times 10^{-5}$.

Pre-training using auto-encoder fails to obtain good initial parameters for tuning OB Fn systems. Therefore, another breakthrough needs to be implemented to increase the performance of the deep learning. The dropout regularization has been proven to increase the performance of the deep learning by preventing overfitting [51, 14]. In the next section, the implementation of dropout regularization will be covered.
6-3 Dropout Regularization

Deep neural networks with multiple non-linear hidden layers are prone to overfitting [51]. Dropout regularization is a technique that prevents overfitting. A brief explanation about dropout has been presented in Section 4-5. This section will talk about how to implement the concept of dropout regularization presented in Section 4-5 for tuning OBFNs.

6-3-1 Dropping Procedure of OBFN Systems

Consider the case of an $M \times 1$ OBFN. The neural network representation has $L$ layers and $2MN$ neurons per layer, where $N$ specifies the number of frequencies. The input vector to the neural network is defined as

$$
\mathbf{x}^{(0)} = \begin{bmatrix}
\text{Re} \left\{ x_1^{(0)} \right\} \\
\text{Im} \left\{ x_1^{(0)} \right\} \\
\vdots \\
\text{Re} \left\{ x_{MN}^{(0)} \right\} \\
\text{Im} \left\{ x_{MN}^{(0)} \right\}
\end{bmatrix} \in \mathbb{R}^{2MN}.
$$

The layers propagate the input as has been explained in Chapter 5. The dropping procedure is similar to the one explained in the example in Section 4-5. The dropping procedure is illustrated in Figure 6-6. If we observe the dropout procedure depicted in Figure 6-6, we can see that there is too much information loss when applying dropout to the neural network representation of the OBFN system.

6-3-2 Dropout Regularization Results in Too Much Information Loss

In order to get useful information on the output layer, the input vector should be propagated through the entire network. If too many parts of the input vector fail to be propagated,
some information regarding how the network behaves given that particular input will be missing. This kind of problem occurs when implementing dropout regularization into the neural network representation of OBFN systems. Figure 6-6 shows the illustration on how the information loss occurs when applying dropout in OBFN systems. When some neurons are dropped some signals cannot be propagated to the next layer anymore, which results in too much information loss (see Figure 6-6b).

Figure 6-7: The test error of dropout regularization implementation in a $4 \times 1$ OBFN with $p = 0.5$ for all layers except input layer whose $p = 0.9$. Nominal parameters: $N_{freq} = 10$, $N_{examples} = 20000$, $N_{stochastic} = 20$, delay $= [0.0 0.1 0.2 0.3]$ ns, $f_c = 107.52$ THz, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.9$, iteration $= 1000$. 
Too much information loss will make deep learning fail to minimize the difference between the actual output and the desired output of the network. Some parts of the actual output vector will just be 0 regardless of the value of $\kappa$ and $\phi$. Figure 6-7 shows the test error result of the implementation of dropout regularization in a $4 \times 1$ OBFN. As we can see, the test error is very big and looks like noise.

Both auto-encoder and dropout regularization do not work well when being implemented for tuning OBFN systems because of the special structure of the OBFNs. We can try to look for other solution, for instance, multi-start optimization in which we restart the optimization several times and choose the best result. Note that the multi-start is a general concept which can be implemented to any optimization problem, it does not belong to the breakthrough in the field of the deep learning.

### 6-4 Multi-start Optimization

Many techniques have been suggested to obtain a good optimum parameters from a complex non-linear optimization problem. One possible way is to re-start the optimization from a new initial guess once a region has been extensively explored [59, p. 355]. The multi-start optimization is a way to find the global optimum by computing different local optima from several random starting points in the feasible set using e.g., a uniform distribution [15, p. 76]. From the set of returned solutions, the one that yields the best value for the objective function is selected.

#### 6-4-1 Multi-start Algorithm

Multi-start methods have two phases: the first one in which the candidate initial parameters are generated and the second one in which the initial parameters are typically improved [59, p. 356]. Algorithm 2 shows a pseudo-code of the multi-start procedure applied to tune OBFN systems.

**Algorithm 2** Multi-start Optimization

```
Initialization:
Initialize $i = 1$
repeat
  Generate candidate initial parameters:
  Choose candidate initial parameters $\kappa_i$ and $\phi_i$ randomly from a feasible set of $\kappa$ and $\phi$
  Optimization:
  Apply gradient projection method to find the local optimum $\kappa_i^*$ and $\phi_i^*$
  Update the best optimum value:
  If $\kappa_i^*$ and $\phi_i^*$ improve the optimum value
  Update the best $\kappa_i^*$ and $\phi_i^*$
  Increment iteration:
  $i = i + 1$
until (maximum number of iterations > than specified) OR (stopping condition is satisfied)
```

Although the multi-start method is widely used in practice, it is not always efficient. Many heuristic algorithms have been proposed to address this problem. Some of them are based on frameworks that can be used directly to design solving methods for any problem, such as tabu search [60], scatter search [61], adaptive multi-start [62], and greedy randomized adaptive search procedures (GRASP) [63]. Some of them are based on mild assumptions.
about the objective function, such as Monte Carlo random re-start combined with local search procedures [64], and some others are based on physical insight or on additional information about the problem [15].

6-4-2 Choosing Candidate Initial Parameters Based on OBFNs Physical Insight

The main problem of the multi-start method is inefficiency. Choosing candidate initial parameters randomly from a feasible set often fails to obtain a good solution, especially for optimization problems with many design variables. Tuning large OBFN setups which consist of many ORRs is one of the examples on how plain vanilla multi-start method with random initializations fails to work.

One important thing to consider is how to efficiently choose a good candidate initial parameters for each run of multi-start optimization. As has been explained before, there have been many heuristic algorithms proposed to address this problem. One of them is based on physical insight of the system. Some heuristic algorithms based on physical insight or on additional information about the system are the simplest case, but they are problem-dependent. Moreover, the ideas and strategies implemented are difficult to apply to different problems [59, p. 356]. In this section, a method of choosing candidate initial parameters based on the OBFNs’ physical system will be derived. In order to be applied on tuning OBFN, this method has to work well for any OBFN setup.

Figure 6-8: Illustration of the binary tree $2^β \times 1$ OBFN. All $2^{n-1}$ ORRs in each stage form several mirror pairs of ORRs. The red dotted-line represents the mirror line.

Consider a binary tree $2^β \times 1$ OBFN depicted in Figure 6-8, which consists of several stages; from stage 1 up to stage $β$. Each stage $n$ has two types of paths, one without any ORR and one with $2^{n-1}$ ORRs. All $2^{n-1}$ ORRs in every stage form several pairs of ORRs, except for the stage 1 (see Figure 6-8). The basic idea is to choose the starting phase shift for a certain ORR $ϕ_{0r}$ and its counterpart $ϕ_{0r}$ such that

$$ϕ_{0r} = ϕ_c + ϕ_{ar},$$  \hfill (6-5)
\[ \hat{\phi}_{ir} = \phi_c - \phi_{ar}, \quad (6-6) \]

where \( \phi_{ar} \) denotes randomly chosen phase shift from a feasible set of \( \phi \), and \( \phi_c \) specifies the phase center. Equation (6-5) and (6-6) indicate that the choice of initial \( \phi \) is mirror to each other with \( \phi_c \) is set as the mirror line (see Figure 6-8). Figure 6-9 shows the illustration of phase center \( \phi_c \) and the choice of phase shift for pair of ORRs. By doing so, we can significantly reduce the number of random initialization of \( \phi_i \).

![Figure 6-9: Illustration of how to choose starting solution \( \phi \) of ORR pairs.](image)

Note that initial \( \kappa_0 \) can be set to be the same for all ORRs. As has been explained in Section 5-3-4, taking the same value of \( \kappa_0 \) is preferable when using random initialization. This is done for the sake of faster computation, since choosing the same value of \( \kappa_0 \) reduces the number of random initializations. This will make the initialization computationally cheaper. The value of \( \kappa_0 \) can be chosen from a set of feasible value of \( \kappa \), e.g., \((0.09, 0.19, \cdots, 0.99)\).

However, for very-large-scale OBFNs, multi-start optimization is not efficient even when using the OBFNs physical insight to choose candidate initial parameters, which will be presented in subsection 6-5-1. This is caused by the number of neurons in larger networks increases exponentially as the number of paths increases. Training a very large network is computationally expensive. Therefore, we need another approach to initialize the parameters in very large networks. The group delay error approach explained in the next section is one of the potential methods for initialization.

### 6-5 Initialization using Group Delay Error Approach

The main idea of this approach is to pre-test candidate initial guesses with a computationally cheap method, e.g., computing squared group delay error from given \( \kappa \) and \( \phi \). The formula for squared group delay error is defined in Equation (5-54). The advantage of using group delay error approach as initialization is that the computationally-expensive gradient projection is performed only once instead of every initial guess.

Normalized squared error of the group delay, which is defined by Equation (5-55), can also be used. However, based on numerical simulations that will be presented in subsection 6-5-2, squared group delay errors are more accurate to obtain good initial parameters rather than normalized ones.

Recall the formula of the group delay response derived in Equation (2-11). We can randomly initialize some \( \kappa_0 \) and \( \phi_0 \) from feasible set of \( \kappa \) and \( \phi \), and compute group delay response
using Equation (2-11). Using Equation (5-54) we can find the squared error of the group delay. Then, we compare the squared group delay error value, and the $\kappa_0$ and $\phi_0$ which result in the smallest squared group delay error will be used as the initial parameters.

Note that the OBFNs’ physical insight explained in the multi-start optimization section can also be implemented in this group delay error approach for simplifying the random initialization. This is done for the sake of fast computation, which is essential in tuning very-large-scale OBFNs. Algorithm 3 shows a pseudo-code for the initialization procedure using group delay error approach.

**Algorithm 3 Initialization using Group Delay Error Approach**

```plaintext
Initialization:
    Initialize $i = 1$
repeat
    Generate candidate initial parameters:
        Choose $\kappa_0$ and $\phi_0$ randomly from a feasible set of $\kappa$ and $\phi$ based on OBFNs’ physical insight
    Compute squared group delay error:
        Apply Equation (5-54) and (2-11) to obtain squared group delay error
    Update the smallest error:
        If $\kappa_0$ and $\phi_0$ improve the error
            Update the best $\kappa_0$ and $\phi_0$
    Increment iteration:
        $i = i + 1$
until (maximum number of iterations > than specified) OR (stopping condition is satisfied)
```

**6-5-1 Comparison between Random Initialization and Group Delay Error Approach**

Consider the case of a $4 \times 1$ OBFN. Based on the explanation in Section 6-4-2, candidate initial parameters $\kappa_0$ and $\phi_0$ are set to be

$$
\begin{bmatrix}
\kappa_0_1 & \kappa_0_2 & \kappa_0_3 & \kappa_0_4
\end{bmatrix} = \begin{bmatrix}
\kappa_a & \kappa_a & \kappa_a & \kappa_a
\end{bmatrix},
$$

(6-7)

$$
\begin{bmatrix}
\phi_0_1 & \phi_0_2 & \phi_0_3 & \phi_0_4
\end{bmatrix} = \begin{bmatrix}
\phi_c & \phi_c - \phi_a & \phi_c + \phi_a & \phi_c
\end{bmatrix},
$$

(6-8)

where $\kappa_a$ defines a randomly chosen value from the feasible set of $\kappa$, $\phi_c$ specifies the phase center, and $\phi_a$ denotes a randomly chosen phase from the feasible set of $\phi$. The numerical parameters used in the simulation correspond to the phase center $\phi_c = 0$ or similarly $\phi_c = 2\pi$.

We randomly select candidate initial parameters 200 times from which we perform multi-start optimization to get the best optimum parameters. Another method is to pre-test those 200 candidate initial parameters using group delay error approach. The candidate whose the smallest group delay error will be used as initial parameters. The comparison between both initialization methods are shown in Table 6-5.

We can observe in Table 6-5 that as the desired delay increases, $\kappa_a$ becomes smaller. This is expected as has been mentioned in the manual tuning principle in [2]. Smaller $\kappa_a$ is needed to achieve higher delay response. However, as we can expect, initialization using group delay error approach results in a shorter run time. This is due to the fact that the computationally expensive gradient descent is executed only once. While, in random initialization, gradient descent is executed every run of multi-start optimization. We can also see from the table that initial parameters obtained using group delay error approach are quite similar to multi-start optimization results. There are slight differences, but they both have reasonable final normalized squared group delay error.
Table 6-5: Initial guesses and run time comparison of two initialization methods: random initialization and group delay error approach for different desired delays $d[0.1 0.2 0.3]$ ns, gradient projection iteration $= 500$.

<table>
<thead>
<tr>
<th>Initialization method</th>
<th>Delay</th>
<th>$\kappa_a$</th>
<th>$\phi_a$</th>
<th>Run time</th>
<th>Final $E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random initialization</td>
<td>$d = 1$</td>
<td>0.99</td>
<td>0.1</td>
<td>109.135 s</td>
<td>2.2056E-04</td>
</tr>
<tr>
<td></td>
<td>$d = 2$</td>
<td>0.79</td>
<td>0</td>
<td>94.162 s</td>
<td>1.5494E-02</td>
</tr>
<tr>
<td></td>
<td>$d = 3$</td>
<td>0.59</td>
<td>0.3</td>
<td>93.773 s</td>
<td>3.9657E-02</td>
</tr>
<tr>
<td>Group delay error approach</td>
<td>$d = 1$</td>
<td>0.99</td>
<td>0</td>
<td>6.40 s</td>
<td>2.1117E-04</td>
</tr>
<tr>
<td></td>
<td>$d = 2$</td>
<td>0.79</td>
<td>0.2</td>
<td>6.22 s</td>
<td>1.3282E-02</td>
</tr>
<tr>
<td></td>
<td>$d = 3$</td>
<td>0.59</td>
<td>0.3</td>
<td>6.11 s</td>
<td>3.9657E-02</td>
</tr>
</tbody>
</table>

6-5-2 Simulation of Initialization using Group Delay Error Approach

It has been demonstrated that the group delay error approach can obtain good initial guesses as proven by the final normalized group delay error data in Table 6-5. However, it is important to make sure that the initialization method always come up with good initial guesses.

Figure 6-10: Histogram plots of 100 group delay errors of initialization results for $4 \times 1$ OBFNs with delays $d[0.1 0.2 0.3]$ns, where (a) $d = 1$, (b) $d = 2$, (c) $d = 3$.

Table 6-6: Initialization results using group delay error approach and final normalized squared group delay error of a $4 \times 1$ OBFNs for delay $= d[0.1 0.2 0.3]$, iteration $= 500$.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\kappa_a$</td>
<td>$d = 1$</td>
<td>$d = 2$</td>
</tr>
<tr>
<td></td>
<td>0.99</td>
<td>0.79</td>
</tr>
<tr>
<td>$\phi_a$</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>Final $E_{NSGD}$</td>
<td>2.092E-04</td>
<td>1.010E-02</td>
</tr>
</tbody>
</table>

We randomly select 200 candidate initial parameters, and pre-test them using group delay error approach. The best initial parameters in terms of the smallest squared group delay error will be saved. We repeat this procedure 100 times. Figure 6-10 shows the histogram plot of 100 group delay errors of $4 \times 1$ OBFNs for three different desired delays. From Figure 6-10 we can conclude that initialization using the group delay error approach will most likely obtain smallest group delay error (the best case), while the worst case has the lowest probability.

Table 6-6 shows the initialization results and final normalized squared group delay error for
both the best and the worst case. We can observe that the difference of final $E_{NSGD}$ between the best and the worst case for $d = 1$ is very small. However, for $d = 2$ and $d = 3$, the difference is quite significant. However, the probability to obtain the worst initialization is very small. We can increase the number of random candidate initial parameters to make sure we obtain good initial parameters.

It has been mentioned in the the beginning of Section 6-5 that pre-testing candidate initial parameters using squared group delay error is more accurate than using normalized squared group delay error. For $4 \times 1$ OBFN cases, this difference in accuracy is not detectable. But, we can clearly see the difference when we simulate $8 \times 1$ OBFNs. Consider $8 \times 1$ OBFNs with desired delay $d[0, 0.1, \cdots, 0.7]$ ns. Let the value of $d$ be equal to 2. We randomly choose 1000 candidate initial parameters, and pre-train them using those two errors. This procedure is repeated 100 times. Histogram plots of the squared group delay error and normalized one are shown in Figure 6-11.

![Figure 6-11: The histogram of (a) squared group delay error, (b) normalized squared group delay error for delay = 2·[0 0.1 0.2 0.3] ns. 1000 candidate initial parameters are randomly chosen, ones with the smallest error is saved. This procedure is repeated 100 times.](image)

From Figure 6-11 we can observe that both approaches indeed are able to obtain smallest error with relatively high probability and worst error with significantly low probability. Using the best initialization results from those two error approaches, we simulate an $8 \times 1$ OBFNs tuning problem. The resulting group delay response of those two initializations are shown in Figure 6-12. Resulting group delay response of initialization using squared group delay error (see Figure 6-12a) with final normalized squared group delay error ($E_{NSGD}$) equal to $7.4273 \times 10^{-3}$ is better than using normalized one (see Figure 6-12b) with $E_{NSGD} = 3.1029 \times 10^{-2}$ for the same 2000 gradient projection iterations. Using a lot more iterations indeed can make delay responses using normalized group delay initialization become close to the other method. However, this is not desirable, since it is inefficient.

### 6-6 Frequency Overfitting Problem

Tuning large-scale OBFNs is computationally expensive. This is due to the fact that the dimension of deep neural networks used to tune them is huge. The dimension is proportional
to the number of paths/AEs, and the number of frequencies of interest. While the number of paths/AEs is fixed, the number of frequencies can be appropriately changed. However, there is a trade-off in determining the number of frequencies. Choosing a few frequencies will make the algorithm optimize for those frequencies only, choosing too many is computationally expensive. This creates a frequency overfitting problem. It is desirable to choose a few frequencies for the sake of fast computation, but still able to prevent frequency overfitting.

Figure 6-13: Test error and group delay response from tuning a $4 \times 1$ OBFN with delay $3[0 0.1 0.2 0.3]$ ns. Nominal parameters: $N_{freq} = 3$, $N_{examples} = 20000$, $N_{stochastic} = 20$, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.9, iteration = 1000.
three frequencies only. It is observable that the test error indeed decreases to a very small number. However, even though the test error is very small, the group delay response is not good because the ripple is very big. This is due to the fact that the algorithm only optimizes for the given three frequencies.

Figure 6-14 shows the simulation result of a $4 \times 1$ OBFN with 20 frequencies. The algorithm optimizes more frequencies which results in better group delay response. However, this simulation takes more time (see Table 6-7 for comparison of run times). It is desirable to simulate using fewer frequencies for fast computation, but in the same time, frequency overfitting needs to be prevented. Random picking frequencies method is proposed to address this problem.

### 6-6-1 Random Picking Frequencies Method

The random picking frequencies method borrows the dropping concept from dropout. However, instead of dropping neurons with probability $p^{(l)}$ differently for each layer $l$, this method drops neurons of the entire layers for some frequencies every iteration. By doing so, input signals of the remaining frequencies will be propagated all the way to the output layer, and therefore there is not any information loss for those signals.

Given a frequency range $\mathbf{f} \in \mathbb{R}^N = [f_1 \ f_2 \ f_3 \ \cdots \ f_N]$, we randomly select $Q$ (note that $Q < N$) frequencies for every iteration such that

$$\hat{\mathbf{f}} \in \mathbb{R}^Q = [\hat{f}_1 \ \hat{f}_2 \ \hat{f}_3 \ \cdots \ \hat{f}_Q]$$

(6-9)

represents the new set of frequencies which will be optimized.
The input vector of the neural network is now defined by

$$\hat{x}^{(0)} \in \mathbb{R}^{2MQ} = \begin{bmatrix} x^{(0)}_1 \\ x^{(0)}_2 \\ \vdots \\ x^{(0)}_M \end{bmatrix} \quad \text{with} \quad \hat{x}^{(0)}_i \in \mathbb{R}^{2Q} = \begin{bmatrix} \text{Re} \{ \hat{x}^{(0)}_i (\hat{f}_1) \} \\ \text{Im} \{ \hat{x}^{(0)}_i (\hat{f}_1) \} \\ \vdots \\ \text{Re} \{ \hat{x}^{(0)}_i (\hat{f}_Q) \} \\ \text{Im} \{ \hat{x}^{(0)}_i (\hat{f}_Q) \} \end{bmatrix}, \quad (6-10)$$

where $M$ specifies the number of antenna element. The desired outputs are modified using the same principle as well. Note that the layers propagation, weight matrices configuration, and backpropagation algorithm should be properly modified to optimize the new set of frequencies, which changes randomly every iteration.

Note that this method is actually a special kind of input dropout regularization. The only difference with the usual input dropout is that instead of randomly dropping out the input vector with certain probability, we drop the input based on the random frequencies.

### 6-6-2 Random Picking Frequencies Results

We compare simulation results of two cases, fixed frequency in which we use the whole set of frequencies in the optimization, and random picking frequencies in which we randomly choose several frequencies from the set of frequencies. Figure 6-15 shows the comparison of test errors of those two cases. The first case is the fixed frequency case with 20 number of frequencies, while for the second case we pick randomly 3 frequencies from those 20 frequencies. We can see that both cases works well, test errors converge to a very small value. One interesting thing is that the test error using the random picking frequencies method converges faster than the fixed frequencies. However, it oscillates wilder compared to the fixed frequencies.

Table 6-7 shows the comparison of the final normalized squared group delay error and run time of three cases: fixed 20 frequencies, random frequencies with 3 out of 20 frequencies, and 3 out of 100 frequencies for three different desired delays. We can observe that all three cases have relatively similar error. One important thing to note is the run time comparison. Randomly picked frequencies significantly reduce the running time. This is caused by the lower number of frequencies computed in the neural network, which reduces the number of neurons.

The comparison results show that the randomly picked frequencies can significantly make the optimization faster with relatively similar error. This is desirable because we need a lot faster computation to tune larger setup of OBFN. In the next chapter, the simulation of larger OBFN setups will be presented to show that indeed deep learning can be used to tune large-scale OBFNs.
Figure 6-15: Test error results of a $4 \times 1$ OB FN with (a) fixed 20 frequencies, (b) randomly pick 3 out of 20 frequencies. Nominal parameters: $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 1000.

Table 6-7: Error and running time comparison of fixed frequencies and randomly pick frequencies case of a $4 \times 1$ OB FN. Nominal parameters: $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, desired delay = $d[0, 0.1, 0.2, 0.3]$ ns, frequency center $(f_c) = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 1000.

<table>
<thead>
<tr>
<th>Desired delay</th>
<th>Frequency</th>
<th>$E_{\text{NSGD}}$</th>
<th>Run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = 1$</td>
<td>Number of Freq</td>
<td>Sampling Freq</td>
<td>$E_{\text{NSGD}}$</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3</td>
<td>$2.0925 \times 10^{-4}$</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3</td>
<td>$2.00043 \times 10^{-4}$</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td></td>
<td>$1.9947 \times 10^{-4}$</td>
</tr>
<tr>
<td>$d = 2$</td>
<td>100</td>
<td>3</td>
<td>$1.070 \times 10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3</td>
<td>$1.050 \times 10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td></td>
<td>$1.020 \times 10^{-3}$</td>
</tr>
<tr>
<td>$d = 3$</td>
<td>100</td>
<td>3</td>
<td>$3.970 \times 10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3</td>
<td>$4.040 \times 10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td></td>
<td>$3.960 \times 10^{-3}$</td>
</tr>
</tbody>
</table>
Chapter 7

Simulations of Larger Setup of Optical Beamforming Networks

It has been shown in Chapter 5 that deep learning can be used to tune $4 \times 1$ OBFNs. In Chapter 6, we have analyzed several breakthroughs that make deep learning work well. In this chapter, the simulation of larger OBFN setups will be presented since we want to prove that the deep learning approach is indeed able to tune large OBFNs. We will use group delay error approach instead of random initialization or auto-encoders to obtain good initial parameters. In order to avoid frequency overfitting, we will implement random picking frequencies.

7-1 Simulation Setup

In this chapter we consider a setup of binary tree $2^\beta \times 1$ OBFN which consists of $\beta$ stages. The number of ORRs used ($r$) in the network is given. This OBFN configuration is translated into weight matrices as explained in Section 5-1-3.

It is assumed that the input signal is a signal comes from a very far source without any aberrations from the atmosphere. Noise signals and other signals coming from different direction are assumed to be very small and are neglected. The wavefront signal is flat and therefore the inputs coming to antenna elements are parallel to each other. The input signal arrives at different AEs with certain time differences/delays. These delay values are given. The given desired delay is defined by

$$D \in \mathbb{R}^M = d \begin{bmatrix} 0 & 0.1 & 0.2 & 0.3 & \cdots \end{bmatrix} \text{ ns},$$

where $d$ is any given positive real number and $M$ specifies number of AE.

Nominal parameters of the OBFN setup simulated in this chapter are similar to the ones presented in [2]. A round-trip time ($T$) for the ORR of 0.08 is used. The wavelength ($\lambda$) is 1550 nm. Frequency center of 107.5249817 THz and a phase center ($\phi_c$) of 0 or $2\pi$ are used. The bandwidth of interest is 2 GHz. This frequency range is discretized into $N$ number of
Simulations of Larger Setup of Optical Beamforming Networks

frequencies. $N = 20$ is chosen since it is faster and has no significant difference in terms of group delay error compared to bigger number, e.g., $N = 100$. Three frequencies are randomly chosen from those set of frequencies for fast computation and preventing frequency overfitting.

The input vector to the reference path is generated randomly. Given the desired delay and frequency range, the input vectors to other paths are generated based on the formula mentioned in Section 5-2-1. The number of randomly generated training examples is 40000, which consist of 20000 for training and 20000 for testing/validation. We will use the stochastic gradient projection method, in which we will randomly pick 20 examples from the whole set of training examples to be used in each optimization step. The separation of complex numbers is implemented to ensure that the resulting $\kappa^*$ and $\phi^*$ are real numbers. The learning rate of $2 \times 10^{-5}$ and momentum coefficient of 0.7 have been proven to be good choice in Section 6-1 and therefore will be used.

7-2 Simulation of 8 × 1 OBFNs

Consider the configuration of an 8 × 1 OBFN, which has 3 stages and 12 ORRs as shown in Figure 7-1. From this configuration, the derivation of weight matrices is straightforward as mentioned in Section 5-1-3.

Figure 7-1: The configuration of an 8 × 1 OBFN which consists of 3 stages and 12 ORRs.

7-2-1 Initialization

Based on the explanation in Section 6-4-2, we can set initial parameters $\kappa_0$ and $\phi_0$ for each ORR as presented in Table 7-1. The initialization based on the group delay error approach will find the value of $\kappa_a$, $\phi_{a1}$, $\phi_{a2}$, and $\phi_{a3}$.

The value of $\kappa_a$ will randomly be chosen from a set of feasible values [0.09, 0.19, ⋯, 0.99]. It is due to the fact that the value of $\kappa$ is bounded from 0.01 to 0.995 and the 0.1 increment is accurate enough to be used as initial guesses. The value of $\phi_{a1}$, $\phi_{a2}$, and $\phi_{a3}$ will be randomly
Table 7-1: Initial guesses for \(8 \times 1\) OBFNs. The initialization based on group delay error approach will find the value of \(\kappa_a, \phi_{a1}, \phi_{a2},\) and \(\phi_{a3} \). Nominal parameter \(\phi_c = 0\) or equivalently \(\phi_c = 2\pi\).

<table>
<thead>
<tr>
<th>Stage</th>
<th>ORR</th>
<th>Initial guesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(H_1) (H_4) (H_5) (H_{12})</td>
<td>(\kappa_a) (\phi_0)</td>
</tr>
<tr>
<td>2</td>
<td>(H_2) (H_{10})</td>
<td>(\kappa_a) (\phi_c - \phi_{a1})</td>
</tr>
<tr>
<td>3</td>
<td>(H_3) (H_{11})</td>
<td>(\kappa_a) (\phi_c + \phi_{a1})</td>
</tr>
<tr>
<td>3</td>
<td>(H_6)</td>
<td>(\kappa_a) (\phi_c - \phi_{a2})</td>
</tr>
<tr>
<td>3</td>
<td>(H_9)</td>
<td>(\kappa_a) (\phi_c + \phi_{a2})</td>
</tr>
<tr>
<td>3</td>
<td>(H_7)</td>
<td>(\kappa_a) (\phi_c - \phi_{a3})</td>
</tr>
<tr>
<td>3</td>
<td>(H_8)</td>
<td>(\kappa_a) (\phi_c + \phi_{a3})</td>
</tr>
</tbody>
</table>

chosen from the set of feasible value \([0.1, 0.2, 0.3, \cdots, 1]\) rad. This is due to the fact that, although the phase range is from 0 to \(2\pi\), but actually the phase from \(\phi_c - 1\) rad to \(\phi_c + 1\) rad is more than enough to cover the whole bandwidth as illustrated in Figure 7-2.

![Figure 7-2: Illustration of \(\phi_c \pm 1\) can cover a whole 2 GHz bandwidth.](image)

We randomly choose 1000 candidate initial parameters, and pre-test them using group delay error approach. The best initial parameters in terms of the smallest group delay error will be saved. This procedure is repeated 100 times to see the probability of the resulting group delay errors. Figure 7-3 shows histogram plots of 100 group delay errors for different desired delays. As we can observe from Figure 7-3, three highest probability results are the smallest group delay errors. The worst case group delay error have the lowest probability.

The best and the worst case from these 100 group delay errors will be used as initial parameters to tune \(8 \times 1\) OBFNs. These two cases are important to perform worst case analysis, where we compare optimization results using those two cases. If the comparison result differs insignificantly, it means that we can use whatever the result of initialization using group delay approach to tune OBFNs. Results of initialization are shown in Table 7-2 including the initial and final normalized group delay error.

Initialization results shown in Table 7-2 indicate that \(\kappa_a\) decreases as the desired delay increases. This as we expected and consistent with the behavior of the group delay response explained in Chapter 2.
Simulations of Larger Setup of Optical Beamforming Networks

Figure 7-3: Histogram plots of 100 errors of initialization results for $8 \times 1$ OBFNs with delays $d[0 \ 0.1 \ 0.2 \cdots 0.7]$ns, where (a) $d = 1$, (b) $d = 2$, (c) $d = 3$. 

Table 7-2: Initialization results, initial and final normalized squared group delay errors using group delay error approach of an $8 \times 1$ OBFN for delays $d[0 \ 0.1 \ 0.2 \cdots 0.7]$. 1000 random candidate initial parameters are chosen. Final $E_{NSGD}$ is obtained after 2000 gradient projection iteration with learning rate $= 2E-5$ and momentum coefficient $= 0.7$. 

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = 1$</td>
<td>$d = 2$</td>
<td>$d = 3$</td>
</tr>
<tr>
<td>$\kappa_0$</td>
<td>0.99</td>
<td>0.69</td>
</tr>
<tr>
<td>$\phi_{\kappa_1}$</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>$\phi_{\kappa_2}$</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>$\phi_{\kappa_3}$</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>Initial $E_{NSGD}$</td>
<td>3.86E-03</td>
<td>3.96E-02</td>
</tr>
<tr>
<td>Final $E_{NSGD}$</td>
<td>5.08E-04</td>
<td>7.40E-03</td>
</tr>
</tbody>
</table>

7-2-2 Tuning Results

After obtaining a set of initial parameters, we have to train the neural network to obtain optimum parameters. The resulting final normalized squared group delay errors are presented in Table 7-2 for both the best and the worst case. We can see that the final normalized group delay errors are reasonable, which prove that the deep learning can tune $8 \times 1$ OBFNs well. More detail result of tuning $8 \times 1$ OBFNs is shown in Table A-4 in the Appendix A. Table A-4 presents optimum values ($\kappa^*$ and $\phi^*$) of all ORRs and normalized squared error of the resulting group delay responses compared to desired ones.

We can see from Table A-4 that for small delay, initial guesses $\kappa_0$ and $\phi_0$ are very close to optimum values. Let the optimum values be

$$
\kappa^*_i = \kappa_0 + \Delta \kappa_i \\
\phi^*_i = \phi_0 + \Delta \phi_i
$$

(7-1)

where $\Delta \kappa_i$ and $\Delta \phi_i$ define the spread of optimum values from initial guesses. For small delay, these spreads are very small. However, for bigger delay, they increase. This is due to the fact that in order to achieve big time difference between paths, ORRs should have wider spread of parameters. This is one of disadvantages of initialization based on physical insight which simplifies initialization variables into $\kappa_a$ and $\phi_a$ only. This also explains the trade-off between initialization accuracy and its computational cost.

Herminarto Nugroho
Master of Science Thesis
Figure 7-4: Group delay responses of $8 \times 1$ OBFNs for delay $d[0, 0.1, 0.2, \cdots, 0.7]$ ns with $d = 1$, $d = 2$, and $d = 3$. (left) the best case, (right) the worst case. Nominal parameters: $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, $f_c = 107.52$ THz, $N_{\text{freq}} = 20$, random freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 2000.
Figure 7-4 shows the results of the group delay response for three desired delays. We can see that the ripple increases as the desired delays increases. This is caused by the number of ORRs available cannot perfectly cover high delays to create flat delay response for a specified 2 GHz bandwidth. This is expected and consistent with the explanation in Chapter 2. The ripple creates bigger group delay errors, as we already observed in Table 7-2, where the normalized group delay error increases as the desired delay increases.

**Worst Case Analysis**

Randomly choosing initial parameters from their set of feasible values has a problem. It is highly possible that initial parameters from a random trial are not good enough, i.e., they are not close enough to optimum values. It is of course not desirable because it can result in high group delay error, which affect the output signal strength badly. One solution to this problem is by randomly choosing many initial parameters. This is not as simple as it sounds. There is no exact number which is considered to be sufficient to obtain good initial parameters. Selecting too many numbers is not a good idea as well, since it will slow down the initialization process.

We have obtained the best and the worst case of 100 initialization results shown in Table 7-2. Tuning $8 \times 1$ OBFNs using these two cases is performed, and normalized squared group delay errors and group delay response plots of those two cases will be compared to each other. The final normalized group delay errors for both the best and the worst case should be close.

Table 7-2 shows the final normalized squared group delay error comparison between those two cases. We can observe that the final normalized squared group delay error of the worst case is not significantly far from the best one. The error for the worst case is still acceptable. Figure 7-4 also shows that the resulting group delay response plots for three different desired delays are reasonable, both the best and the worst case.

**Comparison with Non-linear Programming Method**

Blokpoel [2] has developed a non-linear programming (NLP) method to tune $4 \times 1$ OBFNs. We modify the method presented in [2] such that it can be used to tune $8 \times 1$ OBFNs. The comparison of optimum parameters, final normalized group delay error, and total run time of tuning $8 \times 1$ OBFN with 200 iterations using deep learning approach and NLP is presented in Table 7-3. We can observe that both cases have almost similar optimum parameters. If we compare the group delay response in Figure 7-5, it is apparent that NLP has better result compared to deep learning approach, which is also shown by the resulting normalized squared group delay errors ($E_{NSGD}$) comparison in Table 7-3. However, both group delay responses are still reasonable.

We can also see from Table 7-3 that NLP has longer run time compared to deep learning for 200 iterations. However, if we simulate for more iteration, e.g., 1000 iterations, the final $E_{NSGD}$ of NLP does not change, while for deep learning, the final $E_{NSGD}$ decreases into $E_{NSGD} = 7.40\text{E-}3$. This means NLP needs fewer iteration to converge, which is better than deep learning approach.

Even though NLP performs better than deep learning, it is not data driven. The NLP method tunes OBFN systems from a given desired delay or desired phase. Deep learning,
on the other hand, is data driven, which tunes OBFN systems from a given set of training examples (signals). This is essential for online tuning in the future research.

**Table 7-3:** Comparison of optimum parameters, final normalized squared group delay error, and total run time for 200 iterations of tuning $8 \times 1$ OBFN using deep learning approach and Non-linear programming (NLP). Both simulations used the same initial parameters.

<table>
<thead>
<tr>
<th>ORR</th>
<th>Initial $\kappa_0$, $\phi_0$</th>
<th>Deep learning approach $\kappa^<em>$, $\phi^</em>$, $E_{NSGD}$</th>
<th>Time</th>
<th>NLP $\kappa^<em>$, $\phi^</em>$, $E_{NSGD}$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.69 0</td>
<td>0.790 0.009</td>
<td>7.55E-3</td>
<td>0.788 0.009</td>
<td>3.956 s</td>
</tr>
<tr>
<td>2</td>
<td>0.69 5.883</td>
<td>0.666 0.411</td>
<td>6.47E-3</td>
<td>0.634 0.439</td>
<td>38.250 s</td>
</tr>
<tr>
<td>3</td>
<td>0.69 0.4</td>
<td>0.792 0.009</td>
<td></td>
<td>0.784 0.009</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.69 0</td>
<td>0.792 0.007</td>
<td></td>
<td>0.789 0.009</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.69 5.883</td>
<td>0.666 5.914</td>
<td></td>
<td>0.623 6.089</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.69 5.883</td>
<td>0.666 5.914</td>
<td>3.956 s</td>
<td>0.486 5.670</td>
<td>6.47E-3</td>
</tr>
<tr>
<td>7</td>
<td>0.69 0.4</td>
<td>0.667 0.412</td>
<td></td>
<td>0.623 0.212</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.69 0.4</td>
<td>0.667 0.412</td>
<td></td>
<td>0.486 0.632</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.69 5.883</td>
<td>0.674 5.889</td>
<td></td>
<td>0.661 5.884</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.69 0.4</td>
<td>0.666 0.407</td>
<td></td>
<td>0.661 0.418</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.69 0</td>
<td>0.792 0.006</td>
<td></td>
<td>0.786 0.009</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.69 5.883</td>
<td>0.674 0.412</td>
<td></td>
<td>0.623 0.212</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7-5:** Group delay responses of $8 \times 1$ OBFNs for delay $2[0, 0.1, 0.2, \cdots, 0.7]$ ns. (a) deep learning approach, (b) non-linear programming. Nominal parameters: $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, $f_c = 107.52$ THz, $N_{\text{freq}} = 20$, random freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 200.

7-3 Simulation of $16 \times 1$ OBFNs

Consider the configuration of a $16 \times 1$ OBFN depicted in Figure 7-6 with 4 stages and 32 ORRs. This configuration can be translated into set of weight matrices as mentioned in Section 5-1-3.
Figure 7-6: The configuration of a $16 \times 1$ OBFN which consists of 4 stages and 32 ORRs.

### 7-3-1 Initialization

We can set the candidate initial guesses of $\kappa_0$ and $\phi_0$ of each ORR as presented in Table 7-4.

Table 7-4: Initial guesses for a $16 \times 1$ OBFN. Nominal parameter $\phi_c = 0$ or equivalently $\phi_c = 2\pi$.

<table>
<thead>
<tr>
<th>Stage</th>
<th>ORR</th>
<th>Initial guesses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$H_1\ H_4\ H_5\ H_{12}\ H_{13}\ H_{16}\ H_{17}\ H_{24}$</td>
<td>$\kappa_a\ \phi_0\ \phi_c$</td>
</tr>
<tr>
<td>1</td>
<td>$H_2\ H_{10}\ H_{14}\ H_{22}$</td>
<td>$\kappa_a\ \phi_c - \phi_{a_2}$</td>
</tr>
<tr>
<td>2</td>
<td>$H_3\ H_{11}\ H_{15}\ H_{23}$</td>
<td>$\kappa_a\ \phi_c + \phi_{a_1}$</td>
</tr>
<tr>
<td>3</td>
<td>$H_6\ H_{18}$</td>
<td>$\kappa_a\ \phi_c - \phi_{a_2}$</td>
</tr>
<tr>
<td></td>
<td>$H_9\ H_{21}$</td>
<td>$\kappa_a\ \phi_c + \phi_{a_2}$</td>
</tr>
<tr>
<td></td>
<td>$H_7\ H_{19}$</td>
<td>$\kappa_a\ \phi_c - \phi_{a_3}$</td>
</tr>
<tr>
<td></td>
<td>$H_8\ H_{20}$</td>
<td>$\kappa_a\ \phi_c + \phi_{a_3}$</td>
</tr>
<tr>
<td>4</td>
<td>$H_{25}$</td>
<td>$\kappa_a\ \phi_c - \phi_{a_4}$</td>
</tr>
<tr>
<td></td>
<td>$H_{32}$</td>
<td>$\kappa_a\ \phi_c + \phi_{a_4}$</td>
</tr>
<tr>
<td></td>
<td>$H_{26}$</td>
<td>$\kappa_a\ \phi_c - \phi_{a_5}$</td>
</tr>
<tr>
<td></td>
<td>$H_{31}$</td>
<td>$\kappa_a\ \phi_c + \phi_{a_5}$</td>
</tr>
<tr>
<td></td>
<td>$H_{27}$</td>
<td>$\kappa_a\ \phi_c - \phi_{a_6}$</td>
</tr>
<tr>
<td></td>
<td>$H_{30}$</td>
<td>$\kappa_a\ \phi_c + \phi_{a_6}$</td>
</tr>
<tr>
<td></td>
<td>$H_{28}$</td>
<td>$\kappa_a\ \phi_c - \phi_{a_7}$</td>
</tr>
<tr>
<td></td>
<td>$H_{29}$</td>
<td>$\kappa_a\ \phi_c + \phi_{a_7}$</td>
</tr>
</tbody>
</table>

The same value of $\phi_c = 0$ or $\phi_c = 2\pi$ is used. The initialization based on group delay error
approach will find the value of $\kappa_a$, and $\phi_{a_1}$, \ldots, $\phi_{a_7}$. Similar to the initialization of $8 \times 1$ OBFNs (Section 7-2-1), the value of $\kappa_a$ will randomly be chosen from set of feasible value $[0.09, 0.19, \ldots, 0.99]$. The value of $\phi_{a_1}$, \ldots, $\phi_{a_7}$ will be randomly chosen from the set of feasible value $[0.1, 0.2, \ldots, 1]$ rad.

Since the number of parameters to be initialized in $16 \times 1$ OBFNs (8 parameters) is more than in $8 \times 1$ OBFNs (4 parameters), we need more random candidate initial parameters. We randomly choose 2000 candidate initial parameters, and pre-test them using group delay error approach. The best candidate initial parameters in terms of the smallest group delay error will be saved. This procedure is repeated 100 times. Figure 7-7 shows histogram plots of 100 group delay errors for three desired delays. As we can see from Figure 7-7, the worst error has the lowest probability.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure7-7.png}
\caption{Histogram plots of 100 errors of initialization results for $16 \times 1$ OBFNs with delays $d[0, 0.1, 0.2, \ldots, 1.5]$ns, where (a) $d = 1$, (b) $d = 2$, (c) $d = 3$.}
\end{figure}

The best and the worst case from 100 group delay errors (see Table 7-5) will be used as initial parameters to tune $16 \times 1$ OBFNs. The final normalized squared group delay error of both cases should be close. The comparison of the normalized squared group delay error between the two cases is also shown in Table 7-5. We can see that the normalized squared group delay error of both cases are actually close.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
Parameter & Best case & & & & & \\
\hline
 & $d = 1$ & $d = 2$ & $d = 3$ & $d = 1$ & $d = 2$ & $d = 3$ \\
\hline $\kappa_a$ & 0.99 & 0.69 & 0.39 & 0.99 & 0.69 & 0.39 \\
$\phi_{a_1}$ & 0.1 & 0.5 & 0.3 & 0.1 & 0.5 & 0.4 \\
$\phi_{a_2}$ & 0.2 & 0.3 & 0.5 & 0.3 & 0.2 & 0.3 \\
$\phi_{a_3}$ & 0 & 0.5 & 0.2 & 0.2 & 0.5 & 0.4 \\
$\phi_{a_4}$ & 0.1 & 0.5 & 0.4 & 0.1 & 0.4 & 0 \\
$\phi_{a_5}$ & 0.1 & 0.2 & 0.4 & 0.2 & 0.1 & 0.3 \\
$\phi_{a_6}$ & 0.1 & 0.4 & 0.4 & 0.5 & 0.4 & 0.5 \\
$\phi_{a_7}$ & 0 & 0.6 & 0 & 0.1 & 0.4 & 0.3 \\
\hline Initial $E_{NSGD}$ & 8.82E-3 & 6.22E-2 & 2.74E-1 & 1.10E-2 & 6.40E-2 & 3.00E-1 \\
Final $E_{NSGD}$ & 1.02E-3 & 9.16E-3 & 4.83E-2 & 1.17E-3 & 1.04E-2 & 5.17E-2 \\
\hline
\end{tabular}
\caption{Initialization results and normalized squared group delay errors of an $16 \times 1$ OBFN for delays $d[0, 0.1, \ldots, 1.5]$. 2000 random initial parameters are chosen. Final $E_{NSGD}$ is obtained after 5000 iteration with learning rate = 2E-5 and momentum coefficient = 0.7.}
\end{table}
Figure 7-8: Group delay responses of a $16 \times 1$ OBFN for desired delay $d_{\{0, 0.1, 0.2, \cdots, 1.5\}}$ ns with $d = 1$, $d = 2$, and $d = 3$. (left) best case, (right) worst case initialization. Nominal parameters: $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, $f_c = 107.52$ THz, $N_{\text{freq}} = 20$, random freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 5000.
7-3 Simulation of $16 \times 1$ OBFNs

7-3-2 Tuning Results

More detailed tuning results of $16 \times 1$ OBFNs are presented in Table A-5, A-6, and A-7 in the Appendix. They present optimum parameters all ORRs and the normalized squared group delay errors for 3 different delays. Similar to the result of $8 \times 1$ OBFNs explained in Section 7-2-2, we can see that for small delay, initial guesses are very close to optimum values. For higher delay, the spread of optimum values from initial guesses increases. This is expected and has been mentioned in Section 7-2-2. Figure 7-8 shows group delay responses of three delays for both the best and the worst initialization results. Resulting plots are reasonable. As we have expect, the ripple increases as the delay increases, which also can be proven by the final normalized squared group delay errors in Table 7-5.

Worst Case Analysis

Table 7-5 shows the normalized squared group delay error comparison between the best and the worst initialization for $d = 1$, $d = 2$, and $d = 3$. We can see from those resulting errors that the worst case results are not significantly far from the best one. They are still acceptable.

The same conclusion can be obtained when we observe Figure 7-8. The resulting group delay response plot from three different delays are reasonable for both cases. These results prove that the initialization works well, since it can ensure that we obtain good optimum parameters even from worst initialization result.

Comparison with Non-linear Programming Method

Similar to the comparison between deep learning approach and non-linear programming for tuning $8 \times 1$ OBFNs, we can also modify the method developed by Blokpoel [2] such that it now can be used to tune $16 \times 1$ OBFNs. The comparison of optimum parameters, normalized squared group delay error, and total run time of tuning $16 \times 1$ OBFN using deep learning approach and NLP are presented in Table A-8 in the Appendix. Table 7-6 only shows the error and total run time comparison.

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Deep Learning Approach</th>
<th>Non-linear Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon_{\text{NSGD}}$</td>
<td>8.45E-3</td>
<td>6.57E-3</td>
</tr>
<tr>
<td>Total run time</td>
<td>251.09 s</td>
<td>1109.542 s</td>
</tr>
</tbody>
</table>

We can observe from Table A-8 in the Appendix that both approaches have almost similar optimum parameters. However, the final normalized squared group delay error obtained by NLP is better compared to deep learning, as shown in Table 7-6. If we increase the iteration to 2000, the normalized squared group delay error of NLP method does not change. On the other hand, the normalized squared group delay error of deep learning decreases slightly to 8.32E-3. Similar to the result of $8 \times 1$ OBFNs, for tuning $16 \times 1$ OBFNs, NLP also needs fewer iteration to converge compared to the deep learning approach.
Figure 7-9 shows the group delay responses of those two approaches. We can see that both group delay responses are reasonable compared to the normalized squared group delay error presented in Table A-8 and Table 7-6.

Figure 7-9: Group delay responses of 16 × 1 OBFNs for delay 2[0, 0.1, 0.2, · · · , 1.5] ns. (a) deep learning approach, (b) non-linear programming. Nominal parameters: Nexamples = 20000, Nstochastic = 20, fc = 107.52 THz, Nfreq = 20, random freq = 3, bandwidth = 2 GHz, learning rate = 2 × 10−5, momentum coefficient = 0.7, iteration = 1000.

7-4 Simulation of a 32 × 1 OBFN

Two 16 × 1 OBFNs (illustrated in Figure 7-6) which are combined with one additional stage whose 16 ORRs can create a configuration of a 32 × 1 OBFN. This means, a 32 × 1 OBFN has 80 ORRs and 5 stages. This configuration can be translated into set of weight matrices following the procedure as mentioned in Section 5-1-3.

7-4-1 Initialization

We can set initial guesses of κ0 and φ0 as presented in Table 7-7. The same value of phase center is used, which is φc = 0 or φc = 2π. Initialization algorithm will find the value of κa, and φa1, φa2, · · · , φa15. We randomly choose 5000 candidate initial guesses, and pre-test them using group delay error approach. The reason to use 5000 random initialization is because there are 16 variables need to be initialized, so we need more random guesses than in case of 8 × 1 and 16 × 1 OBFNs. From these 5000 candidate initial parameters, a set of initial parameters with the smallest group delay error will be saved. This procedure is repeated 100 times.

Figure 7-10 shows the histogram plots of 100 group delay errors for three different delays. As we have expected, the worst error has the lowest probability. The resulting values of κa, and φa1, φa2, · · · , φa15 for both best and worst case are presented in Table 7-8. Resulting κa are consistent with the behavior of group delay response mentioned in Chapter 2.
parameters from both the best and the worst case are used as initial parameters for tuning $32 \times 1$ OBFNs.

### 7-4-2 Tuning Results

More detail tuning results of $32 \times 1$ OBFNs are presented in Table A-9, A-10, and A-11 in the Appendix. They present the optimum parameters of all 80 ORRs and the normalized squared group delay errors for 3 desired delays. Similar to the observation of tuning results of $8 \times 1$ (see Section 7-2-2) and $16 \times 1$ OBFNs (see Section 7-3-2), we can also see that for small delay, initial guesses in the case of $32 \times 1$ are also close to optimum values. However, for bigger delay, the spread of optimum parameters from initial guesses is increasing. This is expected as a trade-off between initialization accuracy and its computation cost mentioned.
Simulations of Larger Setup of Optical Beamforming Networks

**Figure 7-10:** Histogram plots of 100 errors of initialization results for $32 \times 1$ OBFNs with delays $d[0.1 \ 0.2 \ \cdots \ \cdots \ \cdot \ 3.1]$ns, where (a) $d = 1$, (b) $d = 2$, (c) $d = 3$.

**Table 7-8:** Initialization results and normalized squared group delay errors of an $32 \times 1$ OBFN for delays $d[0.1 \ 0.2 \ \cdots \ \cdot \ 1.5]$. 5000 random candidate initial parameters are chosen. Final $E_{NSGD}$ is obtained after 10000 gradient projection iteration with learning rate $= 2E-5$ and momentum coefficient $= 0.7$.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\kappa_a$</td>
<td>$\begin{bmatrix} 0.99 &amp; 0.59 &amp; 0.39 \ 0.99 &amp; 0.59 &amp; 0.39 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0.8 &amp; 0.6 &amp; 0.2 \ 0.1 &amp; 0.4 &amp; 0.5 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a1}$</td>
<td>$\begin{bmatrix} 0.4 &amp; 0.5 &amp; 0.4 \ 0 &amp; 0.3 &amp; 0.3 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 1 &amp; 0.4 &amp; 0.2 \ 0.1 &amp; 0.4 &amp; 0.5 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a2}$</td>
<td>$\begin{bmatrix} 0 &amp; 0.6 &amp; 0.7 \ 0 &amp; 0.5 &amp; 0.4 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0.4 &amp; 0.5 &amp; 0.5 \ 0.6 &amp; 0.1 &amp; 0.1 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a3}$</td>
<td>$\begin{bmatrix} 0.3 &amp; 0.1 &amp; 0.1 \ 0 &amp; 0.6 &amp; 0.4 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0.5 &amp; 0.6 &amp; 0.3 \ 0.6 &amp; 0.1 &amp; 0.1 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a4}$</td>
<td>$\begin{bmatrix} 0.1 &amp; 0.6 &amp; 0.5 \ 0 &amp; 0.6 &amp; 0.5 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0 &amp; 0.6 &amp; 0.3 \ 0 &amp; 0.6 &amp; 0.5 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a5}$</td>
<td>$\begin{bmatrix} 0.7 &amp; 0.6 &amp; 0.3 \ 0.3 &amp; 0.5 &amp; 0.1 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0.5 &amp; 0.6 &amp; 0.3 \ 0.2 &amp; 0.4 &amp; 0.5 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a6}$</td>
<td>$\begin{bmatrix} 0.1 &amp; 0.6 &amp; 0.5 \ 0 &amp; 0.6 &amp; 0.5 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0 &amp; 0.6 &amp; 0.3 \ 0 &amp; 0.6 &amp; 0.5 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a7}$</td>
<td>$\begin{bmatrix} 0.2 &amp; 0.4 &amp; 0.4 \ 0 &amp; 0.6 &amp; 0.2 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0.4 &amp; 0.5 &amp; 0 \ 0.4 &amp; 0.6 &amp; 0.2 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a8}$</td>
<td>$\begin{bmatrix} 0 &amp; 0.6 &amp; 0.4 \ 0 &amp; 0.6 &amp; 0.4 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0 &amp; 0.5 &amp; 0.4 \ 0 &amp; 0.4 &amp; 0.2 \end{bmatrix}$</td>
</tr>
<tr>
<td>$\phi_{a9}$</td>
<td>$\begin{bmatrix} 0 &amp; 0.6 &amp; 0.4 \ 0 &amp; 0.6 &amp; 0.4 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0 &amp; 0.5 &amp; 0.4 \ 0 &amp; 0.4 &amp; 0.2 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

Initial $E_{NSGD} = 2.48E-2 \ \ 1.51E-2 \ \ 4.11E-1 \ \ 5.42E-2 \ \ 2.00E-2 \ \ 5.66E-1$

Final $E_{NSGD} = 1.50E-3 \ \ 9.06E-3 \ \ 7.99E-2 \ \ 2.11E-3 \ \ 1.03E-2 \ \ 8.00E-2$

in Section 7-2-2. Figure 7-11 shows group delay responses of three delays for both best and worst initializations. As we have expected, the ripple increases as the delay increases.

**Worst Case Analysis**

Based on the final normalized squared group delay error results from Table 7-8 we can see that indeed the final group delay error difference between the best and the worst case is relatively significant. However, the chance of choosing bad case is low. We can increase the number of random initialization, e.g., 10000 to ensure the initialization result in good initial parameters.

Herminarto Nugroho

Master of Science Thesis
Figure 7-11: Group delay responses of a $32 \times 1$ OBFN for desired delay $d\{0, 0.1, 0.2, \cdots, 3.1\}$ ns with $d = 1$, $d = 2$, and $d = 3$. (left) best case, (right) worst case initialization. Nominal parameters: Nexamples $= 20000$, Nstochastic $= 20$, $f_c = 107.52$ THz, Nfreq $= 20$, random freq $= 3$, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.7$, iteration $= 10000$. 
Chapter 8

Conclusion and Recommendation

8-1 Conclusion

Phased Array Antennas (PAAs) controlled by Optical Beamforming Networks (OBFNs) are a promising solution for communication between planes and satellites. However, OBFNs pose challenges because of their highly non-linear nature which makes them difficult to tune. Recent tuning methods, which use black-box non-linear programming solvers, are limited to small-scale OBFNs. Therefore, a novel method has to be developed in order to be able to tune large-scale OBFNs efficiently.

Ideas from deep learning seem to be promising for tuning large OBFNs. Tuning large OBFNs is a complex large-scale and highly non-linear problem, and ideas from deep learning such as backpropagation, auto-encoders, and dropout regularization seem to be useful for training them. However, in order to analyze how principles of deep learning can be applied, a comparison between OBFN systems and neural networks has to be carried out.

The problem of tuning a large-scale OBFNs is in many aspects similar to training a deep neural network. The special structure of OBFNs can be represented by a neural network with many layers. The weight matrices are composed of frequency responses of some ORRs in the respective layer. Given a certain structure of a binary tree OBFN, this method works well to find the optimum parameters, under the condition that the initial value of the parameters are close enough to optima. This is why initialization is very important.

Pre-training using auto-encoders was proposed as a method to find good initial parameters. However, auto-encoders failed to extract useful information from the neural network. Consequently, they cannot be used to obtain good initial parameters. Multi-start optimization with random initial guesses based on OBFNs physical insight as an alternative solution is also proven to be inefficient when being implemented for tuning large-scale OBFNs. Therefore, initialization based on group delay error approach was proposed. This method is computationally cheap and has been proven to work well for large OBFN setups.

Another issue which needs to be tackled is overfitting. Dropout regularization was proposed to tackle overfitting. However, the special structure of the neural network representation
of OBFNs causes the dropout regularization to not work. It is due to the fact that dropout regularization results in losing too much information when being implemented in this network. Another problem called frequency overfitting, will be addressed by random picking frequencies method. This method borrows the dropping concept from dropout, but instead of dropping neurons in each layer, this method drops neurons for some frequencies for the entire layers.

The final conclusion is that deep learning approach has been proven to work well for tuning large-scale OBFNs, e.g., $8 \times 1$, $16 \times 1$, and $32 \times 1$ binary tree OBFNs for any desired delays. The deep learning approach is data driven, which tunes OBFN systems from a given set of training examples (signals). This is an advantage of the deep learning approach, because the signals are measurable so we can use real data, which is essential for online tuning in the future research.

### 8-2 Recommendation

The first recommendation will be regarding the initialization method. On this thesis, the initialization based on group delay approach was used. Moreover, to make it more efficient, choosing candidate initial guesses based on physical insight was implemented to reduce the number of initialization variables. This approach obviously cannot be implemented in arbitrary OBFN setups. It is limited to binary tree structure. An initialization method which can be implemented to any given OBFN structures is desirable and needs to be investigated. For example, instead of randomly choosing initial parameters from a given feasible set, we can design a heuristic algorithm based on framework that can be directly used to solve any problems. In multi-start concept, there are tabu-search [60], scatter search [61], and greedy randomized adaptive search procedures (GRASP) [63].

The optimization algorithm in this thesis is an off-line optimization method, in which the desired delays and OBFN structure were given. In the real application, on-line optimization needs to be designed. This includes further research about how fast the optimization is, how often the tuning is updated, and how to predict aberration in real time and compensate it. Note that the deep learning approach is already data-driven. We could use real data in the future implementation. This is essential for on-line tuning process.
Appendix A

Detailed Numerical Results

A-1 Simulation of $4 \times 1$ OBFNs

A-1.1 Simulation Result of a $4 \times 1$ OBFN for Several Desired Delays

Table A-1: The resulting $\kappa_i^*$ and $\phi_i^*$ of $4 \times 1$ OBFNs for several delays, with $\kappa_0 = [0 \text{.}9 \text{,} 0 \text{.}9 \text{,} 0 \text{.}9 \text{,} 0 \text{.}9]$ and $\phi_0 = [0 \text{,} (2\pi - 0 \text{.}4) \text{,} 0 \text{.}4 \text{,} 0]$ using stochastic gradient projection. Nominal parameters: $N_{\text{freq}} = 10$, $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000.

<table>
<thead>
<tr>
<th>$d$</th>
<th>$\kappa_1^*$</th>
<th>$\kappa_2^*$</th>
<th>$\kappa_3^*$</th>
<th>$\kappa_4^*$</th>
<th>$\phi_1^*$</th>
<th>$\phi_2^*$</th>
<th>$\phi_3^*$</th>
<th>$\phi_4^*$</th>
<th>$E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9870</td>
<td>0.9873</td>
<td>0.9803</td>
<td>0.9933</td>
<td>0.0018</td>
<td>5.9191</td>
<td>0.3713</td>
<td>0.0027</td>
<td>2.0235 x 10^{-4}</td>
</tr>
<tr>
<td>1.1</td>
<td>0.9737</td>
<td>0.9608</td>
<td>0.9640</td>
<td>0.9721</td>
<td>0.0018</td>
<td>5.8285</td>
<td>0.4564</td>
<td>0.0000</td>
<td>4.5599 x 10^{-4}</td>
</tr>
<tr>
<td>1.2</td>
<td>0.9570</td>
<td>0.9404</td>
<td>0.9425</td>
<td>0.9561</td>
<td>0.0018</td>
<td>5.8829</td>
<td>0.3972</td>
<td>0.0000</td>
<td>8.2673 x 10^{-4}</td>
</tr>
<tr>
<td>1.3</td>
<td>0.9387</td>
<td>0.9191</td>
<td>0.9221</td>
<td>0.9410</td>
<td>0.0018</td>
<td>5.8813</td>
<td>0.4041</td>
<td>0.0000</td>
<td>1.3115 x 10^{-3}</td>
</tr>
<tr>
<td>1.4</td>
<td>0.9192</td>
<td>0.8958</td>
<td>0.8940</td>
<td>0.9242</td>
<td>0.0018</td>
<td>5.8790</td>
<td>0.4031</td>
<td>0.0000</td>
<td>1.9003 x 10^{-3}</td>
</tr>
<tr>
<td>1.5</td>
<td>0.8989</td>
<td>0.8680</td>
<td>0.8624</td>
<td>0.9053</td>
<td>0.0018</td>
<td>5.8796</td>
<td>0.3996</td>
<td>0.0000</td>
<td>2.5756 x 10^{-3}</td>
</tr>
<tr>
<td>1.6</td>
<td>0.8781</td>
<td>0.8341</td>
<td>0.8279</td>
<td>0.8848</td>
<td>0.0018</td>
<td>5.8796</td>
<td>0.4004</td>
<td>0.0000</td>
<td>3.3153 x 10^{-3}</td>
</tr>
<tr>
<td>1.7</td>
<td>0.8570</td>
<td>0.7858</td>
<td>0.7808</td>
<td>0.8624</td>
<td>0.0018</td>
<td>5.8637</td>
<td>0.4117</td>
<td>0.0000</td>
<td>4.0978 x 10^{-3}</td>
</tr>
<tr>
<td>1.8</td>
<td>0.8359</td>
<td>0.7167</td>
<td>0.7158</td>
<td>0.8372</td>
<td>0.0018</td>
<td>5.8387</td>
<td>0.4443</td>
<td>0.0000</td>
<td>4.9126 x 10^{-3}</td>
</tr>
<tr>
<td>1.9</td>
<td>0.8148</td>
<td>0.6676</td>
<td>0.6705</td>
<td>0.8132</td>
<td>0.0001</td>
<td>5.8424</td>
<td>0.4405</td>
<td>0.0000</td>
<td>5.7753 x 10^{-3}</td>
</tr>
<tr>
<td>2</td>
<td>0.7940</td>
<td>0.6413</td>
<td>0.6450</td>
<td>0.7920</td>
<td>0.0001</td>
<td>5.8612</td>
<td>0.4215</td>
<td>0.0000</td>
<td>6.7455 x 10^{-3}</td>
</tr>
<tr>
<td>2.1</td>
<td>0.7733</td>
<td>0.6177</td>
<td>0.6220</td>
<td>0.7716</td>
<td>0.0001</td>
<td>5.8787</td>
<td>0.4037</td>
<td>0.0000</td>
<td>7.9416 x 10^{-3}</td>
</tr>
<tr>
<td>2.2</td>
<td>0.7529</td>
<td>0.5947</td>
<td>0.6000</td>
<td>0.7517</td>
<td>0.0001</td>
<td>5.8939</td>
<td>0.3882</td>
<td>0.0000</td>
<td>9.5362 x 10^{-3}</td>
</tr>
<tr>
<td>2.3</td>
<td>0.7328</td>
<td>0.5723</td>
<td>0.5790</td>
<td>0.7322</td>
<td>0.0001</td>
<td>5.9072</td>
<td>0.3746</td>
<td>0.0000</td>
<td>1.1721 x 10^{-2}</td>
</tr>
<tr>
<td>2.4</td>
<td>0.7130</td>
<td>0.5505</td>
<td>0.5596</td>
<td>0.7128</td>
<td>0.0001</td>
<td>5.9192</td>
<td>0.3626</td>
<td>0.0000</td>
<td>1.4645 x 10^{-2}</td>
</tr>
<tr>
<td>2.5</td>
<td>0.6935</td>
<td>0.5291</td>
<td>0.5418</td>
<td>0.6932</td>
<td>0.0001</td>
<td>5.9301</td>
<td>0.3521</td>
<td>0.0000</td>
<td>1.8360 x 10^{-2}</td>
</tr>
<tr>
<td>2.6</td>
<td>0.6744</td>
<td>0.5081</td>
<td>0.5261</td>
<td>0.6734</td>
<td>0.0002</td>
<td>5.9402</td>
<td>0.3427</td>
<td>0.0000</td>
<td>2.2816 x 10^{-2}</td>
</tr>
<tr>
<td>2.7</td>
<td>0.6557</td>
<td>0.4879</td>
<td>0.5123</td>
<td>0.6529</td>
<td>0.0002</td>
<td>5.9497</td>
<td>0.3340</td>
<td>0.0000</td>
<td>2.7907 x 10^{-2}</td>
</tr>
<tr>
<td>2.8</td>
<td>0.6374</td>
<td>0.4681</td>
<td>0.4997</td>
<td>0.6311</td>
<td>0.0002</td>
<td>5.8577</td>
<td>0.3259</td>
<td>0.0000</td>
<td>3.3540 x 10^{-2}</td>
</tr>
<tr>
<td>2.9</td>
<td>0.6196</td>
<td>0.4482</td>
<td>0.4878</td>
<td>0.6082</td>
<td>0.0003</td>
<td>5.9677</td>
<td>0.3186</td>
<td>0.0000</td>
<td>3.9682 x 10^{-2}</td>
</tr>
<tr>
<td>3</td>
<td>0.6024</td>
<td>0.4285</td>
<td>0.4762</td>
<td>0.5859</td>
<td>0.0003</td>
<td>5.9767</td>
<td>0.3122</td>
<td>0.0020</td>
<td>4.6366 x 10^{-2}</td>
</tr>
</tbody>
</table>
A-1-2  Simulation Result of a 4 × 1 OBFN for Some Different Initializations

Table A-2: The resulting $\kappa_i^*$ and $\phi_i^*$ of 4 × 1 OBFNs for different $\kappa_0$ but the same $\phi_0 = [0,(2\pi - 0.4),0.4,0]$ using stochastic gradient projection. Nominal parameters: Nfreq = 10, Nexamples = 20000, Nstochastic = 20, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000.

<table>
<thead>
<tr>
<th>$\kappa_0$</th>
<th>ORR 1</th>
<th>ORR 2</th>
<th>ORR 3</th>
<th>ORR 4</th>
<th>$E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.9 0.9 0.9 0.9]</td>
<td>$\kappa^<em>$ 0.9870, $\phi^</em>$ 0.0018</td>
<td>0.9873, 5.9191</td>
<td>0.9803, 0.3713</td>
<td>0.9933, 0.0027</td>
<td>$2.0235 \times 10^{-4}$</td>
</tr>
<tr>
<td>[0.7 0.7 0.7 0.7]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0024</td>
<td>0.9870, 5.8783</td>
<td>0.9798, 0.4059</td>
<td>0.9867, 0.0012</td>
<td>$1.9079 \times 10^{-4}$</td>
</tr>
<tr>
<td>[0.5 0.5 0.5 0.5]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0029</td>
<td>0.9861, 5.8502</td>
<td>0.9797, 0.4357</td>
<td>0.9868, 0.0009</td>
<td>$1.8535 \times 10^{-4}$</td>
</tr>
<tr>
<td>[0.3 0.3 0.3 0.3]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0032</td>
<td>0.0326, 5.8729</td>
<td>0.0295, 0.4199</td>
<td>0.5986, 0.0033</td>
<td>$2.2898$</td>
</tr>
<tr>
<td>[0.9 0.8 0.7 0.6]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0020</td>
<td>0.9874, 5.8793</td>
<td>0.9794, 0.4026</td>
<td>0.9867, 0.0017</td>
<td>$1.9478 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

Table A-3: The resulting $\kappa_i^*$ and $\phi_i^*$ of 4 × 1 OBFNs for different $\phi_0$ but the same $\kappa_0 = [0.9,0.9,0.9,0.9]$ using stochastic gradient projection. Nominal parameters: Nfreq = 10, Nexamples = 20000, Nstochastic = 20, delay = [0 0.1 0.2 0.3] ns, $f_c = 107.52$ THz, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, iteration = 1000.

<table>
<thead>
<tr>
<th>$\phi_0$</th>
<th>ORR 1</th>
<th>ORR 2</th>
<th>ORR 3</th>
<th>ORR 4</th>
<th>$E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0 (2$\pi$ − 0.4) 0.4 0]</td>
<td>$\kappa^<em>$ 0.9870, $\phi^</em>$ 0.0018</td>
<td>0.9873, 5.9191</td>
<td>0.9803, 0.3713</td>
<td>0.9933, 0.0027</td>
<td>$2.0235 \times 10^{-4}$</td>
</tr>
<tr>
<td>[0 (2$\pi$ − 0.6) 0.6 0]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0020</td>
<td>0.9810, 5.6827</td>
<td>0.9749, 0.6008</td>
<td>0.9868, 0.0009</td>
<td>$1.6707 \times 10^{-4}$</td>
</tr>
<tr>
<td>[0 (2$\pi$ − 0.8) 0.8 0]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0020</td>
<td>0.9640, 5.4822</td>
<td>0.9577, 0.8013</td>
<td>0.9877, 0.0009</td>
<td>$1.1615 \times 10^{-4}$</td>
</tr>
<tr>
<td>[0 (2$\pi$ − 1.0) 1.0 0]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0020</td>
<td>0.9037, 5.2838</td>
<td>0.8952, 0.9996</td>
<td>0.9770, 0.0011</td>
<td>$7.3265 \times 10^{-4}$</td>
</tr>
<tr>
<td>[0 (2$\pi$ − 1.2) 1.2 0]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0020</td>
<td>0.9481, 5.0860</td>
<td>0.9414, 1.1972</td>
<td>0.9422, 0.0014</td>
<td>$9.3128 \times 10^{-3}$</td>
</tr>
<tr>
<td>[0.1 (2$\pi$ − 0.4) 0.4 0]</td>
<td>$\kappa^<em>$ 0.9922, $\phi^</em>$ 0.0822</td>
<td>0.9873, 5.8830</td>
<td>0.9797, 0.4005</td>
<td>0.9868, 0.0010</td>
<td>$1.5972 \times 10^{-3}$</td>
</tr>
<tr>
<td>[0.1 (2$\pi$ − 0.4) 0.4 0.1]</td>
<td>$\kappa^<em>$ 0.9922, $\phi^</em>$ 0.0822</td>
<td>0.9779, 5.8827</td>
<td>0.9844, 0.4002</td>
<td>0.9934, 0.0923</td>
<td>$1.7089 \times 10^{-3}$</td>
</tr>
<tr>
<td>[0 (2$\pi$ − 0.4) 0.6 0]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0020</td>
<td>0.9430, 5.8742</td>
<td>0.9950, 0.5945</td>
<td>0.9950, 0.004</td>
<td>$2.2239 \times 10^{-3}$</td>
</tr>
<tr>
<td>[0 (2$\pi$ − 0.4) 0.8 0]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0020</td>
<td>0.8857, 5.8469</td>
<td>0.9950, 0.7823</td>
<td>0.9950, 0.004</td>
<td>$1.5347 \times 10^{-2}$</td>
</tr>
<tr>
<td>[0 π 0.8 0]</td>
<td>$\kappa^<em>$ 0.9867, $\phi^</em>$ 0.0020</td>
<td>0.9950, 3.1788</td>
<td>0.1096, 0.8138</td>
<td>0.7629, 0.0401</td>
<td>$1.5210 \times 10^{-1}$</td>
</tr>
</tbody>
</table>
A-2 Simulation of $8 \times 1$ OBFNs

Table A-4: Initial and optimum values of $\kappa$ and $\phi$ for some different desired delays obtained by tuning $8 \times 1$ OBFNs considering two initialization cases: the best and the worst case. Nominal parameters: Nexamples = 20000, Nstochastic = 20, desired delay = $d[0, 0.1, 0.2, \cdots, 0.7]$ ns, $f_c = 107.52$ THz, Nfreq = 20, random picked freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$; momentum coefficient = 0.7; iteration = 2000.

<table>
<thead>
<tr>
<th>$d$</th>
<th>ORR</th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Initial</td>
<td>Optimum</td>
</tr>
<tr>
<td>1</td>
<td>0.99</td>
<td>0.986</td>
<td>0.007</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>6.283</td>
<td>0.992</td>
</tr>
<tr>
<td>3</td>
<td>0.99</td>
<td>0.980</td>
<td>0.017</td>
</tr>
<tr>
<td>4</td>
<td>0.99</td>
<td>0.986</td>
<td>0.009</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.987</td>
<td>0.005</td>
</tr>
<tr>
<td>6</td>
<td>0.99</td>
<td>6.283</td>
<td>0.986</td>
</tr>
<tr>
<td>7</td>
<td>0.99</td>
<td>6.283</td>
<td>0.986</td>
</tr>
<tr>
<td>8</td>
<td>0.99</td>
<td>0.987</td>
<td>0.020</td>
</tr>
<tr>
<td>9</td>
<td>0.99</td>
<td>0.987</td>
<td>0.020</td>
</tr>
<tr>
<td>10</td>
<td>0.99</td>
<td>6.283</td>
<td>0.987</td>
</tr>
<tr>
<td>11</td>
<td>0.99</td>
<td>0.986</td>
<td>0.014</td>
</tr>
<tr>
<td>12</td>
<td>0.99</td>
<td>0.986</td>
<td>0.007</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>0.790</td>
<td>0.009</td>
</tr>
<tr>
<td>3</td>
<td>0.99</td>
<td>5.883</td>
<td>0.665</td>
</tr>
<tr>
<td>4</td>
<td>0.99</td>
<td>0.4</td>
<td>0.662</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.792</td>
<td>0.008</td>
</tr>
<tr>
<td>6</td>
<td>0.99</td>
<td>0.792</td>
<td>0.006</td>
</tr>
<tr>
<td>7</td>
<td>0.99</td>
<td>5.883</td>
<td>0.665</td>
</tr>
<tr>
<td>8</td>
<td>0.99</td>
<td>0.4</td>
<td>0.666</td>
</tr>
<tr>
<td>9</td>
<td>0.99</td>
<td>0.4</td>
<td>0.666</td>
</tr>
<tr>
<td>10</td>
<td>0.99</td>
<td>5.883</td>
<td>0.673</td>
</tr>
<tr>
<td>11</td>
<td>0.99</td>
<td>0.4</td>
<td>0.665</td>
</tr>
<tr>
<td>12</td>
<td>0.69</td>
<td>0.792</td>
<td>0.006</td>
</tr>
<tr>
<td>3</td>
<td>0.39</td>
<td>0.593</td>
<td>0.008</td>
</tr>
<tr>
<td>4</td>
<td>0.39</td>
<td>5.883</td>
<td>0.434</td>
</tr>
<tr>
<td>5</td>
<td>0.39</td>
<td>0.4</td>
<td>0.435</td>
</tr>
<tr>
<td>6</td>
<td>0.39</td>
<td>0.589</td>
<td>0.009</td>
</tr>
<tr>
<td>7</td>
<td>0.39</td>
<td>0.616</td>
<td>0.007</td>
</tr>
<tr>
<td>8</td>
<td>0.39</td>
<td>6.083</td>
<td>0.352</td>
</tr>
<tr>
<td>9</td>
<td>0.39</td>
<td>5.783</td>
<td>0.311</td>
</tr>
<tr>
<td>10</td>
<td>0.39</td>
<td>0.5</td>
<td>0.300</td>
</tr>
<tr>
<td>11</td>
<td>0.39</td>
<td>0.2</td>
<td>0.343</td>
</tr>
<tr>
<td>12</td>
<td>0.39</td>
<td>0.4</td>
<td>0.410</td>
</tr>
</tbody>
</table>

Master of Science Thesis Herminarto Nugroho
A-3 Simulation of $16 \times 1$ OBFNs

Table A-5: Initial and optimum values of $\kappa$ and $\phi$ for some different desired delays obtained by tuning $16 \times 1$ OBFNs considering two initialization cases: the best and the worst case. Nominal parameters: $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, desired delay = $[0, 0.1, 0.2, \ldots, 1.5]$ ns, $f_c = 107.52$ THz, $N_{\text{freq}} = 20$, random picked freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 5000.

<table>
<thead>
<tr>
<th>$d$</th>
<th>ORR</th>
<th>$E_{\text{NSGD}}$</th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Initial</td>
<td>Optimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\kappa$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1.02E-3</td>
<td>0.99</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1.17E-3</td>
<td>0.99</td>
<td>0</td>
</tr>
</tbody>
</table>

Herminarto Nugroho
Master of Science Thesis
Table A-6: Initial and optimum values of $\kappa$ and $\phi$ for some different desired delays obtained by tuning $16 \times 1$ OBFNs considering two initialization cases: the best and the worst case. Nominal parameters: Nexamples = 20000, Nstochastic = 20, desired delay = $[0, 0.1, 0.2, \cdots , 1.5]$ ns, $f_c = 107.52$ THz, Nfreq = 20, random picked freq = 3, bandwidth = 2 GHz, learning rate = $2 \times 10^{-5}$, momentum coefficient = 0.7, iteration = 5000.

<table>
<thead>
<tr>
<th>$d$</th>
<th>ORR</th>
<th>Initial</th>
<th>Optimum</th>
<th>$E_{NSGD}$</th>
<th>Initial</th>
<th>Optimum</th>
<th>$E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>1</td>
<td>0.69</td>
<td>0</td>
<td>0.791</td>
<td>0.009</td>
<td>0.69</td>
<td>0</td>
<td>0.791</td>
</tr>
<tr>
<td>2</td>
<td>0.69</td>
<td>5.783</td>
<td>0.632</td>
<td>5.862</td>
<td>9.16E-3</td>
<td>0.69</td>
<td>5.783</td>
</tr>
<tr>
<td>3</td>
<td>0.69</td>
<td>0.5</td>
<td>0.631</td>
<td>0.440</td>
<td>0.69</td>
<td>0.5</td>
<td>0.631</td>
</tr>
<tr>
<td>4</td>
<td>0.69</td>
<td>0</td>
<td>0.788</td>
<td>0.009</td>
<td>0.69</td>
<td>0</td>
<td>0.788</td>
</tr>
<tr>
<td>5</td>
<td>0.69</td>
<td>0</td>
<td>0.797</td>
<td>0.009</td>
<td>0.69</td>
<td>0</td>
<td>0.797</td>
</tr>
<tr>
<td>6</td>
<td>0.69</td>
<td>5.983</td>
<td>0.651</td>
<td>5.989</td>
<td>0.69</td>
<td>6.083</td>
<td>0.707</td>
</tr>
<tr>
<td>7</td>
<td>0.69</td>
<td>5.783</td>
<td>0.675</td>
<td>5.790</td>
<td>0.69</td>
<td>5.783</td>
<td>0.615</td>
</tr>
<tr>
<td>8</td>
<td>0.69</td>
<td>0.5</td>
<td>0.638</td>
<td>0.506</td>
<td>0.69</td>
<td>0.5</td>
<td>0.588</td>
</tr>
<tr>
<td>9</td>
<td>0.69</td>
<td>0.3</td>
<td>0.678</td>
<td>0.350</td>
<td>0.69</td>
<td>0.2</td>
<td>0.735</td>
</tr>
<tr>
<td>10</td>
<td>0.69</td>
<td>5.783</td>
<td>0.603</td>
<td>5.835</td>
<td>0.69</td>
<td>5.783</td>
<td>0.602</td>
</tr>
<tr>
<td>11</td>
<td>0.69</td>
<td>0.5</td>
<td>0.600</td>
<td>0.465</td>
<td>0.69</td>
<td>0.5</td>
<td>0.600</td>
</tr>
<tr>
<td>12</td>
<td>0.69</td>
<td>0</td>
<td>0.780</td>
<td>0.009</td>
<td>0.69</td>
<td>0</td>
<td>0.780</td>
</tr>
<tr>
<td>13</td>
<td>0.69</td>
<td>0</td>
<td>0.805</td>
<td>0.004</td>
<td>0.69</td>
<td>0</td>
<td>0.808</td>
</tr>
<tr>
<td>14</td>
<td>0.69</td>
<td>5.783</td>
<td>0.609</td>
<td>5.806</td>
<td>0.69</td>
<td>5.783</td>
<td>0.615</td>
</tr>
<tr>
<td>15</td>
<td>0.69</td>
<td>0.5</td>
<td>0.586</td>
<td>0.472</td>
<td>0.69</td>
<td>0.5</td>
<td>0.593</td>
</tr>
<tr>
<td>16</td>
<td>0.69</td>
<td>0</td>
<td>0.765</td>
<td>0.006</td>
<td>0.69</td>
<td>0</td>
<td>0.763</td>
</tr>
<tr>
<td>17</td>
<td>0.69</td>
<td>0</td>
<td>0.810</td>
<td>0.009</td>
<td>0.69</td>
<td>0</td>
<td>0.803</td>
</tr>
<tr>
<td>18</td>
<td>0.69</td>
<td>5.983</td>
<td>0.605</td>
<td>6.006</td>
<td>0.69</td>
<td>6.083</td>
<td>0.623</td>
</tr>
<tr>
<td>19</td>
<td>0.69</td>
<td>5.783</td>
<td>0.896</td>
<td>5.806</td>
<td>0.69</td>
<td>5.783</td>
<td>0.983</td>
</tr>
<tr>
<td>20</td>
<td>0.69</td>
<td>0.3</td>
<td>0.695</td>
<td>0.329</td>
<td>0.69</td>
<td>0.2</td>
<td>0.721</td>
</tr>
<tr>
<td>21</td>
<td>0.69</td>
<td>0.3</td>
<td>0.695</td>
<td>0.329</td>
<td>0.69</td>
<td>0.2</td>
<td>0.721</td>
</tr>
<tr>
<td>22</td>
<td>0.69</td>
<td>5.783</td>
<td>0.579</td>
<td>5.805</td>
<td>0.69</td>
<td>5.783</td>
<td>0.543</td>
</tr>
<tr>
<td>23</td>
<td>0.69</td>
<td>0.5</td>
<td>0.579</td>
<td>0.499</td>
<td>0.69</td>
<td>0.5</td>
<td>0.546</td>
</tr>
<tr>
<td>24</td>
<td>0.69</td>
<td>0</td>
<td>0.755</td>
<td>0.004</td>
<td>0.69</td>
<td>0</td>
<td>0.761</td>
</tr>
<tr>
<td>25</td>
<td>0.69</td>
<td>5.783</td>
<td>0.627</td>
<td>5.791</td>
<td>0.69</td>
<td>5.883</td>
<td>0.619</td>
</tr>
<tr>
<td>26</td>
<td>0.69</td>
<td>6.083</td>
<td>0.623</td>
<td>6.089</td>
<td>0.69</td>
<td>6.183</td>
<td>0.819</td>
</tr>
<tr>
<td>27</td>
<td>0.69</td>
<td>5.883</td>
<td>0.609</td>
<td>5.889</td>
<td>0.69</td>
<td>5.883</td>
<td>0.619</td>
</tr>
<tr>
<td>28</td>
<td>0.69</td>
<td>5.683</td>
<td>0.685</td>
<td>5.692</td>
<td>0.69</td>
<td>5.883</td>
<td>0.619</td>
</tr>
<tr>
<td>29</td>
<td>0.69</td>
<td>0.6</td>
<td>0.692</td>
<td>0.605</td>
<td>0.69</td>
<td>0.4</td>
<td>0.622</td>
</tr>
<tr>
<td>30</td>
<td>0.69</td>
<td>0.4</td>
<td>0.648</td>
<td>0.405</td>
<td>0.69</td>
<td>0.4</td>
<td>0.622</td>
</tr>
<tr>
<td>31</td>
<td>0.69</td>
<td>0.2</td>
<td>0.627</td>
<td>0.203</td>
<td>0.69</td>
<td>0.1</td>
<td>0.843</td>
</tr>
<tr>
<td>32</td>
<td>0.69</td>
<td>0.5</td>
<td>0.627</td>
<td>0.505</td>
<td>0.69</td>
<td>0.4</td>
<td>0.622</td>
</tr>
</tbody>
</table>
Table A-7: Initial and optimum values of $\kappa$ and $\phi$ for some different desired delays obtained by tuning $16 \times 1$ OBFNs considering two initialization cases: the best and the worst case. Nominal parameters: $N_{\text{examples}} = 20000$, $N_{\text{stochastic}} = 20$, desired delay $= [0, 0.1, 0.2, \ldots, 1.5]$ ns, $f_c = 107.52$ THz, $N_{\text{freq}} = 20$, random picked freq $= 3$, bandwidth $= 2$ GHz, learning rate $= 2 \times 10^{-5}$, momentum coefficient $= 0.7$, iteration $= 5000$.

<table>
<thead>
<tr>
<th>$d$</th>
<th>ORR</th>
<th>Best case</th>
<th></th>
<th></th>
<th>Worst case</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Initial</td>
<td>Optimum</td>
<td>$E_{\text{NSGD}}$</td>
<td>Initial</td>
<td>Optimum</td>
<td>$E_{\text{NSGD}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>1</td>
<td>0.39</td>
<td>0.592</td>
<td>0.009</td>
<td>0.39</td>
<td>0.592</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.39</td>
<td>5.883</td>
<td>0.436</td>
<td>5.989</td>
<td>0.39</td>
<td>5.833</td>
<td>0.435</td>
</tr>
<tr>
<td>3</td>
<td>0.39</td>
<td>0.4</td>
<td>0.313</td>
<td>0.39</td>
<td>0.4</td>
<td>0.435</td>
<td>0.39</td>
</tr>
<tr>
<td>4</td>
<td>0.39</td>
<td>0.589</td>
<td>0.009</td>
<td>0.39</td>
<td>0.589</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.39</td>
<td>0.602</td>
<td>0.009</td>
<td>0.39</td>
<td>0.601</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.39</td>
<td>6.083</td>
<td>0.394</td>
<td>6.135</td>
<td>0.39</td>
<td>6.083</td>
<td>0.394</td>
</tr>
<tr>
<td>7</td>
<td>0.39</td>
<td>5.783</td>
<td>0.312</td>
<td>5.833</td>
<td>0.39</td>
<td>5.783</td>
<td>0.312</td>
</tr>
<tr>
<td>8</td>
<td>0.39</td>
<td>0.5</td>
<td>0.469</td>
<td>0.39</td>
<td>0.4</td>
<td>0.428</td>
<td>0.157</td>
</tr>
<tr>
<td>9</td>
<td>0.39</td>
<td>0.2</td>
<td>0.166</td>
<td>0.39</td>
<td>0.2</td>
<td>0.328</td>
<td>0.448</td>
</tr>
<tr>
<td>10</td>
<td>0.39</td>
<td>5.883</td>
<td>0.454</td>
<td>5.997</td>
<td>0.39</td>
<td>5.883</td>
<td>0.454</td>
</tr>
<tr>
<td>11</td>
<td>0.39</td>
<td>0.4</td>
<td>0.305</td>
<td>0.39</td>
<td>0.4</td>
<td>0.456</td>
<td>0.303</td>
</tr>
<tr>
<td>12</td>
<td>0.39</td>
<td>0.592</td>
<td>0.009</td>
<td>0.39</td>
<td>0.590</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.39</td>
<td>0.619</td>
<td>0.004</td>
<td>0.39</td>
<td>0.635</td>
<td>0.010</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.39</td>
<td>5.883</td>
<td>0.441</td>
<td>5.958</td>
<td>0.39</td>
<td>5.883</td>
<td>0.402</td>
</tr>
<tr>
<td>15</td>
<td>0.39</td>
<td>0.425</td>
<td>0.330</td>
<td>0.39</td>
<td>0.405</td>
<td>0.363</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.39</td>
<td>0</td>
<td>0.008</td>
<td>0.39</td>
<td>0</td>
<td>0.550</td>
<td>0.008</td>
</tr>
<tr>
<td>17</td>
<td>0.39</td>
<td>0</td>
<td>0.014</td>
<td>0.39</td>
<td>0</td>
<td>0.635</td>
<td>0.002</td>
</tr>
<tr>
<td>18</td>
<td>0.39</td>
<td>6.083</td>
<td>0.353</td>
<td>6.104</td>
<td>0.39</td>
<td>6.083</td>
<td>0.353</td>
</tr>
<tr>
<td>19</td>
<td>0.39</td>
<td>5.783</td>
<td>0.636</td>
<td>5.837</td>
<td>0.39</td>
<td>5.783</td>
<td>0.636</td>
</tr>
<tr>
<td>20</td>
<td>0.39</td>
<td>0.2</td>
<td>0.282</td>
<td>0.39</td>
<td>0.4</td>
<td>0.424</td>
<td>0.324</td>
</tr>
<tr>
<td>21</td>
<td>0.39</td>
<td>0.2</td>
<td>0.282</td>
<td>0.39</td>
<td>0.4</td>
<td>0.424</td>
<td>0.324</td>
</tr>
<tr>
<td>22</td>
<td>0.39</td>
<td>5.883</td>
<td>0.333</td>
<td>5.921</td>
<td>0.39</td>
<td>5.883</td>
<td>0.416</td>
</tr>
<tr>
<td>23</td>
<td>0.39</td>
<td>0.4</td>
<td>0.402</td>
<td>0.39</td>
<td>0.5</td>
<td>0.404</td>
<td>0.431</td>
</tr>
<tr>
<td>24</td>
<td>0.39</td>
<td>0</td>
<td>0.005</td>
<td>0.39</td>
<td>0</td>
<td>0.384</td>
<td>0.006</td>
</tr>
<tr>
<td>25</td>
<td>0.39</td>
<td>5.783</td>
<td>0.247</td>
<td>5.789</td>
<td>0.39</td>
<td>5.683</td>
<td>0.228</td>
</tr>
<tr>
<td>26</td>
<td>0.39</td>
<td>6.183</td>
<td>0.312</td>
<td>6.191</td>
<td>0.39</td>
<td>5.983</td>
<td>0.364</td>
</tr>
<tr>
<td>27</td>
<td>0.39</td>
<td>5.883</td>
<td>0.385</td>
<td>5.889</td>
<td>0.39</td>
<td>5.983</td>
<td>0.364</td>
</tr>
<tr>
<td>28</td>
<td>0.39</td>
<td>5.983</td>
<td>0.362</td>
<td>5.995</td>
<td>0.39</td>
<td>6.283</td>
<td>0.360</td>
</tr>
<tr>
<td>29</td>
<td>0.39</td>
<td>0.3</td>
<td>0.304</td>
<td>0.39</td>
<td>0</td>
<td>0.361</td>
<td>0.015</td>
</tr>
<tr>
<td>30</td>
<td>0.39</td>
<td>0.4</td>
<td>0.411</td>
<td>0.39</td>
<td>0.3</td>
<td>0.358</td>
<td>0.320</td>
</tr>
<tr>
<td>31</td>
<td>0.39</td>
<td>0.1</td>
<td>0.106</td>
<td>0.39</td>
<td>0.3</td>
<td>0.358</td>
<td>0.320</td>
</tr>
<tr>
<td>32</td>
<td>0.39</td>
<td>0.5</td>
<td>0.324</td>
<td>0.39</td>
<td>0.6</td>
<td>0.188</td>
<td>0.535</td>
</tr>
</tbody>
</table>

Herminarto Nugroho Master of Science Thesis
Comparison with Non-linear Programming Method

Table A-8: Comparison of optimum parameters, final normalized squared group delay error, and total run time of tuning 16 $\times$ 1 OBFN using deep learning approach and non-linear programming. Both simulations use the same initial parameters and 1000 iterations.

<table>
<thead>
<tr>
<th>ORR</th>
<th>Initial $\kappa_0$</th>
<th>$\phi_0$</th>
<th>Deep learning approach $\kappa^*$</th>
<th>$\phi^*$</th>
<th>$E_{NSGD}$</th>
<th>Time (s)</th>
<th>NLP $\kappa^*$</th>
<th>$\phi^*$</th>
<th>$E_{NSGD}$</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.69</td>
<td>0</td>
<td>0.791</td>
<td>0.009</td>
<td>0.788</td>
<td>0.009</td>
<td>8.45E-03</td>
<td>251.09</td>
<td>6.57E-3</td>
<td>1109.542</td>
</tr>
<tr>
<td>2</td>
<td>0.69</td>
<td>5.783</td>
<td>0.637</td>
<td>5.866</td>
<td>0.634</td>
<td>5.862</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.69</td>
<td>0.5</td>
<td>0.637</td>
<td>0.436</td>
<td>0.634</td>
<td>0.439</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.69</td>
<td>0</td>
<td>0.789</td>
<td>0.009</td>
<td>0.784</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.69</td>
<td>0</td>
<td>0.795</td>
<td>0.009</td>
<td>0.789</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.69</td>
<td>5.983</td>
<td>0.636</td>
<td>5.989</td>
<td>0.623</td>
<td>6.089</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.69</td>
<td>5.783</td>
<td>0.665</td>
<td>5.790</td>
<td>0.486</td>
<td>5.670</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.69</td>
<td>0.5</td>
<td>0.633</td>
<td>0.305</td>
<td>0.632</td>
<td>0.317</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.69</td>
<td>0.3</td>
<td>0.680</td>
<td>0.305</td>
<td>0.632</td>
<td>0.317</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.69</td>
<td>5.783</td>
<td>0.610</td>
<td>5.841</td>
<td>0.661</td>
<td>5.884</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.69</td>
<td>0.5</td>
<td>0.608</td>
<td>0.459</td>
<td>0.661</td>
<td>0.418</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.69</td>
<td>0</td>
<td>0.783</td>
<td>0.009</td>
<td>0.786</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.69</td>
<td>0</td>
<td>0.801</td>
<td>0.007</td>
<td>0.789</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.69</td>
<td>5.783</td>
<td>0.593</td>
<td>5.818</td>
<td>0.661</td>
<td>5.884</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.69</td>
<td>0.5</td>
<td>0.582</td>
<td>0.475</td>
<td>0.661</td>
<td>0.418</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.69</td>
<td>0</td>
<td>0.772</td>
<td>0.007</td>
<td>0.786</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0.69</td>
<td>0</td>
<td>0.809</td>
<td>0.005</td>
<td>0.789</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0.69</td>
<td>5.983</td>
<td>0.649</td>
<td>5.987</td>
<td>0.628</td>
<td>6.086</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0.69</td>
<td>5.783</td>
<td>0.675</td>
<td>5.789</td>
<td>0.527</td>
<td>5.672</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.69</td>
<td>0.5</td>
<td>0.633</td>
<td>0.505</td>
<td>0.528</td>
<td>0.633</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0.69</td>
<td>0.3</td>
<td>0.679</td>
<td>0.305</td>
<td>0.629</td>
<td>0.221</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>0.69</td>
<td>5.783</td>
<td>0.596</td>
<td>5.802</td>
<td>0.661</td>
<td>5.884</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>0.69</td>
<td>0.5</td>
<td>0.582</td>
<td>0.489</td>
<td>0.661</td>
<td>0.418</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>0.69</td>
<td>0</td>
<td>0.756</td>
<td>0.004</td>
<td>0.786</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>0.69</td>
<td>5.783</td>
<td>0.636</td>
<td>5.791</td>
<td>0.395</td>
<td>5.624</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>0.69</td>
<td>6.083</td>
<td>0.619</td>
<td>6.089</td>
<td>0.636</td>
<td>6.136</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>0.69</td>
<td>5.883</td>
<td>0.618</td>
<td>5.889</td>
<td>0.666</td>
<td>5.877</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>0.69</td>
<td>5.683</td>
<td>0.684</td>
<td>5.691</td>
<td>0.745</td>
<td>5.804</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>0.69</td>
<td>0.6</td>
<td>0.613</td>
<td>0.604</td>
<td>0.621</td>
<td>0.055</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0.69</td>
<td>0.4</td>
<td>0.655</td>
<td>0.404</td>
<td>0.529</td>
<td>0.458</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>0.69</td>
<td>0.2</td>
<td>0.620</td>
<td>0.202</td>
<td>0.624</td>
<td>0.260</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0.69</td>
<td>0.5</td>
<td>0.635</td>
<td>0.504</td>
<td>0.371</td>
<td>0.681</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## A-4 Simulation of $32 \times 1$ OBFNs

**Table A-9:** Initial and optimum parameters for $32 \times 1$ OBFNs. Nexamples = 20/20k, $d = 1$, Nfreq = 3/20, learning rate = $2 \times 10^{-5}$, momentum = 0.7, iteration = 10000.

<table>
<thead>
<tr>
<th>$d$</th>
<th>ORR</th>
<th>Initial</th>
<th>Optimum</th>
<th>$E_{NSGD}$</th>
<th>Initial</th>
<th>Optimum</th>
<th>$E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>1</td>
<td>1.50E-3</td>
<td>21.11E-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>5.883</td>
<td>0.987</td>
<td>5.884</td>
<td>0.99</td>
<td>5.483</td>
<td>0.965</td>
</tr>
<tr>
<td>3</td>
<td>0.99</td>
<td>0.4</td>
<td>0.980</td>
<td>0.401</td>
<td>0.99</td>
<td>0.8</td>
<td>0.960</td>
</tr>
<tr>
<td>4</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.008</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>6.283</td>
<td>0.987</td>
<td>6.283</td>
<td>0.99</td>
<td>6.183</td>
<td>0.975</td>
</tr>
<tr>
<td>6</td>
<td>0.99</td>
<td>6.283</td>
<td>0.987</td>
<td>6.283</td>
<td>0.99</td>
<td>5.283</td>
<td>0.922</td>
</tr>
<tr>
<td>7</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.019</td>
<td>0.99</td>
<td>1</td>
<td>0.908</td>
</tr>
<tr>
<td>8</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.019</td>
<td>0.99</td>
<td>0.1</td>
<td>0.987</td>
</tr>
<tr>
<td>9</td>
<td>0.99</td>
<td>5.883</td>
<td>0.987</td>
<td>5.884</td>
<td>0.99</td>
<td>5.483</td>
<td>0.965</td>
</tr>
<tr>
<td>10</td>
<td>0.99</td>
<td>0.4</td>
<td>0.980</td>
<td>0.401</td>
<td>0.99</td>
<td>0.8</td>
<td>0.960</td>
</tr>
<tr>
<td>11</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.007</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
</tr>
<tr>
<td>12</td>
<td>0.99</td>
<td>0</td>
<td>0.988</td>
<td>0.004</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
</tr>
<tr>
<td>13</td>
<td>0.99</td>
<td>5.883</td>
<td>0.986</td>
<td>5.885</td>
<td>0.99</td>
<td>5.483</td>
<td>0.965</td>
</tr>
<tr>
<td>14</td>
<td>0.99</td>
<td>0.4</td>
<td>0.982</td>
<td>0.402</td>
<td>0.99</td>
<td>0.8</td>
<td>0.961</td>
</tr>
<tr>
<td>15</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.006</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
</tr>
<tr>
<td>16</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.009</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
</tr>
<tr>
<td>17</td>
<td>0.99</td>
<td>6.283</td>
<td>0.987</td>
<td>6.283</td>
<td>0.99</td>
<td>6.183</td>
<td>0.980</td>
</tr>
<tr>
<td>18</td>
<td>0.99</td>
<td>6.283</td>
<td>0.987</td>
<td>6.283</td>
<td>0.99</td>
<td>5.283</td>
<td>0.922</td>
</tr>
<tr>
<td>19</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.017</td>
<td>0.99</td>
<td>1</td>
<td>0.910</td>
</tr>
<tr>
<td>20</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.017</td>
<td>0.99</td>
<td>0.1</td>
<td>0.984</td>
</tr>
<tr>
<td>21</td>
<td>0.99</td>
<td>5.883</td>
<td>0.987</td>
<td>5.884</td>
<td>0.99</td>
<td>5.483</td>
<td>0.964</td>
</tr>
<tr>
<td>22</td>
<td>0.99</td>
<td>0.4</td>
<td>0.980</td>
<td>0.401</td>
<td>0.99</td>
<td>0.8</td>
<td>0.960</td>
</tr>
<tr>
<td>23</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.004</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
</tr>
<tr>
<td>24</td>
<td>0.99</td>
<td>6.283</td>
<td>0.990</td>
<td>6.283</td>
<td>0.99</td>
<td>6.183</td>
<td>0.990</td>
</tr>
<tr>
<td>25</td>
<td>0.99</td>
<td>5.883</td>
<td>0.985</td>
<td>5.900</td>
<td>0.99</td>
<td>5.883</td>
<td>0.985</td>
</tr>
<tr>
<td>26</td>
<td>0.99</td>
<td>6.183</td>
<td>0.989</td>
<td>6.190</td>
<td>0.99</td>
<td>5.683</td>
<td>0.975</td>
</tr>
<tr>
<td>27</td>
<td>0.99</td>
<td>5.883</td>
<td>0.961</td>
<td>5.890</td>
<td>0.99</td>
<td>5.783</td>
<td>0.981</td>
</tr>
<tr>
<td>28</td>
<td>0.99</td>
<td>0.7</td>
<td>0.949</td>
<td>0.709</td>
<td>0.99</td>
<td>0.5</td>
<td>0.980</td>
</tr>
<tr>
<td>29</td>
<td>0.99</td>
<td>0.1</td>
<td>0.990</td>
<td>0.105</td>
<td>0.99</td>
<td>0.6</td>
<td>0.971</td>
</tr>
<tr>
<td>30</td>
<td>0.99</td>
<td>0.3</td>
<td>0.987</td>
<td>0.305</td>
<td>0.99</td>
<td>0.4</td>
<td>0.985</td>
</tr>
<tr>
<td>31</td>
<td>0.99</td>
<td>0</td>
<td>0.990</td>
<td>0.006</td>
<td>0.99</td>
<td>0.1</td>
<td>0.991</td>
</tr>
<tr>
<td>32</td>
<td>0.99</td>
<td>0</td>
<td>0.991</td>
<td>0.001</td>
<td>0.99</td>
<td>0</td>
<td>0.990</td>
</tr>
<tr>
<td>33</td>
<td>0.99</td>
<td>5.883</td>
<td>0.989</td>
<td>5.884</td>
<td>0.99</td>
<td>5.483</td>
<td>0.969</td>
</tr>
<tr>
<td>34</td>
<td>0.99</td>
<td>0.4</td>
<td>0.983</td>
<td>0.401</td>
<td>0.99</td>
<td>0.8</td>
<td>0.965</td>
</tr>
<tr>
<td>35</td>
<td>0.99</td>
<td>0</td>
<td>0.988</td>
<td>0.003</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
</tr>
<tr>
<td>36</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.007</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
</tr>
<tr>
<td>37</td>
<td>0.99</td>
<td>6.283</td>
<td>0.989</td>
<td>6.283</td>
<td>0.99</td>
<td>6.183</td>
<td>0.983</td>
</tr>
<tr>
<td>38</td>
<td>0.99</td>
<td>6.283</td>
<td>0.989</td>
<td>6.283</td>
<td>0.99</td>
<td>5.283</td>
<td>0.936</td>
</tr>
<tr>
<td>39</td>
<td>0.99</td>
<td>0</td>
<td>0.986</td>
<td>0.013</td>
<td>0.99</td>
<td>1</td>
<td>0.926</td>
</tr>
<tr>
<td>40</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.013</td>
<td>0.99</td>
<td>1</td>
<td>0.926</td>
</tr>
<tr>
<td>$d$</td>
<td>ORR</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>41</td>
<td>0.99</td>
<td>0.986</td>
<td>0.013</td>
<td>0.99</td>
<td>0.1</td>
<td>0.982</td>
<td>0.101</td>
</tr>
<tr>
<td>42</td>
<td>0.99</td>
<td>5.883</td>
<td>5.884</td>
<td>0.99</td>
<td>5.483</td>
<td>0.964</td>
<td>5.484</td>
</tr>
<tr>
<td>43</td>
<td>0.99</td>
<td>0.4</td>
<td>0.401</td>
<td>0.99</td>
<td>0.8</td>
<td>0.959</td>
<td>0.800</td>
</tr>
<tr>
<td>44</td>
<td>0.99</td>
<td>0.987</td>
<td>0.002</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.002</td>
</tr>
<tr>
<td>45</td>
<td>0.99</td>
<td>0.988</td>
<td>0.000</td>
<td>0.99</td>
<td>0.988</td>
<td>0.000</td>
<td>0.001</td>
</tr>
<tr>
<td>46</td>
<td>0.99</td>
<td>5.883</td>
<td>5.884</td>
<td>0.99</td>
<td>5.483</td>
<td>0.967</td>
<td>5.484</td>
</tr>
<tr>
<td>47</td>
<td>0.99</td>
<td>0.4</td>
<td>0.400</td>
<td>0.99</td>
<td>0.8</td>
<td>0.962</td>
<td>0.800</td>
</tr>
<tr>
<td>48</td>
<td>0.99</td>
<td>0.987</td>
<td>0.001</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.001</td>
</tr>
<tr>
<td>49</td>
<td>0.99</td>
<td>0.987</td>
<td>0.004</td>
<td>0.99</td>
<td>0.987</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>50</td>
<td>0.99</td>
<td>6.283</td>
<td>6.283</td>
<td>0.99</td>
<td>6.183</td>
<td>0.977</td>
<td>6.184</td>
</tr>
<tr>
<td>51</td>
<td>0.99</td>
<td>6.283</td>
<td>6.283</td>
<td>0.99</td>
<td>5.283</td>
<td>0.965</td>
<td>5.284</td>
</tr>
<tr>
<td>52</td>
<td>0.99</td>
<td>0.986</td>
<td>0.007</td>
<td>0.99</td>
<td>1</td>
<td>0.965</td>
<td>1.001</td>
</tr>
<tr>
<td>53</td>
<td>0.99</td>
<td>0.986</td>
<td>0.007</td>
<td>0.99</td>
<td>0.1</td>
<td>0.980</td>
<td>0.100</td>
</tr>
<tr>
<td>54</td>
<td>0.99</td>
<td>5.883</td>
<td>5.884</td>
<td>0.99</td>
<td>5.483</td>
<td>0.966</td>
<td>5.484</td>
</tr>
<tr>
<td>55</td>
<td>0.99</td>
<td>0.4</td>
<td>0.400</td>
<td>0.99</td>
<td>0.8</td>
<td>0.961</td>
<td>0.800</td>
</tr>
<tr>
<td>56</td>
<td>0.99</td>
<td>0.987</td>
<td>0.001</td>
<td>0.99</td>
<td>0</td>
<td>0.987</td>
<td>0.001</td>
</tr>
<tr>
<td>57</td>
<td>0.99</td>
<td>6.283</td>
<td>6.283</td>
<td>0.99</td>
<td>6.183</td>
<td>0.991</td>
<td>6.185</td>
</tr>
<tr>
<td>58</td>
<td>0.99</td>
<td>5.983</td>
<td>5.985</td>
<td>0.99</td>
<td>5.883</td>
<td>0.987</td>
<td>5.885</td>
</tr>
<tr>
<td>59</td>
<td>0.99</td>
<td>6.183</td>
<td>6.185</td>
<td>0.99</td>
<td>5.783</td>
<td>0.982</td>
<td>5.785</td>
</tr>
<tr>
<td>60</td>
<td>0.99</td>
<td>5.983</td>
<td>5.985</td>
<td>0.99</td>
<td>5.783</td>
<td>0.977</td>
<td>5.785</td>
</tr>
<tr>
<td>61</td>
<td>0.99</td>
<td>0.7</td>
<td>0.702</td>
<td>0.99</td>
<td>0.5</td>
<td>0.977</td>
<td>0.502</td>
</tr>
<tr>
<td>62</td>
<td>0.99</td>
<td>0.1</td>
<td>0.101</td>
<td>0.99</td>
<td>0.6</td>
<td>0.970</td>
<td>0.602</td>
</tr>
<tr>
<td>63</td>
<td>0.99</td>
<td>0.3</td>
<td>0.301</td>
<td>0.99</td>
<td>0.4</td>
<td>0.981</td>
<td>0.402</td>
</tr>
<tr>
<td>64</td>
<td>0.99</td>
<td>0.0</td>
<td>0.002</td>
<td>0.99</td>
<td>0.1</td>
<td>0.989</td>
<td>0.101</td>
</tr>
<tr>
<td>65</td>
<td>0.99</td>
<td>5.983</td>
<td>5.996</td>
<td>0.99</td>
<td>6.083</td>
<td>0.988</td>
<td>6.089</td>
</tr>
<tr>
<td>66</td>
<td>0.99</td>
<td>6.183</td>
<td>6.195</td>
<td>0.99</td>
<td>6.083</td>
<td>0.988</td>
<td>6.089</td>
</tr>
<tr>
<td>67</td>
<td>0.99</td>
<td>6.283</td>
<td>6.283</td>
<td>0.99</td>
<td>6.183</td>
<td>0.989</td>
<td>6.189</td>
</tr>
<tr>
<td>68</td>
<td>0.99</td>
<td>6.083</td>
<td>6.096</td>
<td>0.99</td>
<td>5.883</td>
<td>0.985</td>
<td>5.890</td>
</tr>
<tr>
<td>69</td>
<td>0.99</td>
<td>6.283</td>
<td>6.283</td>
<td>0.99</td>
<td>5.883</td>
<td>0.985</td>
<td>5.890</td>
</tr>
<tr>
<td>70</td>
<td>0.99</td>
<td>6.183</td>
<td>6.195</td>
<td>0.99</td>
<td>5.783</td>
<td>0.980</td>
<td>5.789</td>
</tr>
<tr>
<td>71</td>
<td>0.99</td>
<td>6.083</td>
<td>6.096</td>
<td>0.99</td>
<td>5.883</td>
<td>0.985</td>
<td>5.890</td>
</tr>
<tr>
<td>72</td>
<td>0.99</td>
<td>6.283</td>
<td>6.283</td>
<td>0.99</td>
<td>5.983</td>
<td>0.986</td>
<td>5.990</td>
</tr>
<tr>
<td>73</td>
<td>0.99</td>
<td>0.989</td>
<td>0.011</td>
<td>0.99</td>
<td>0.3</td>
<td>0.986</td>
<td>0.304</td>
</tr>
<tr>
<td>74</td>
<td>0.99</td>
<td>0.2</td>
<td>0.210</td>
<td>0.99</td>
<td>0.4</td>
<td>0.980</td>
<td>0.405</td>
</tr>
<tr>
<td>75</td>
<td>0.99</td>
<td>0.1</td>
<td>0.110</td>
<td>0.99</td>
<td>0.5</td>
<td>0.973</td>
<td>0.506</td>
</tr>
<tr>
<td>76</td>
<td>0.99</td>
<td>0.989</td>
<td>0.011</td>
<td>0.99</td>
<td>0.4</td>
<td>0.980</td>
<td>0.405</td>
</tr>
<tr>
<td>77</td>
<td>0.99</td>
<td>0.2</td>
<td>0.210</td>
<td>0.99</td>
<td>0.4</td>
<td>0.980</td>
<td>0.405</td>
</tr>
<tr>
<td>78</td>
<td>0.99</td>
<td>0.989</td>
<td>0.011</td>
<td>0.99</td>
<td>0.1</td>
<td>0.990</td>
<td>0.105</td>
</tr>
<tr>
<td>79</td>
<td>0.99</td>
<td>0.1</td>
<td>0.110</td>
<td>0.99</td>
<td>0.2</td>
<td>0.989</td>
<td>0.204</td>
</tr>
<tr>
<td>80</td>
<td>0.99</td>
<td>0.3</td>
<td>0.311</td>
<td>0.99</td>
<td>0.2</td>
<td>0.989</td>
<td>0.204</td>
</tr>
</tbody>
</table>
Table A-10: Initial and optimum parameters for $32 \times 1$ OBFNs. $\text{Nexamples} = 20/20k$, $d = 2$, $\text{Nfreq} = 3/20$, learning rate = $2 \times 10^{-5}$, momentum = 0.7, iteration = 10000.

<table>
<thead>
<tr>
<th>d</th>
<th>ORR</th>
<th>Best case</th>
<th></th>
<th></th>
<th>Worst case</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
</tr>
<tr>
<td>1</td>
<td>0.59</td>
<td>0.791</td>
<td>0.009</td>
<td>0.59</td>
<td>0.791</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.59</td>
<td>5.783</td>
<td>0.632</td>
<td>5.802</td>
<td>0.59</td>
<td>5.683</td>
<td>0.629</td>
</tr>
<tr>
<td>3</td>
<td>0.59</td>
<td>0.5</td>
<td>0.631</td>
<td>0.440</td>
<td>0.59</td>
<td>0.6</td>
<td>0.628</td>
</tr>
<tr>
<td>4</td>
<td>0.59</td>
<td>0</td>
<td>0.788</td>
<td>0.009</td>
<td>0.59</td>
<td>0</td>
<td>0.787</td>
</tr>
<tr>
<td>5</td>
<td>0.59</td>
<td>0</td>
<td>0.796</td>
<td>0.009</td>
<td>0.59</td>
<td>0</td>
<td>0.804</td>
</tr>
<tr>
<td>6</td>
<td>0.59</td>
<td>5.983</td>
<td>0.595</td>
<td>6.000</td>
<td>0.59</td>
<td>5.883</td>
<td>0.673</td>
</tr>
<tr>
<td>7</td>
<td>0.59</td>
<td>5.683</td>
<td>0.681</td>
<td>5.690</td>
<td>0.59</td>
<td>5.883</td>
<td>0.673</td>
</tr>
<tr>
<td>8</td>
<td>0.59</td>
<td>0.6</td>
<td>0.648</td>
<td>0.603</td>
<td>0.59</td>
<td>0.4</td>
<td>0.674</td>
</tr>
<tr>
<td>9</td>
<td>0.59</td>
<td>0.3</td>
<td>0.607</td>
<td>0.290</td>
<td>0.59</td>
<td>0.4</td>
<td>0.674</td>
</tr>
<tr>
<td>10</td>
<td>0.59</td>
<td>5.783</td>
<td>0.604</td>
<td>5.835</td>
<td>0.59</td>
<td>5.683</td>
<td>0.585</td>
</tr>
<tr>
<td>11</td>
<td>0.59</td>
<td>0.5</td>
<td>0.602</td>
<td>0.464</td>
<td>0.59</td>
<td>0.6</td>
<td>0.580</td>
</tr>
<tr>
<td>12</td>
<td>0.59</td>
<td>0</td>
<td>0.780</td>
<td>0.009</td>
<td>0.59</td>
<td>0</td>
<td>0.766</td>
</tr>
<tr>
<td>13</td>
<td>0.59</td>
<td>0</td>
<td>0.806</td>
<td>0.006</td>
<td>0.59</td>
<td>0</td>
<td>0.838</td>
</tr>
<tr>
<td>14</td>
<td>0.59</td>
<td>5.783</td>
<td>0.579</td>
<td>5.811</td>
<td>0.59</td>
<td>5.683</td>
<td>0.587</td>
</tr>
<tr>
<td>15</td>
<td>0.59</td>
<td>0.5</td>
<td>0.571</td>
<td>0.483</td>
<td>0.59</td>
<td>0.6</td>
<td>0.575</td>
</tr>
<tr>
<td>16</td>
<td>0.59</td>
<td>0</td>
<td>0.769</td>
<td>0.006</td>
<td>0.59</td>
<td>0</td>
<td>0.718</td>
</tr>
<tr>
<td>17</td>
<td>0.59</td>
<td>0</td>
<td>0.809</td>
<td>0.004</td>
<td>0.59</td>
<td>0</td>
<td>0.856</td>
</tr>
<tr>
<td>18</td>
<td>0.59</td>
<td>5.983</td>
<td>0.585</td>
<td>5.988</td>
<td>0.59</td>
<td>5.883</td>
<td>0.659</td>
</tr>
<tr>
<td>19</td>
<td>0.59</td>
<td>5.683</td>
<td>0.636</td>
<td>5.687</td>
<td>0.59</td>
<td>5.883</td>
<td>0.659</td>
</tr>
<tr>
<td>20</td>
<td>0.59</td>
<td>0.6</td>
<td>0.595</td>
<td>0.604</td>
<td>0.59</td>
<td>0.4</td>
<td>0.653</td>
</tr>
<tr>
<td>21</td>
<td>0.59</td>
<td>0.3</td>
<td>0.602</td>
<td>0.300</td>
<td>0.59</td>
<td>0.4</td>
<td>0.653</td>
</tr>
<tr>
<td>22</td>
<td>0.59</td>
<td>5.783</td>
<td>0.592</td>
<td>5.796</td>
<td>0.59</td>
<td>5.683</td>
<td>0.659</td>
</tr>
<tr>
<td>23</td>
<td>0.59</td>
<td>0.5</td>
<td>0.579</td>
<td>0.493</td>
<td>0.59</td>
<td>0.6</td>
<td>0.598</td>
</tr>
<tr>
<td>24</td>
<td>0.59</td>
<td>0</td>
<td>0.748</td>
<td>0.004</td>
<td>0.59</td>
<td>0</td>
<td>0.682</td>
</tr>
<tr>
<td>25</td>
<td>0.59</td>
<td>5.783</td>
<td>0.558</td>
<td>5.791</td>
<td>0.59</td>
<td>5.883</td>
<td>0.623</td>
</tr>
<tr>
<td>26</td>
<td>0.59</td>
<td>6.183</td>
<td>0.556</td>
<td>6.186</td>
<td>0.59</td>
<td>5.783</td>
<td>0.612</td>
</tr>
<tr>
<td>27</td>
<td>0.59</td>
<td>5.683</td>
<td>0.612</td>
<td>5.691</td>
<td>0.59</td>
<td>6.183</td>
<td>0.630</td>
</tr>
<tr>
<td>28</td>
<td>0.59</td>
<td>5.683</td>
<td>0.612</td>
<td>5.691</td>
<td>0.59</td>
<td>5.683</td>
<td>0.621</td>
</tr>
<tr>
<td>29</td>
<td>0.59</td>
<td>0.6</td>
<td>0.573</td>
<td>0.606</td>
<td>0.59</td>
<td>0.6</td>
<td>0.551</td>
</tr>
<tr>
<td>30</td>
<td>0.59</td>
<td>0.6</td>
<td>0.573</td>
<td>0.606</td>
<td>0.59</td>
<td>0.1</td>
<td>0.629</td>
</tr>
<tr>
<td>31</td>
<td>0.59</td>
<td>0.1</td>
<td>0.549</td>
<td>0.103</td>
<td>0.59</td>
<td>0.5</td>
<td>0.615</td>
</tr>
<tr>
<td>32</td>
<td>0.59</td>
<td>0.5</td>
<td>0.600</td>
<td>0.505</td>
<td>0.59</td>
<td>0.4</td>
<td>0.656</td>
</tr>
<tr>
<td>33</td>
<td>0.59</td>
<td>0</td>
<td>0.785</td>
<td>0.001</td>
<td>0.59</td>
<td>0</td>
<td>0.816</td>
</tr>
<tr>
<td>34</td>
<td>0.59</td>
<td>5.783</td>
<td>0.590</td>
<td>5.789</td>
<td>0.59</td>
<td>5.683</td>
<td>0.609</td>
</tr>
<tr>
<td>35</td>
<td>0.59</td>
<td>0.5</td>
<td>0.574</td>
<td>0.497</td>
<td>0.59</td>
<td>0.6</td>
<td>0.595</td>
</tr>
<tr>
<td>36</td>
<td>0.59</td>
<td>0</td>
<td>0.708</td>
<td>0.002</td>
<td>0.59</td>
<td>0</td>
<td>0.634</td>
</tr>
<tr>
<td>37</td>
<td>0.59</td>
<td>0</td>
<td>0.721</td>
<td>0.001</td>
<td>0.59</td>
<td>0</td>
<td>0.772</td>
</tr>
<tr>
<td>38</td>
<td>0.59</td>
<td>5.983</td>
<td>0.609</td>
<td>5.984</td>
<td>0.59</td>
<td>5.883</td>
<td>0.671</td>
</tr>
<tr>
<td>39</td>
<td>0.59</td>
<td>5.683</td>
<td>0.612</td>
<td>5.684</td>
<td>0.59</td>
<td>5.883</td>
<td>0.671</td>
</tr>
<tr>
<td>40</td>
<td>0.59</td>
<td>0.6</td>
<td>0.576</td>
<td>0.603</td>
<td>0.59</td>
<td>0.4</td>
<td>0.655</td>
</tr>
<tr>
<td>41</td>
<td>0.59</td>
<td>0.3</td>
<td>0.612</td>
<td>0.302</td>
<td>0.59</td>
<td>0.4</td>
<td>0.655</td>
</tr>
<tr>
<td>42</td>
<td>0.59</td>
<td>5.783</td>
<td>0.598</td>
<td>5.786</td>
<td>0.59</td>
<td>5.683</td>
<td>0.607</td>
</tr>
</tbody>
</table>
Table A-10 – continued from previous page

<table>
<thead>
<tr>
<th>d</th>
<th>ORR</th>
<th>Best case</th>
<th>Worst case</th>
<th>$\kappa$</th>
<th>$\phi$</th>
<th>$\kappa$</th>
<th>$\phi$</th>
<th>$E_{NSGD}$</th>
<th>$\kappa$</th>
<th>$\phi$</th>
<th>$\kappa$</th>
<th>$\phi$</th>
<th>$E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>0.59</td>
<td>0.5</td>
<td>0.581</td>
<td>0.500</td>
<td>0.59</td>
<td>0.6</td>
<td>0.588</td>
<td>0.508</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>0.59</td>
<td>0</td>
<td>0.651</td>
<td>0.001</td>
<td>0.59</td>
<td>0</td>
<td>0.625</td>
<td>0.001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>0.59</td>
<td>0</td>
<td>0.651</td>
<td>0.000</td>
<td>0.59</td>
<td>0.6</td>
<td>0.673</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>0.59</td>
<td>5.783</td>
<td>0.591</td>
<td>5.785</td>
<td>0.59</td>
<td>5.683</td>
<td>0.600</td>
<td>5.686</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>0.59</td>
<td>0.5</td>
<td>0.581</td>
<td>0.500</td>
<td>0.59</td>
<td>0.6</td>
<td>0.586</td>
<td>0.599</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>0.59</td>
<td>0</td>
<td>0.612</td>
<td>0.000</td>
<td>0.59</td>
<td>0</td>
<td>0.597</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>0.59</td>
<td>0</td>
<td>0.616</td>
<td>0.000</td>
<td>0.59</td>
<td>0</td>
<td>0.635</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0.59</td>
<td>5.983</td>
<td>0.596</td>
<td>5.984</td>
<td>0.59</td>
<td>5.883</td>
<td>0.627</td>
<td>5.882</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>0.59</td>
<td>5.683</td>
<td>0.603</td>
<td>5.684</td>
<td>0.59</td>
<td>5.883</td>
<td>0.627</td>
<td>5.882</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>0.59</td>
<td>0.6</td>
<td>0.577</td>
<td>0.601</td>
<td>0.59</td>
<td>0.4</td>
<td>0.611</td>
<td>0.403</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>0.59</td>
<td>0.3</td>
<td>0.588</td>
<td>0.301</td>
<td>0.59</td>
<td>0.4</td>
<td>0.611</td>
<td>0.403</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>0.59</td>
<td>5.783</td>
<td>0.595</td>
<td>5.784</td>
<td>0.59</td>
<td>5.683</td>
<td>0.599</td>
<td>5.684</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>0.59</td>
<td>0.5</td>
<td>0.583</td>
<td>0.500</td>
<td>0.59</td>
<td>0.6</td>
<td>0.583</td>
<td>0.601</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>0.59</td>
<td>0</td>
<td>0.599</td>
<td>0.000</td>
<td>0.59</td>
<td>0</td>
<td>0.602</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>0.59</td>
<td>5.783</td>
<td>0.595</td>
<td>5.785</td>
<td>0.59</td>
<td>5.883</td>
<td>0.637</td>
<td>5.882</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>0.59</td>
<td>6.183</td>
<td>0.575</td>
<td>6.184</td>
<td>0.59</td>
<td>5.783</td>
<td>0.620</td>
<td>5.782</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>0.59</td>
<td>5.683</td>
<td>0.605</td>
<td>5.685</td>
<td>0.59</td>
<td>6.183</td>
<td>0.679</td>
<td>6.184</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>0.59</td>
<td>5.683</td>
<td>0.605</td>
<td>5.685</td>
<td>0.59</td>
<td>5.683</td>
<td>0.607</td>
<td>5.682</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>0.59</td>
<td>0.6</td>
<td>0.574</td>
<td>0.601</td>
<td>0.59</td>
<td>0.6</td>
<td>0.567</td>
<td>0.605</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>0.59</td>
<td>0.6</td>
<td>0.574</td>
<td>0.601</td>
<td>0.59</td>
<td>0.1</td>
<td>0.682</td>
<td>0.102</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>0.59</td>
<td>0.1</td>
<td>0.582</td>
<td>0.101</td>
<td>0.59</td>
<td>0.5</td>
<td>0.596</td>
<td>0.505</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>0.59</td>
<td>0.5</td>
<td>0.580</td>
<td>0.301</td>
<td>0.59</td>
<td>0.4</td>
<td>0.628</td>
<td>0.405</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>0.59</td>
<td>5.783</td>
<td>0.597</td>
<td>5.787</td>
<td>0.59</td>
<td>5.883</td>
<td>0.595</td>
<td>5.888</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>0.59</td>
<td>5.683</td>
<td>0.617</td>
<td>5.687</td>
<td>0.59</td>
<td>6.283</td>
<td>0.532</td>
<td>6.283</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>0.59</td>
<td>5.683</td>
<td>0.617</td>
<td>5.687</td>
<td>0.59</td>
<td>5.683</td>
<td>0.629</td>
<td>5.687</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>0.59</td>
<td>5.883</td>
<td>0.579</td>
<td>5.887</td>
<td>0.59</td>
<td>5.783</td>
<td>0.614</td>
<td>5.787</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>0.59</td>
<td>5.683</td>
<td>0.617</td>
<td>5.687</td>
<td>0.59</td>
<td>5.683</td>
<td>0.629</td>
<td>5.687</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>0.59</td>
<td>5.683</td>
<td>0.617</td>
<td>5.687</td>
<td>0.59</td>
<td>5.683</td>
<td>0.629</td>
<td>5.687</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>0.59</td>
<td>6.083</td>
<td>0.564</td>
<td>6.086</td>
<td>0.59</td>
<td>5.983</td>
<td>0.575</td>
<td>5.988</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>0.59</td>
<td>6.283</td>
<td>0.564</td>
<td>6.283</td>
<td>0.59</td>
<td>5.783</td>
<td>0.614</td>
<td>5.787</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>0.59</td>
<td>0</td>
<td>0.654</td>
<td>0.002</td>
<td>0.59</td>
<td>0.5</td>
<td>0.608</td>
<td>0.504</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>0.59</td>
<td>0.2</td>
<td>0.597</td>
<td>0.201</td>
<td>0.59</td>
<td>0.3</td>
<td>0.623</td>
<td>0.302</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>0.59</td>
<td>0.6</td>
<td>0.575</td>
<td>0.604</td>
<td>0.59</td>
<td>0.6</td>
<td>0.577</td>
<td>0.605</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>0.59</td>
<td>0.6</td>
<td>0.575</td>
<td>0.604</td>
<td>0.59</td>
<td>0.6</td>
<td>0.577</td>
<td>0.605</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>0.59</td>
<td>0.4</td>
<td>0.615</td>
<td>0.403</td>
<td>0.59</td>
<td>0.5</td>
<td>0.608</td>
<td>0.504</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>0.59</td>
<td>0.6</td>
<td>0.575</td>
<td>0.604</td>
<td>0.59</td>
<td>0.6</td>
<td>0.577</td>
<td>0.605</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>0.59</td>
<td>0.6</td>
<td>0.575</td>
<td>0.604</td>
<td>0.59</td>
<td>0</td>
<td>0.543</td>
<td>0.002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>0.59</td>
<td>0.5</td>
<td>0.600</td>
<td>0.504</td>
<td>0.59</td>
<td>0.4</td>
<td>0.626</td>
<td>0.403</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table A-11: Initial and optimum parameters for $32 \times 1$ OBFNs. $N_{\text{examples}} = 20/20k$, $d = 3$, $N_{\text{freq}} = 3/20$, learning rate $= 2 \times 10^{-5}$, momentum $= 0.7$, iteration $= 10000$.  

<table>
<thead>
<tr>
<th>$d$</th>
<th>ORR</th>
<th>Initial</th>
<th>Optimum</th>
<th>$E_{NSGD}$</th>
<th>Initial</th>
<th>Optimum</th>
<th>$E_{NSGD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>1</td>
<td>0.39</td>
<td>0.598</td>
<td>0.009</td>
<td>0.39</td>
<td>0.595</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.39</td>
<td>5.883</td>
<td>0.436</td>
<td>5.988</td>
<td>0.39</td>
<td>5.883</td>
<td>0.429</td>
</tr>
<tr>
<td>3</td>
<td>0.39</td>
<td>0.4</td>
<td>0.344</td>
<td>0.314</td>
<td>0.39</td>
<td>0.4</td>
<td>0.342</td>
</tr>
<tr>
<td>4</td>
<td>0.39</td>
<td>0</td>
<td>0.591</td>
<td>0.009</td>
<td>0.39</td>
<td>0</td>
<td>0.589</td>
</tr>
<tr>
<td>5</td>
<td>0.39</td>
<td>0</td>
<td>0.606</td>
<td>0.010</td>
<td>0.39</td>
<td>0</td>
<td>0.607</td>
</tr>
<tr>
<td>6</td>
<td>0.39</td>
<td>5.983</td>
<td>0.360</td>
<td>6.123</td>
<td>0.39</td>
<td>5.983</td>
<td>0.378</td>
</tr>
<tr>
<td>7</td>
<td>0.39</td>
<td>5.683</td>
<td>0.306</td>
<td>5.815</td>
<td>0.39</td>
<td>5.783</td>
<td>0.349</td>
</tr>
<tr>
<td>8</td>
<td>0.39</td>
<td>0.7</td>
<td>0.297</td>
<td>0.488</td>
<td>0.39</td>
<td>0.5</td>
<td>0.339</td>
</tr>
<tr>
<td>9</td>
<td>0.39</td>
<td>0</td>
<td>0.353</td>
<td>0.170</td>
<td>0.39</td>
<td>0</td>
<td>0.375</td>
</tr>
<tr>
<td>10</td>
<td>0.39</td>
<td>5.883</td>
<td>0.432</td>
<td>5.977</td>
<td>0.39</td>
<td>5.883</td>
<td>0.429</td>
</tr>
<tr>
<td>11</td>
<td>0.39</td>
<td>0</td>
<td>0.381</td>
<td>0.009</td>
<td>0.39</td>
<td>0</td>
<td>0.553</td>
</tr>
<tr>
<td>12</td>
<td>0.39</td>
<td>0</td>
<td>0.602</td>
<td>0.005</td>
<td>0.39</td>
<td>0</td>
<td>0.641</td>
</tr>
<tr>
<td>13</td>
<td>0.39</td>
<td>0</td>
<td>0.421</td>
<td>0.521</td>
<td>0.39</td>
<td>0.4</td>
<td>0.381</td>
</tr>
<tr>
<td>14</td>
<td>0.39</td>
<td>0</td>
<td>0.417</td>
<td>0.376</td>
<td>0.39</td>
<td>0</td>
<td>0.501</td>
</tr>
<tr>
<td>15</td>
<td>0.39</td>
<td>0</td>
<td>0.530</td>
<td>0.005</td>
<td>0.39</td>
<td>0</td>
<td>0.469</td>
</tr>
<tr>
<td>16</td>
<td>0.39</td>
<td>0</td>
<td>0.498</td>
<td>0.003</td>
<td>0.39</td>
<td>0</td>
<td>0.606</td>
</tr>
<tr>
<td>17</td>
<td>0.39</td>
<td>0</td>
<td>0.598</td>
<td>0.615</td>
<td>0.39</td>
<td>0</td>
<td>0.573</td>
</tr>
<tr>
<td>18</td>
<td>0.39</td>
<td>0</td>
<td>0.7</td>
<td>0.669</td>
<td>0.39</td>
<td>0</td>
<td>0.425</td>
</tr>
<tr>
<td>19</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.232</td>
<td>7.99E-2</td>
<td>0.39</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.234</td>
<td>0.39</td>
<td>0</td>
<td>0.389</td>
</tr>
<tr>
<td>21</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.322</td>
<td>0.39</td>
<td>0</td>
<td>0.377</td>
</tr>
<tr>
<td>22</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.324</td>
<td>0.39</td>
<td>0</td>
<td>0.364</td>
</tr>
<tr>
<td>23</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.332</td>
<td>0.39</td>
<td>0</td>
<td>0.398</td>
</tr>
<tr>
<td>24</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.398</td>
<td>0.39</td>
<td>0</td>
<td>0.583</td>
</tr>
<tr>
<td>25</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.328</td>
<td>0.39</td>
<td>0</td>
<td>0.398</td>
</tr>
<tr>
<td>26</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.326</td>
<td>0.39</td>
<td>0</td>
<td>0.283</td>
</tr>
<tr>
<td>27</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.328</td>
<td>0.39</td>
<td>0</td>
<td>0.315</td>
</tr>
<tr>
<td>28</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.317</td>
<td>0.39</td>
<td>0</td>
<td>0.398</td>
</tr>
<tr>
<td>29</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.514</td>
<td>0.39</td>
<td>0</td>
<td>0.391</td>
</tr>
<tr>
<td>30</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.325</td>
<td>0.39</td>
<td>0</td>
<td>0.398</td>
</tr>
<tr>
<td>31</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.314</td>
<td>0.39</td>
<td>0</td>
<td>0.391</td>
</tr>
<tr>
<td>32</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.325</td>
<td>0.39</td>
<td>0</td>
<td>0.391</td>
</tr>
<tr>
<td>33</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.338</td>
<td>0.39</td>
<td>0</td>
<td>0.398</td>
</tr>
<tr>
<td>34</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.397</td>
<td>0.39</td>
<td>0</td>
<td>0.384</td>
</tr>
<tr>
<td>35</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.407</td>
<td>0.39</td>
<td>0</td>
<td>0.400</td>
</tr>
<tr>
<td>36</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.350</td>
<td>0.39</td>
<td>0</td>
<td>0.384</td>
</tr>
<tr>
<td>37</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.249</td>
<td>0.39</td>
<td>0</td>
<td>0.359</td>
</tr>
<tr>
<td>38</td>
<td>0.39</td>
<td>0</td>
<td>0</td>
<td>0.474</td>
<td>0.39</td>
<td>0</td>
<td>0.573</td>
</tr>
<tr>
<td>39</td>
<td>0.39</td>
<td>0</td>
<td>0.7</td>
<td>0.435</td>
<td>0.39</td>
<td>0</td>
<td>0.380</td>
</tr>
<tr>
<td>40</td>
<td>0.39</td>
<td>0</td>
<td>0.250</td>
<td>0.289</td>
<td>0.39</td>
<td>0</td>
<td>0.359</td>
</tr>
<tr>
<td>41</td>
<td>0.39</td>
<td>0</td>
<td>0.364</td>
<td>5.888</td>
<td>0.39</td>
<td>0</td>
<td>0.386</td>
</tr>
<tr>
<td>$d$</td>
<td>ORR</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
<td>$\kappa$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>43</td>
<td>0.39</td>
<td>0.4</td>
<td>0.362</td>
<td>0.396</td>
<td>0.39</td>
<td>0.383</td>
<td>0.399</td>
</tr>
<tr>
<td>44</td>
<td>0.39</td>
<td>0</td>
<td>0.358</td>
<td>0.000</td>
<td>0.39</td>
<td>0</td>
<td>0.382</td>
</tr>
<tr>
<td>45</td>
<td>0.39</td>
<td>0</td>
<td>0.403</td>
<td>0.000</td>
<td>0.39</td>
<td>0</td>
<td>0.393</td>
</tr>
<tr>
<td>46</td>
<td>0.39</td>
<td>5.883</td>
<td>0.394</td>
<td>5.883</td>
<td>0.39</td>
<td>5.883</td>
<td>0.391</td>
</tr>
<tr>
<td>47</td>
<td>0.39</td>
<td>0.4</td>
<td>0.389</td>
<td>0.401</td>
<td>0.39</td>
<td>0.4</td>
<td>0.388</td>
</tr>
<tr>
<td>48</td>
<td>0.39</td>
<td>0</td>
<td>0.394</td>
<td>0.000</td>
<td>0.39</td>
<td>0</td>
<td>0.390</td>
</tr>
<tr>
<td>49</td>
<td>0.39</td>
<td>0</td>
<td>0.387</td>
<td>0.000</td>
<td>0.39</td>
<td>0</td>
<td>0.389</td>
</tr>
<tr>
<td>50</td>
<td>0.39</td>
<td>5.983</td>
<td>0.382</td>
<td>5.984</td>
<td>0.39</td>
<td>5.983</td>
<td>0.388</td>
</tr>
<tr>
<td>51</td>
<td>0.39</td>
<td>5.583</td>
<td>0.398</td>
<td>5.584</td>
<td>0.39</td>
<td>5.783</td>
<td>0.391</td>
</tr>
<tr>
<td>52</td>
<td>0.39</td>
<td>0.7</td>
<td>0.392</td>
<td>0.700</td>
<td>0.39</td>
<td>0.5</td>
<td>0.389</td>
</tr>
<tr>
<td>53</td>
<td>0.39</td>
<td>0.3</td>
<td>0.379</td>
<td>0.300</td>
<td>0.39</td>
<td>0.3</td>
<td>0.387</td>
</tr>
<tr>
<td>54</td>
<td>0.39</td>
<td>5.883</td>
<td>0.388</td>
<td>5.883</td>
<td>0.39</td>
<td>5.883</td>
<td>0.390</td>
</tr>
<tr>
<td>55</td>
<td>0.39</td>
<td>0.4</td>
<td>0.387</td>
<td>0.400</td>
<td>0.39</td>
<td>0.4</td>
<td>0.389</td>
</tr>
<tr>
<td>56</td>
<td>0.39</td>
<td>0</td>
<td>0.388</td>
<td>0.000</td>
<td>0.39</td>
<td>0</td>
<td>0.389</td>
</tr>
<tr>
<td>57</td>
<td>0.39</td>
<td>5.883</td>
<td>0.398</td>
<td>5.884</td>
<td>0.39</td>
<td>5.883</td>
<td>0.392</td>
</tr>
<tr>
<td>58</td>
<td>0.39</td>
<td>6.183</td>
<td>0.402</td>
<td>6.184</td>
<td>0.39</td>
<td>6.183</td>
<td>0.392</td>
</tr>
<tr>
<td>59</td>
<td>0.39</td>
<td>5.983</td>
<td>0.398</td>
<td>5.884</td>
<td>0.39</td>
<td>5.983</td>
<td>0.395</td>
</tr>
<tr>
<td>60</td>
<td>0.39</td>
<td>5.983</td>
<td>0.401</td>
<td>5.984</td>
<td>0.39</td>
<td>5.983</td>
<td>0.392</td>
</tr>
<tr>
<td>61</td>
<td>0.39</td>
<td>0.3</td>
<td>0.391</td>
<td>0.302</td>
<td>0.39</td>
<td>0.4</td>
<td>0.385</td>
</tr>
<tr>
<td>62</td>
<td>0.39</td>
<td>0.4</td>
<td>0.384</td>
<td>0.402</td>
<td>0.39</td>
<td>0.4</td>
<td>0.385</td>
</tr>
<tr>
<td>63</td>
<td>0.39</td>
<td>0.1</td>
<td>0.398</td>
<td>0.101</td>
<td>0.39</td>
<td>0.4</td>
<td>0.385</td>
</tr>
<tr>
<td>64</td>
<td>0.39</td>
<td>0.4</td>
<td>0.384</td>
<td>0.402</td>
<td>0.39</td>
<td>0.4</td>
<td>0.385</td>
</tr>
<tr>
<td>65</td>
<td>0.39</td>
<td>6.183</td>
<td>0.387</td>
<td>6.188</td>
<td>0.39</td>
<td>6.183</td>
<td>0.373</td>
</tr>
<tr>
<td>66</td>
<td>0.39</td>
<td>5.783</td>
<td>0.370</td>
<td>5.792</td>
<td>0.39</td>
<td>5.883</td>
<td>0.385</td>
</tr>
<tr>
<td>67</td>
<td>0.39</td>
<td>5.783</td>
<td>0.370</td>
<td>5.792</td>
<td>0.39</td>
<td>6.183</td>
<td>0.373</td>
</tr>
<tr>
<td>68</td>
<td>0.39</td>
<td>5.883</td>
<td>0.337</td>
<td>5.888</td>
<td>0.39</td>
<td>5.883</td>
<td>0.385</td>
</tr>
<tr>
<td>69</td>
<td>0.39</td>
<td>6.083</td>
<td>0.386</td>
<td>6.087</td>
<td>0.39</td>
<td>5.783</td>
<td>0.419</td>
</tr>
<tr>
<td>70</td>
<td>0.39</td>
<td>5.883</td>
<td>0.337</td>
<td>5.888</td>
<td>0.39</td>
<td>5.883</td>
<td>0.385</td>
</tr>
<tr>
<td>71</td>
<td>0.39</td>
<td>6.283</td>
<td>0.381</td>
<td>6.283</td>
<td>0.39</td>
<td>6.083</td>
<td>0.375</td>
</tr>
<tr>
<td>72</td>
<td>0.39</td>
<td>5.983</td>
<td>0.369</td>
<td>5.987</td>
<td>0.39</td>
<td>5.883</td>
<td>0.385</td>
</tr>
<tr>
<td>73</td>
<td>0.39</td>
<td>0.3</td>
<td>0.379</td>
<td>0.306</td>
<td>0.39</td>
<td>0.4</td>
<td>0.375</td>
</tr>
<tr>
<td>74</td>
<td>0.39</td>
<td>0</td>
<td>0.381</td>
<td>0.005</td>
<td>0.39</td>
<td>0.2</td>
<td>0.392</td>
</tr>
<tr>
<td>75</td>
<td>0.39</td>
<td>0.4</td>
<td>0.350</td>
<td>0.407</td>
<td>0.39</td>
<td>0.4</td>
<td>0.375</td>
</tr>
<tr>
<td>76</td>
<td>0.39</td>
<td>0.2</td>
<td>0.380</td>
<td>0.205</td>
<td>0.39</td>
<td>0.5</td>
<td>0.336</td>
</tr>
<tr>
<td>77</td>
<td>0.39</td>
<td>0.4</td>
<td>0.350</td>
<td>0.407</td>
<td>0.39</td>
<td>0.4</td>
<td>0.375</td>
</tr>
<tr>
<td>78</td>
<td>0.39</td>
<td>0.5</td>
<td>0.297</td>
<td>0.506</td>
<td>0.39</td>
<td>0.1</td>
<td>0.381</td>
</tr>
<tr>
<td>79</td>
<td>0.39</td>
<td>0.5</td>
<td>0.297</td>
<td>0.506</td>
<td>0.39</td>
<td>0.4</td>
<td>0.375</td>
</tr>
<tr>
<td>80</td>
<td>0.39</td>
<td>0.1</td>
<td>0.376</td>
<td>0.105</td>
<td>0.39</td>
<td>0.1</td>
<td>0.381</td>
</tr>
</tbody>
</table>
Appendix B

MATLAB Scripts and Functions

B-1 Initialization Script

This is the main code for initialization of an 8 x 1 OBFN system. Other OBFN setups have similar code except some variables, e.g., Nrings, AE, P_init, OBFN_config and structure of initial \( \kappa_0 \) and \( \phi_0 \) (K0 and P0) need to be appropriately changed.

```matlab
%--------------------------- READ ME -------------------------------
% Initialization of 8x1 OBFN
%-------------------------------------------------------------------
clear all; clc; clf; rng('shuffle')
% Given OBFN configuration:
OBFN_config = [1 1 1 1 1 1 1;
               -1 1 1 1 1 1 1;
               1 -2 -3 1 1 1 1;
               -4 -2 -3 1 1 1 1;
               1 1 1 -6 -7 -8 -9;
               -5 1 1 -6 -7 -8 -9;
               1 -10 -11 -6 -7 -8 -9;
               -12 -10 -11 -6 -7 -8 -9];
Nrings = 12; % number of ORR
AE = 8; % number of AE / path
P_init = 3; % number of initialization parameter of phi
delay = 2e-10*(0:1:AE-1); % Desired delays
% Initialization using group delay error approach:
K_array = (0.09:0.1:0.99); % Array of Kappa
P_array = (0:0.1:1); % Array of Phi
iter = 1; % initialize iteration
max_iter = 1000; % maximum iteration of random initialization
C = 10*ones(1,max_iter); % to save the error
K1 = zeros(max_iter,Nrings); % to save the good initial guess
P1 = zeros(max_iter,Nrings); % to save the good initial guess
P2 = zeros(max_iter,3);
% iteration:
```
while iter <= max_iter
    K = randsample(K_array,1);  % Randomly picking kappa
    P = randsample(P_array,P_init,'true');  % Randomly picking phi
    K0 = K*ones(1,Nrings);  % Initial guess of kappa
    P0 = [0 2*pi-P(1) P(1) 0 0 2*pi-P(2) 2*pi-P(3) P(3) P(2) ... 2*pi-P(1) P(1) 0];  % Initial guess of phi
    % Find the error:
    % Normalized squared group delay error:
    [~,~,err,err1] = norm_squared_error_delay(K0,P0,delay,...
        OBFN_config,100);
    % Squared group delay error:
    [err,err1,~,~] = norm_squared_error_delay(K0,P0,delay,...
        OBFN_config,100);
    % Save the error into C:
    C(iter,:) = err1;
    fprintf('Initial start # %d of %d ---- Error = %d
',iter,...
        max_iter,C(iter));
    % Save initial guess into K1 and P1:
    K1(iter,:) = K0;
    P1(iter,:) = P0;
    P2(iter,:) = P;
    % Increment the iteration:
    iter = iter + 1;
end
[val,id] = min(C);  % Find the minimum error
% Print the result:
fprintf('
The best initial guess for Kappa: \n');
display(K1(id,:));
fprintf('The best initial guess for Phi: \n');
display(P1(id,:));

B-2 Main Simulation Script

This is the main code for simulation of an 8 x 1 OBFN system. Other OBFN setups have similar code except some variables, e.g., Nrings, AE, Nlayers, OBFN_config and structure of initial \( \kappa_0 \) and \( \phi_0 \) (\( \kappa_0 \) and \( \phi_0 \)) need to be appropriately changed.

%----------------------------- READ ME -----------------------------
% Simulation of 8x1 OBFN
%______________________________________________________________
clear all; clc; clf; rng(5);
T = 0.08e-9;  % roundtrip time in seconds
r = 0.95;  % power loss
c = 1.66663721635e+08;  % speed of signal propagation
lambda = 1550e-9;  % wavelength
cf = c/lambda;  % optical carrier frequency for 1550nm laser
bandwidth = 2e9;
fsmin = cf - bandwidth/2;  % minimum frequency of bandwidth
fmax = cf + bandwidth/2;  % maximum frequency of bandwidth
Nfreq = 20;  % discretise frequency to this number
24
Nneurons 23
AE Nrings 22
Nexamples_stoc 21
Nexamples 20
freq 19
kappa0 18
phi 17
% Kappa and phi in vector:
16
15
freq = fmin:(fmax-fmin)/(Nfreq-1):fmax;  % frequency vector
Nfreq_samp = 3;  % number of random picking frequencies
%
17
16

15
%
14
% Iteration steps, examples, layers:
Nsteps = 5000;  % iteration steps
Nextamples = 20000;  % number of training examples
Nextamples_stoc = 2;  % examples used in stochastic gradient projection
Nrings = 12;  % number of ORR
AE = 8;  % number of AE / path
Nlayers = 7;  % number of layers
Nneurons = AE*Nfreq;  % number of neurons each layer
learn_rate = 2e-5;  % step size / learning rate
mu = 0.7;  % momentum coefficient between [0,1]
%
28
27
%
26
% Given OBFN configuration. 1 = identity, -a = ORR index a.
OBFN_config = [1 1 1 1 1 1 1; 24
-1 1 1 1 1 1 1; 23
1 -2 -3 1 1 1 1; 22
-4 -2 -3 1 1 1 1; 21
1 1 1 -6 -7 -8 -9; 20
-5 1 1 -6 -7 -8 -9; 19
1 -10 -11 -6 -7 -8 -9; 18
-12 -10 -11 -6 -7 -8 -9]; 17
OBFN_structure = zeros(AE,AE,Nlayers);
for l = 1:Nlayers
for i = 1:AE
    OBFN_structure(i,i,l) = OBFN_config(i,l);
end
end
%
37
36
% Delays:
35
delay = 2e-10*(0:1:AE-1);
%
34
33
% Initial Kappa: (from initialization using group delay approach)
kappa = zeros(AE,AE,Nrings);
%
32
31
% Initial Phi: (from initialization using group delay approach)
phi = zeros(AE,AE,Nrings);
%
30
29
% Constraint:

38
37
36
kappa = 0.0;  % constrain Kappa and phi
phi = 0.0;  % constrain Kappa and phi

35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
Nfreq_samp = 3;  % number of random picking frequencies
20
Nneurons = AE*Nfreq;  % number of neurons each layer
19
freq = fmin:(fmax-fmin)/(Nfreq-1):fmax;  % frequency vector
18
kappa0 = zeros(AE,AE,Nrings);
17
phi = zeros(AE,AE,Nrings);
16
freq = fmin:(fmax-fmin)/(Nfreq-1):fmax;  % frequency vector
15
\begin{verbatim}
67    f*T+p)) \times r \times \sqrt{(1-k)^2};
68    \%
69    M'autitude response:
70    H_{mag} = @(f,k,p) (\sqrt{((1-k+r^2-2*\sqrt{(1-k)*r})+\cos(2*pi*f*T+p))});
71
72    \%
73    Generating training examples:
74    \%
75    Signal coming to reference path (AE 1):
76    input = zeros(AE*Nfreq,Nexamples);
77    input(1:Nfreq,:) = randn(Nfreq,Nexamples)+1j*randn(Nfreq,Nexamples);
78    \%
79    Desired output:
80    output = zeros(AE*Nfreq,Nexamples);
81    output(1:Nfreq,:) = input(1:Nfreq,:);
82    \%
83    Signal coming to other paths:
84    \%
85    for i=1:AE-1
86       for j=1:Nfreq
87          diff = 2*pi*freq(j)*delay(i+1);
88          input(i*Nfreq+j,:) = exp(-1j*diff)*input(j,:);
89          output(i*Nfreq+j,:) = output(j,:); 
90    end
91 end
92 \%
93 \%
94 \%
95 \%
96 \%
97 \%
98 \%
99 \%
100 \%
101 \%
102 \%
103 \%
104 \%
105 \%
106 \%
107 \%
108 \%
109 \%
110 \%
111 \%
112 \%
113 \%
114 \%
115 \%
116 \%
117 \%
118 \%
119 \%
\end{verbatim}
\% Stochastic examples and frequencies:
[X0_samp2,D_samp2,X0_test_samp2,D_test_samp2,freq_new] = ... 
  stochastic_example(X0,D,X0_test,D_test,Nexamples,... 
  Nexamples_stoc,Nfreq,Nfreq_samp,AE,freq);
\% Weights:
W  = create_weights(OBFN_structure,freq_new,kappa,phi,H);
W_des = create_weights(OBFN_structure,freq_new,kappa,phi,H_mag);
\% Forward pass:
for j = 1:Nexamples_stoc
  x(:,j) = X0_test_samp2(:,j);
  d(:,j) = D_samp2(:,j);
  d_test(:,j) = D_test_samp2(:,j);
  for n = 1:Nlayers
    d(:,j) = W_des(:,n)*d(:,j);
    d_test(:,j) = W_des(:,n)*d_test(:,j);
    x(:,j) = W(:,n)*x(:,j);
  end
\% Add extra phase shift:
  d(:,j) = W_ext*d(:,j);
  d_test(:,j) = W_ext*d_test(:,j);
\% Test error:
err(i) = err(i) + 0.5*norm(x(:,j)-d_test(:,j))^2;
end
\% Normalized test error:
err(i) = err(i)/Nexamples_stoc;
\% One step of gradient descent:
\% Backpropagation algorithm:
dCw = obfn_backprop(X0_samp2,d,W,obfn_mask,Nfreq_samp);
\% dCw with constraints: (part of complex number separation concept)
dCw = constrained_dCw(dCw,obfn_mask);
\% dC/dkappa and dC/dphi:
[dCk, dCp] = find_dCparameters(OBFN_structure,freq_new,kappa,phi,...
  obfn_mask,dCw,dHk,dHp);
\% updating kappa and phi:
step_size = learn_rate./(1+i/Nsteps);
vk = mu*vk - 100*step_size.*dCk;  \% implementing momentum
vp = mu*vp - step_size.*dCp;  \% implementing momentum
kappa = kappa + vk;
phi = phi + vp;
\% Gradient projection method:
for nr = 1:Nrings
  kappa(nr) = max(kappa_low,kappa(nr));
  kappa(nr) = min(kappa_hi,kappa(nr));
  phi(nr) = max(phi_low,phi(nr));
  phi(nr) = min(phi_hi,phi(nr));
end
\% print itertion status:
fprintf('iteration %d of %d ... err = %g \n',i,Nsteps,err(i));
end
\% Normalized squared group delay error:
[~,~,err0,err1] = norm_squared_error_delay(kappa,phi,delay,...
  OBFN_config,100);
% Plot results:
% Test error plot:
array = (1:Nsteps);
figure(1);
semilogy(array, err);
ylabel('Test error'); legend('Cost function'); xlabel('Iteration');
grid on;
%
% Group delay response plot:
figure(2);
plot_delay_response(OBFN_config, kappa, phi, delay)
title(['Group Delay Response of ' num2str(AE) ' \times 1 OBFN']);
xlabel('Normalized Frequency (Hz)'); ylabel('Group Delay (s)');

B-3 MATLAB Functions

This section presents scripts for all MATLAB functions executed to tune OBFNs using deep learning approach.

B-3-1 Complex Number Separation Function

Complex number separation is implemented into two functions, one is to separate the complex number, and the other to deform it into required configuration.

```matlab
function X = complex_decompose(X0)
%---------------------------- READ ME ---------------------------------
% This function is used to decompose matrix of complex numbers into its
% real and imaginary numbers
%----------------------------------------------------------------------

[Ninputs, Nexamples] = size(X0);
X = zeros(2*Ninputs, Nexamples);
X(1:2:2*Ninputs-1,:) = real(X0);
X(2:2:2*Ninputs,:) = imag(X0);
end

function Hnew = Hcomplex_decompose(H0)
%---------------------------- READ ME ---------------------------------
% This function is used to deform the array H0 of complex numbers:
% H0 = [a1+ib1 a2+ib2 a3+ib3 ..... an+ibn]
% into:  Hnew = [a1 -b1 0 0 ..... 0 0;  
%              b1 a1 0 0 ..... 0 0; 
%              ... ... ... ... ... ;
%              ... ... ... ... ... ;
%              0 0 0 0 ..... an -bn; 
%              0 0 0 0 ..... bn an]

sizeH = length(H0);
Hnew = zeros(sizeH+2, sizeH+2);
for i = 1:sizeH
    Hnew((2*i-1):(2*i), (2*i-1):(2*i)) = [real(H0(i)) - imag(H0(i));
                                          imag(H0(i)) real(H0(i))];
end
```

Herminarto Nugroho Master of Science Thesis
B-3 MATLAB Functions

B-3-2 Extra Phase Shifters Function

```matlab
function [W_ext,ang] = phase_shifter(AE,Nfreq,cf,delay)
%-------------------------- READ ME ---------------------------------
% This function is used to compute extra phase shifters from given
% frequency center (cf), delay, number of AE, and Nfreq. Output of
% this function is weight of the extra phase shifters with the same
% dimension as weight of NN and the angle of extra phase shifters.
%--------------------------------------------------------------------
% Structure and angle of the extra phase shifter:
for i = 1:AE
    ext_structure(i,i) = -1*i;
end
for i=2:AE
    ang(i) = 2.*pi.*cf.*delay(i)-(i-1)*pi;
    ang(i) = ang(i) - floor(ang(i)/(2*pi))*2*pi;
end
% Weight of extra phase shifters:
shift_path = exp(1i*ang);
W_ext = zeros(2*AE+Nfreq,2*AE+Nfreq);
W1 = zeros(Nfreq,1);
W2 = zeros(2*Nfreq,2*Nfreq);
for i=1:AE
    for j=1:AE
        if ext_structure(i,j) == 0
            continue
        else
            for l=1:Nfreq
                W1(l) = shift_path(i);
                W2(2*l-1:2*l-1:2*l-1:2*l) = Hcomplex_decompose(W1(l));
            end
            W_ext((i-1)*2+Nfreq+1:i)*2+Nfreq,(j-1)*2+Nfreq+1:j*2+...
                Nfreq) = W2;
        end
    end
end
```

B-3-3 Stochastic Gradient Projection Method and Random Picking Frequencies Function

This function is used to implement both stochastic gradient projection method and random picking frequencies.

```matlab
function [X0_samp2,D_samp2,X0_test_samp2,D_test_samp2,freq_new] = ...
    stochastic_example(X0,D,X0_test,D_test,Nexamples,Nexamples_stoc,...
        Nfreq,Nfreq_samp,AE,freq)
```

Master of Science Thesis

Herminarto Nugroho
% Read Me

% This function is used to implement stochastic gradient projection method and random picking frequency.

expl_array = 1:Nexamples;  % indexing examples
freq_array = 1:Nfreq;  % indexing frequency range

% Random picking examples: (stochastic gradient projection)
expl_id = randsample(expl_array, Nexamples_stoc);

% Random picking frequencies:
freq_id = randsample(freq_array, Nfreq_samp);
freq_new = freq(freq_id);

% Creating new input vector and desired output vector:
X0_samp = X0(:, expl_id);
D_samp = D(:, expl_id);
X0_test_samp = X0_test(:, expl_id);
D_test_samp = D_test(:, expl_id);

id = zeros(1, 2*Nfreq_samp*AE);
for i = 1:AE
    for j = 1:Nfreq_samp
        id(2*Nfreq_samp*(i-1)+2*j-1) = 2*Nfreq*(i-1)+2*freq_id(j)-1;
        id(2*Nfreq_samp*(i-1)+2*j) = 2*Nfreq*(i-1)+2*freq_id(j);
    end
end
X0_samp2 = X0_samp(id,:);
D_samp2 = D_samp(id,:);
X0_test_samp2 = X0_test_samp(id,:);
D_test_samp2 = D_test_samp(id,:);

B-3-4 Creating Mask Matrices Function

function obfn_mask = create_mask(OBFN_structure, Nfreq)

% Read Me

% This function is used to create mask matrix from the given OBFN structure and Nfreq.

[AE, ~, Nlayers] = size(OBFN_structure);
obfn_mask = zeros(2*AE*Nfreq, 2*AE*Nfreq, Nlayers);
for n = 1:Nlayers
    for i = 1:AE
        if ~OBFN_structure(i, j, n) == 1 || OBFN_structure(i, j, n) == 0
            continue
        end
        for l = 1:Nfreq
            obfn_mask((i-1)*2*Nfreq+1+2*(l-1):i*2*Nfreq+2*... (1-l), (j-1)*2*Nfreq+1+2*(l-1):j*2*Nfreq+2*... (1-l), n) = ones(2, 2);
        end
    end
end
B-3-5 Creating Weight Matrices Function

```matlab
function W = create_weights(OBFN_structure, freq, kappa, phi, H)

%_____________________________ READ ME ______________________________
% This function is to create weight matrix from the given OBFN_structure
% and Nfreq
%______________________________________________________________________

[AE, ~, Nlayers] = size(OBFN_structure);
Nfreq = length(freq);
W = zeros(2*AE*Nfreq, 2*AE*Nfreq, Nlayers);
H1 = zeros(Nfreq, Nlayers);
H2 = zeros(2*Nfreq, 2*Nfreq, Nlayers);

for n = 1:Nlayers
    for i = 1:AE
        for j = 1:AE
            if OBFN_structure(i, j, n) == 0
                continue
            end
            elseif OBFN_structure(i, j, n) == 1
                W((i-1)*2*Nfreq+1:i+2*Nfreq, (j-1)*2*Nfreq+1:j+2*Nfreq, ...
                n) = eye(2*Nfreq, 2*Nfreq);
            else
                for l = 1:Nfreq
                    H1(l, n) = H(freq(l), kappa(-1*OBFN_structure(i, j, ...)
                    n), phi(-1*OBFN_structure(i, j, n)));
                    % Separate real and imaginary part:
                    H2(2*l-1:2*l, 2*l-1:2*l, n) = Hcomplex_decompose(H1...
                    (l, n));
                end
                W((i-1)*2*Nfreq+1:i+2*Nfreq, (j-1)*2*Nfreq+1:j+2*...
                Nfreq, n) = H2(:, :, n);
            end
        end
    end
end
end
end
end
end
```

B-3-6 Backpropagation Algorithm Function

```matlab
function dW = obfn_backprop(X0, D, W, obfn_mask, Nfreq)

%_____________________________ READ ME ______________________________
% This function is to perform backpropagation algorithm.
% Output of this function is dW, which is the devative of the cost
% function with respect to weights.
%______________________________________________________________________

[Nneurons, Nexamples] = size(X0);
```
\[ \text{size}(W) \]
\[ X = \text{zeros}(\text{Nneurons}, \text{Nlayers}+1); \]
\[ dW = \text{zeros}(\text{Nneurons}, \text{Nneurons}, \text{Nlayers}); \]
\[ \text{for cur_example} = 1: \text{Nexamples} \]
\[ \% \text{Forward pass:} \]
\[ X(:,1) = X0(:,\text{cur_example}); \]
\[ \text{for } n = 1: \text{Nlayers} \]
\[ X(:,n+1) = W(:,:,n) \times X(:,n); \]
\[ \text{end} \]
\[ \% \text{Back pass:} \]
\[ \text{for cur_layer} = \text{Nlayers}:-1:1 \]
\[ \text{for } i = 1:2* \text{Nfreq} : \text{Nneurons} \]
\[ \text{for } j = 1:2* \text{Nfreq} : \text{Nneurons} \]
\[ \% \text{Checking mask matrices:} \]
\[ \text{if obfn_mask}(i,j,\text{cur_layer})==0 \]
\[ \text{continue} \]
\[ \text{else} \]
\[ \% \text{Derivative of the last layer:} \]
\[ dCx = -(D(:,\text{cur_example}) - X(:,\text{Nlayers}+1)); \]
\[ \% \text{Derivative of layer } 1 \text{ to } \text{N-1:} \]
\[ \text{for } \text{nn} = \text{Nlayers}:-1:(\text{cur_layer}+1) \]
\[ dCx = W(:,:,\text{nn}).' \times dCx; \]
\[ \text{end} \]
\[ \text{while } \text{iter} < \text{Nfreq} + 2 \]
\[ dW(i+\text{iter},j+\text{iter},\text{cur_layer}) = dW(i+\text{iter},j+\text{iter},\text{cur_layer}) + dCx(i+\text{iter},\text{cur_layer})/\text{Nexamples}; \]
\[ dW(i+\text{iter}+1,j+\text{iter}+1,\text{cur_layer}) = dW(i+\text{iter}+1,j+\text{iter}+1,\text{cur_layer}) + dCx(i+\text{iter}+1,\text{cur_layer})/\text{Nexamples}; \]
\[ X(j+\text{iter},\text{cur_layer}) + dCx(i+\text{iter}+1,\text{cur_layer})/\text{Nexamples}; \]
\[ dW(i+\text{iter}+1,j+\text{iter}+1,\text{cur_layer}) = dW(i+\text{iter}+1,j+\text{iter}+1,\text{cur_layer}) + dCx(i+\text{iter}+1,\text{cur_layer}) + X(j+\text{iter}+1,\text{cur_layer})/\text{Nexamples}; \]
\[ dW(i+\text{iter},j+\text{iter}+1,\text{cur_layer}) = dW(i+\text{iter},j+\text{iter}+1,\text{cur_layer}) + dCx(i+\text{iter}+1,\text{cur_layer}) + X(j+\text{iter}+1,\text{cur_layer})/\text{Nexamples}; \]
\[ \text{iter} = \text{iter} + 2; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]

**B-3-7 Constrained Backpropagation Function**

The implementation of separation of complex number requires modification in backpropagation by introducing constraints. This function is used to implement that constraint.

```matlab
function dCw_new = constrained_dCw(dCw, obfn_mask)
%________________________________________ READ ME ________________________________
% This function is used to change matrix dCw
```
% dCw = [a b; c d]  
% into: dCw_new = [(a+d) -(c-b); (c-b) (a+d)]  
% [Nneurons, , Nlayers] = size(dCw);  
dCw_new = zeros(Nneurons, Nneurons, Nlayers);  
for n=1:Nlayers  
[I, J] = find(obfn_mask(:, :, n)==0);  
for l=1:4:length(I)-2  
i = I(l); k = I(l+1);  
j = J(l); m = J(l+3);  
dCw_new(i, j, n) = dCw(i, j, n) + dCw(k, m, n);  
dCw_new(k, m, n) = dCw_new(i, j, n);  
dCw_new(k, j, n) = dCw(k, j, n) - dCw(i, m, n);  
dCw_new(i, m, n) = -dCw_new(k, j, n);  
end  
end  
end

B-3-8 Derivative with respect to Parameters Function

Given gradient obtained by backpropagation, we compute the derivative of the cost function with respect to $\kappa$ and $\phi$ via the chain rule.

% This function is used to change dCw into dCk and dCp
% Parameters needed:
% [Nneurons, , Nlayers] = size(dCw);
% Nfreq = length(freq);
% a = Nneurons/2;
% Nrings = length(kappa);
% Initialization:

function [dCk, dCp] = find_dCparameters(OBFN_structure, freq, kappa, phi, ...
    obfn_mask, dCw, dHk, dHp)
% ___________________________________ READ ME ______________________________
% % This function is used to change dCw into dCk and dCp
% % ___________________________________ READ ME ______________________________

% Parameters needed:
% [Nneurons, , Nlayers] = size(dCw);
% Nfreq = length(freq);
% a = Nneurons/2;
% Nrings = length(kappa);

% Initialization:

dk = zeros(a, a, Nlayers);  
dp = zeros(a, a, Nlayers);  
dCk1 = zeros(a/Nfreq, a/Nfreq, Nlayers);  
dCp1 = zeros(a/Nfreq, a/Nfreq, Nlayers);  
dCk = zeros(1, Nrings);  
dCp = zeros(1, Nrings);  

kappa1 = OBFN_structure;  
phi1 = OBFN_structure;  

kappa1(kappa1==1) = 0;  
phi1(phi1==1) = 0;  
for i=1:Nrings  
    kappa1(kappa1==i) = kappa(i);  
    phi1(phi1==i) = phi(i);  
end
% Calculating partial derivative of cost function w.r.t to kappa and
% phi in the same dimension as dCw:
for n = 1:Nlayers
    [I,J] = find(obfn_mask(:,:,:n)==0);
    for l=1:length(I)
        % Indexing:
        i = I(l);
        k = I(l+1);
        j = J(l);
        m = J(l+2);
        % Computing dk and dp:
        dk(k/2,m/2,n) = dCw(i,j,n)*real(dHk(freq(k/2,Nfreq),
            floor((i-1)/(2*Nfreq))) ,kappa1(ceil(i/(2*Nfreq)) ,ceil(i/...
            (2*Nfreq))) ,n) + dCw(k,j,n)*imag(dHk(freq(k/2,Nfreq)*
            floor((i-1)/(2*Nfreq))) ,kappa1(ceil(i/(2*Nfreq)) ,ceil(i/(2*Nfreq)) ,...
            (2*Nfreq))) ,n)*phi1(ceil(i/(2*Nfreq)) ,ceil(i/(2*Nfreq)) ,n);)
        dp(k/2,m/2,n) = dCw(i,j,n)*real(dHp(freq(0.5*k-Nfreq)*
            floor((i-1)/(2*Nfreq))) ,kappa1(ceil(i/(2*Nfreq)) ,ceil(i/(2*Nfreq)) ,...
            (2*Nfreq))) ,n) + dCw(k,j,n)*imag(dHp(freq(0.5*k-Nfreq)*
            floor((i-1)/(2*Nfreq))) ,kappa1(ceil(i/(2*Nfreq)) ,ceil(i/(2*Nfreq)) ,n),...
            (2*Nfreq)) ,n)*phi(ceil(i/(2*Nfreq)) ,ceil(i/(2*Nfreq)) ,n);)
    end
end
for n = 1:Nlayers
    for i = 2:2:Nneurons
        % Checking mask:
        if obfn_mask(i,j,n)==0
            continue
        end
        dCk1(ceil(i/(2*Nfreq)) ,ceil(i/(2*Nfreq)) ,n) = dCk1(ceil(i/...
            (2*Nfreq)) ,ceil(i/(2*Nfreq)) ,n) + dk(i/2,j/2,n);)
        dCp1(ceil(i/(2*Nfreq)) ,ceil(i/(2*Nfreq)) ,n) = dCp1(ceil(i/...
            (2*Nfreq)) ,ceil(i/(2*Nfreq)) ,n) + dp(i/2,j/2,n);)
    end
end
for n=1:Nlayers
    for iter = 1:Nrings
        [X,Y] = find(OBFN_structure(:,:,:n)==-iter);
        if isempty(X)
            continue
        else
            for i=1:length(X)
                dCk(iter) = dCk(iter) + dCk1(X(i),Y(i),n);
                dCp(iter) = dCp(iter) + dCp1(X(i),Y(i),n);
            end
        end
    end
end
B-3-9  Group Delay Error Function

function [err, err1, norm_err, norm_err1] = norm_squared_error_delay...
   (kappa1, phi1, delay, obfn_config, points)
%---------------------------------------------------------------------
% The function is used to compute the normalized squared group delay
% error of given kappa, phi, and desired delay
% Descriptive name
%____________________________________________________________________
T = 0.08e-9;  % roundtrip time in seconds, from Blokpoel’s thesis
r = 0.95;    % power loss
lambda = 1.66663721635e+08;  % speed of signal propagation
cf = c/lambda;  % optical carrier frequency for 1550nm laser
bandwidth = 2e9;
fmin = cf - bandwidth/2;  % minimum frequency of bandwidth
fmax = cf + bandwidth/2;  % maximum frequency of bandwidth
Nrings = length(kappa1);
AE = length(delay);
% Frequency set:
f = fmin:((fmax-fmin)/(points-1)):fmax;
% Delay response:
tao = @(k,p) ((0.5*T).*((1-r^2*(1-k))./(1+r^2*(1-k)-2*r*sqrt(1-k).*cos...
(2*pi*f*T+p))+(r^2*(1-k))./(1-k+r^2-2*r*sqrt(1-k).*cos(2*pi*f*T+p))));
% For kappa1 and phi1:
% Actual group delay:
output = zeros(Nrings,points);
for i=1:Nrings
   output(i,:) = tao(kappa1(i),phi1(i));
end
path1 = zeros(AE-1,points);
for i=2:AE
   for j=1:AE-1
      if obfn_config(i,j) == 1
         continue
      end
      index = -1*obfn_config(i,j);
      path1(i-1,:) = path1(i-1,:) + output(index,:);
   end
end
% Desired group delay:
delay_array = zeros(AE-1,points);
for j=2:AE
   delay_array(j-1,:) = ones(1,points).*delay(j);
end
% Computing errors:
err = zeros(AE-1,1);  err1 = 0;
norm_err = zeros(AE-1,1); norm_err1 = 0;
for k=2:AE
   norm_err(k-1) = 0.5*(norm(path1(k-1,:)-delay_array(k-1,:))^2)/...
   (norm(delay_array(k-1,:))^2);
   err(k-1) = (norm(path1(k-1,:)-delay_array(k-1,:))^2);
   norm_err1 = norm_err1 + norm_err(k-1);
end
err1 = err1 + err(k-1);
B-3-10  Plot Delay Responses Function

1 function plot_delay_response(obfn_config,kappa,phi,delay)
2 %________________________________ READ ME ______________________________
3 %The function ORR_delayresponse gives the delay response of an optical
4 %ring resonator (ORR).
5 %
6 %kappa and phi are the parameters of the ORR and points is the number
7 %of sampling in the frequency range.
8 %____________________________________________________________________
9 T = 0.08e−9;  % roundtrip time in seconds, from Blokpoel’s thesis
10 r = 1;  % power loss
11 c = 1.66663721635e+08;  % speed of signal propagation
12 lambda = 1550e−9;  % wavelength
13 cf = c/lambda;  % optical carrier frequency for 1550nm laser
14 bandwidth = 2e9;
15 fmin = cf−bandwidth/2;  % minimum frequency of bandwidth
16 fmax = cf + bandwidth/2;  % maximum frequency of bandwidth
17 Ndelays = length(delay);
18 Nrings = length(kappa);
19 [AE,Nlayers] = size(obfn_config);
20 points = 100;
21 % Frequency set:
22 start = fmin − 0.5∗bandwidth;
23 stop = fmax + 0.5∗bandwidth;
24 f = start:(stop−start)/(points−1):stop;
25 % Delay response:
26 tao = @(k,p) (((0.5∗T).*((1−r−2*(1−k))./(1+r−2*(1−k)−2*r∗sqrt(1−k).*cos(2*pi*f*T+p))));
27 % Output delay response for each ring:
28 output = zeros(Nrings,points);
29 for i = 1:Nrings
30    output(i,:) = tao(kappa(i),phi(i));
31 end
32 % Delay response for each path:
33 d_response = zeros(AE,points);
34 for j = 2:AE
35    for k = 1:Nlayers
36      if obfn_config(j,k) == 1
37         continue
38      end
39      d_response(j,:) = d_response(j,:)+output(−1*obfn_config(j,k,:));
40    end
41 end
42 % Plotting:
43 colorVec = hsv(AE);
44 for m = 2:AE
45    plot(f−cf,d_response(m,:),’Color’,colorVec(m,:));
46    hold on;
47 end
48 end
ylim([0 delay(Ndelays)+delay(2)]);

% Plot desired delay:
for m = 2:Ndelays
    graph2d.constantline(delay(m),'LineStyle','--','Color',colorVec(m,:));
    hold on;
end
ax = gca;
set(ax,'YTick',delay);

%Bandwidth plot:
hx = graph2d.constantline(fmin-cf,'LineStyle','-.','Color','k');
changedependvar(hx,'x');
hx = graph2d.constantline(fmax-cf,'LineStyle','-.','Color','k');
changedependvar(hx,'x');
end

B-4 Non-linear Programming (NLP) Scripts

The following scripts are based on the work of Blokpoel [2] to tune 4×1 OBFNs. We modify his code such that it can be used to tune 8×1 OBFNs. For 16×1 OBFNs, another cost function and optimization script for paths 8 to 16 are derived using the similar approach to 8×1 OBFNs.

B-4-1 Main Script for Tuning using NLP

%______________________________ READ ME ________________________________
% Simulation of tuning 8x1 OBFNs using Non-linear Programming
%_____________________________________________________________________
% clear all; clc;
% Define reference frequency FREF (in Hz)
global FREF max_iter bandwidth
FREF = 1.075249817e14; max_iter = 50;
% Define points of calculation
points = 100; bandwidth = 2e9;
targetdelay = 0.2*[1 2 3 4 5 6 7];
% Optimization process
[x1,FVAL1] = ORR_delayopt(points,targetdelay);
[x23,FVAL23] = ORR_delayopt2and3(points,targetdelay);
[x47,FVAL47] = ORR_delayopt4to7(points,targetdelay);
kappa = [x1(1) x23(1) x23(3) x23(5) x47(9) x47(1) x47(3) x47(5) x47(7)
x47(11) x47(13) x47(15)];
phi = [x1(2) x23(2) x23(4) x23(6) x47(10) x47(2) x47(4) x47(6) x47(8) x47(12)
x47(14) x47(16)];
[err,err1,norm_err,norm_err1] = norm_squared_error_delay(kappa,phi,delay,
OBFN_config,100);

B-4-2 Optimization Scripts for Different Paths
function [parameters,FVAL] = ORR_delayopt(points,targetdelay)

% The function ORR_delayopt optimizes one ORR for a specific target bandwidth and target delay.

global max_iter
options = optimset(@fminbnd);
options = optimset(options,'TolFun',1e-16,'TolX',1e-16,...
'MaxIter',max_iter,'MaxFunEvals',1e6);

global FREF DREF FREQARRAY LENGTH bandwidth
DREF = targetdelay(1); LENGTH = points;
start = FREF-bandwidth/2; stop = FREF+bandwidth/2;
FREQARRAY(1:LENGTH) = start:(stop-start)/(LENGTH-1):stop;
init_par = [.69,0]; % initial parameters
low_constraint = [0,-2*pi]; % lower constraints
hi_constraint = [.995,2*pi]; % higher constraints

% Optimization:
[parameters,FVAL] = fmincon(@delayfun1,init_par,[],[],[],[],...,low_constraint,hi_constraint,[],options);
end

function [parameters,FVAL] = ORR_delayopt2and3(points,targetdelay)
% The function ORR_delayopt2and3 optimizes two and three ORR at once, for a specific target bandwidth and two target delays.

global max_iter
options = optimset(@fminbnd);
options = optimset(options,'TolFun',1e-16,'TolX',1e-16,...
'MaxIter',max_iter,'MaxFunEvals',1e6);

global FREF DREF2 DREF3 FREQARRAY LENGTH bandwidth
DREF2 = targetdelay(2); DREF3 = targetdelay(3); LENGTH = points;
start = FREF-bandwidth/2; stop = FREF+bandwidth/2;
FREQARRAY(1:LENGTH) = start:(stop-start)/(LENGTH-1):stop;
init_par = [.69,.4,.69,-.4,.69,0]; % initial parameters
low_constraint = [0,-2*pi,0,-2*pi,0,-2*pi]; % lower constraints
hi_constraint = [.995,2*pi,.995,2*pi,.995,2*pi]; % higher constraints

% Optimization:
[parameters,FVAL] = fmincon(@delayfun2and3,init_par,[],[],[],[],...,low_constraint,hi_constraint,[],options);
end

function [parameters,FVAL] = ORR_delayopt4to7(points,targetdelay)
% The function ORR_delayopt4to7 optimizes the 4th to 7th path, for a specific target bandwidth and two target delays.

global max_iter
options = optimset(@fminbnd);
options = optimset(options,'TolFun',1e-16,'TolX',1e-16,...
'MaxIter',max_iter,'MaxFunEvals',1e6);

global FREF DREF4 DREF5 DREF6 DREF7 FREQARRAY LENGTH bandwidth
DREF4 = targetdelay(4); DREF5 = targetdelay(5); LENGTH = points;
DREF6 = targetdelay(6); DREF7 = targetdelay(7);
start = FREF-bandwidth/2; stop = FREF+bandwidth/2;
FREQARRAY(1:LENGTH) = start:(stop-start)/(LENGTH-1):stop;
init_par = [.69,.4,.69,.4,.69,-.4,.69,-.4,.69,0,.69,.4,...
.69,-.4,.69,0];
low_constraint = [0,-2*pi,0,-2*pi,0,-2*pi,0,-2*pi,0,-2*pi,0,...
B-4-3 Cost Functions for Different Paths

function mu1 = delayfun1(x)
global DREF FREQARRAY LENGTH
mu1 = 0;
func_mu = @(x1,x2,kk)((x1+.08)/(2-x1-2*sqrt(1-x1)*cos(8*atan(1)*...
FREQARRAY(kk)*.08e-9+x2)));
for k=1:1:LENGTH
mu1 = mu1+(func_mu(x(1),x(2),k)-DREF)^2;
end

function mu2and3 = delayfun2and3(x)
global DREF2 DREF3 FREQARRAY LENGTH
mu2and3 = 0;
func_mu = @(x1,x2,kk) ((x1+.08)/(2-x1-2*sqrt(1-x1)*cos(8*atan(1)*...
FREQARRAY(kk)*.08e-9+x2)));
for k=1:1:LENGTH
%For two ORRs:
mu2and3=mu2and3+(func_mu(x(1),x(2),k)+func_mu(x(3),x(4),k)...
- DREF2)^2;

%For three ORRs:
mu2and3=mu2and3+(func_mu(x(1),x(2),k)+func_mu(x(3),x(4),k)...
+ func_mu(x(5),x(6),k)-DREF3)^2;
end

function mu4to7 = delayfun4to7(x)
global DREF4 DREF5 DREF6 DREF7 FREQARRAY LENGTH
mu4to7 = 0;
func_mu = @(x1,x2,kk) ((x1+.08)/(2-x1-2*sqrt(1-x1)*cos(8*atan(1)*...
FREQARRAY(kk)*.08e-9+x2)));
for k=1:1:LENGTH
%For 4 ORRs:
mu4to7=mu4to7+(func_mu(x(1),x(2),k)+func_mu(x(3),x(4),k)...
+ func_mu(x(5),x(6),k)+func_mu(x(7),x(8),k)...
- DREF4)^2;

%For 5 ORRs:
mu4to7=mu4to7+(func_mu(x(1),x(2),k)+func_mu(x(3),x(4),k)...
+ func_mu(x(5),x(6),k)+func_mu(x(7),x(8),k)...
+ func_mu(x(9),x(10),k)-DREF5)^2;

%For 6 ORRs:
mu4to7=mu4to7+(func_mu(x(1),x(2),k)+func_mu(x(3),x(4),k)...
+ func_mu(x(5),x(6),k)+func_mu(x(7),x(8),k)...
+ func_mu(x(9),x(10),k)-DREF6)^2;
end
19 + func_mu(x(11),x(12),k)+func_mu(x(13),x(14),k)...
20 - DREF6)^2;
21
22 %For 7 ORRs:
23 mu4to7=mu4to7 + (func_mu(x(1),x(2),k)+func_mu(x(3),x(4),k)...
24 + func_mu(x(5),x(6),k)+func_mu(x(7),x(8),k)...
25 + func_mu(x(11),x(12),k)+func_mu(x(13),x(14),k)...
26 + func_mu(x(15),x(16),k)-DREF7)^2;
Bibliography


Herminarto Nugroho Master of Science Thesis


Master of Science Thesis Herminarto Nugroho
Glossary

List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBFN</td>
<td>Optical Beamforming Network</td>
</tr>
<tr>
<td>ORR</td>
<td>Optical Ring Resonator</td>
</tr>
<tr>
<td>NLP</td>
<td>Non-linear Programming</td>
</tr>
<tr>
<td>PAA</td>
<td>Phased Array Antenna</td>
</tr>
<tr>
<td>AE</td>
<td>Antenna Element</td>
</tr>
<tr>
<td>GRASP</td>
<td>Greedy Randomized Adaptive Search Procedures</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>FSR</td>
<td>Free Spectral Range</td>
</tr>
<tr>
<td>DE</td>
<td>Delay Element</td>
</tr>
<tr>
<td>IM</td>
<td>Intensity Modulation</td>
</tr>
<tr>
<td>SSB-SC</td>
<td>Single-Sideband Suppressed-Carrier</td>
</tr>
<tr>
<td>DBN</td>
<td>Deep Belief Network</td>
</tr>
<tr>
<td>RBM</td>
<td>Restricted Boltzmann Machine</td>
</tr>
<tr>
<td>LBN</td>
<td>Logistic Belief Network</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to Noise Ratio</td>
</tr>
</tbody>
</table>