DEEP LEARNING BASED CHANNEL EQUALIZATION FOR MIMO ISI CHANNELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Berke Eren
September 2022

Deep Learning Based Channel Equalization for MIMO ISI Channels
By Berke Eren
September 2022

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Tolga Mete Duman (Advisor)

Sinan Gezici

Ayşe Melda Yüksel Turgut

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

DEEP LEARNING BASED CHANNEL EQUALIZATION FOR MIMO ISI CHANNELS

Berke Eren
M.S. in Electrical and Electronics Engineering
Advisor: Tolga Mete Duman
September 2022

Future wireless communications is expected to bring significant changes along with a number of emerging technologies such as 5G, virtual reality, edge computing, and IoT. These developments pose unprecedented demands in terms of capacity, coverage, latency, efficiency, flexibility, compatibility, and quality of experience on wireless communication systems. Machine Learning (ML) techniques are considered as a promising tool to tackle this challenge due to their ability to manage big data, powerful nonlinear mapping, and distributed computing capabilities. There have been many research results addressing different aspects of ML algorithms and their connections to wireless communications; however, there are still various challenges that need to be addressed. In particular, their use for communication systems with memory, is not fully investigated. With this motivation, this thesis considers an application of ML, in particular, deep learning (DL), techniques for communications over intersymbol interference (ISI) channels.

In this thesis, we propose DL-based channel equalization algorithms for channels with ISI. We introduce three different DL-based ISI detectors, namely sliding bidirectional long short term memory (Sli-BiLSTM), sliding multi layer perceptron (Sli-MLP), and sliding iterative (Sli-Iterative), and demonstrate that they are computationally efficient and capable of performing equalization under a variety of channel conditions with the knowledge of the channel state information. We also employ sliding bidirectional gated recurrent unit (Sli-BiGRU) and Sli-MLP, which are more suitable for use with fixed ISI channels. As an extension, we also examine DL-based equalization techniques for multiple-input multiple-output (MIMO) ISI channels. Numerical results show that proposed models are well suited for equalization of ISI channels with perfect as well as noisy CSI for a broad range of signal-to-noise ratio (SNR) levels as long as the ISI length is not excessive. It is also shown that the proposed DL-based ISI detectors perform

very close to the optimal solution, namely, the maximum likelihood sequence estimation, implemented through the Viterbi Algorithm while having considerably less complexity, and they have superior performance compared to MMSE-based channel equalization.

Keywords: Deep learning, neural networks, intersymbol interference, MIMO detection, equalization, wireless communications.

ÖZET

ÇOKLU GİRİŞ ÇOKLU ÇIKIŞ SİSTEMLERDE SEMBOLLER ARASI GİRİŞİM KANALLARI İÇİN DERİN ÖĞRENME TABANLI KANAL EŞİTLEME

Berke Eren
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Tolga Mete Duman
Eylül 2022

Geleceğin kablosuz iletişiminin; 5G, sanal gerçeklik, uç hesaplama, nesnelerin interneti gibi bir dizi yeni teknolojiyle birlikte önemli değişiklikler getirmesi bekleniyor. Bu gelişmeler, kablosuz iletişim ve sistemlerde kapasite, kapsama alanı, gecikme, verimlilik, esneklik, uyumluluk, deneyim kalitesi açısından gittikçe artan talepler doğurmaktadır. Makine öğrenimi (ML) teknikleri; büyük verileri yönetme, doğrusal olmayan eşlem ve dağıtılmış bilgi işlem yeteneği nedeniyle bu zorluğun üstesinden gelmek için çok umut verici bir araç olarak kabul ediliyor. Makine öğrenimi algoritmalarının farklı yönlerini ve bunların kablosuz iletişimle olan bağlantılarını ele alan birçok araştırma bulunsa da hala ele alınması gereken çeşitli problemler vardır. Özellikle hafızalı iletişim sistemleri için kullanımları tam olarak araştırılmamıştır. Bu motivasyonla, bu tez özellikle semboller arası girişim (ISI) kanalları için derin öğrenme methodlarını ele almaktadır ve bu çözümleri çoklu giriş çoklu çıkış sistemleri (MIMO) için de genişletmektedir.

Bu tezde, ISI kanalları için derin öğrenme (DL) tabanlı kanal eşitleme algoritmaları öneriyoruz. Biz üç farklı derin öğrenme tabanlı ISI dedektör modellerini (Sli-BiLSTM, Sli-MLP, Sli-Iterative) tanıtıyoruz ve bunların hesaplama açısından daha verimli olduklarını ve temel kanal modelini bilerek çeşitli kanal koşulları altında algılama gerçekleştirebileceklerini gösteriyoruz. Ayrıca sabit ISI kanalları ile kullanımı daha uygun olan Sli-BiGRU ve Sli-MLP modellerini kullanıyoruz. Buna ek olarak, MIMO ISI kanalları için de DL tabanlı eşitleme tekniklerini de inceliyoruz. Sayısal sonuçlar, önerilen modellerin, ISI uzunluğu aşırı olmadığı sürece, tam veya hatalı kanal bilgisiyle (CSI) ve geniş bir sinyal-gürültü oranı (SNR) seviyelerinde ISI kanallarının eşitlemesi için oldukça uygun olduğunu göstermektedir. Ayrıca önerilen derin öğrenme tabanlı ISI dedektörlerinin,

Viterbi algoritması ile uygulanan maksimum olabilirlik dizisi tespiti (MLSE) olan optimal çözüme çok daha az karmaşıklığa sahip olurken oldukça yakın performans gösterdiği ve MMSE-tabanlı kanal eşitleme yöntemine göre üstün performansa sahip olduğu gösterilmiştir.

Anahtar sözcükler: Derin öğrenme, sinir ağları, semboller arası girişim, çoklu giriş çoklu çıkış, kanal denkleme, kablosuz haberleşme.

Acknowledgement

Prior to anything else, I would like to convey my profound gratitude to my advisor Prof. Tolga M. Duman for his tireless assistance, vast knowledge, inspiration, and patience throughout my M.S. studies. He helped and inspired my research with enlightening conversations and suggestions, and I would like to thank him for that. Without his valuable support, I would not have been able to conduct this research, thus I consider it a great privilege to have worked with him for three years.

This work was supported by Türk Telekom through BTK 5G and Beyond graduate research fellowship program, and I gratefully acknowledge this support.

I would also want to express my sincere gratitude for the financial assistance provided by the 2210-A program of The Scientific and Technological Research Council of Turkey (TÜBİTAK).

I would like to thank all the members of the Bilkent Communication Theory and Application Research (CTAR) Lab, Mohammad Kazemi, Javad Haghighat, Mert Özateş, Ozan Aygün, Sadra Charandabi, Büşra Tegin, Muhammad Atif Ali, and Mohammad Javad Ahmadi.

I would like to thank my friends Eren Kurkut, Evren Bayram, Gökalp Şerbetçi, Cenk Cidecio, Ege Köknaroğlu, Kaan Özkara, and Yılmaz Korkmaz for both their ideas and support.

Last but not least, I would like to thank my beloved family for their invaluable support and encouragement.

Contents

List of Figures

List of Tables

List of Acronyms

AE Autoencoder

ANN Artificial neural network

AWGN Additive white Gaussian noise

BER Bit error rate

Bi-GRU Bidirectional gated recurrent unit

Bi-LSTM Bidirectional long-short-term memory networks

BRNN Bidirectional recurrent neural network

CNN Convolutional neural network

CSI Channel state information

DL Deep learning

ISI Intersymbol interference

MIMO Multiple-input multiple-output

ML Machine learning

MLP Multilayer perceptron

MLSE Maximum likelihood sequence estimation

MMSE Minimum mean square error

NN Neural network

ReLU Rectified linear unit

RNN Recurrent neural network

SISO Single-input single-output

SNR Signal-to-noise ratio

Chapter 1

Introduction

1.1 Overview

Future wireless communications is anticipated to bring about major changes along with several developing technologies including 5G, virtual reality, edge computing, and internet of things (IoT). These innovations place unheard of demands on capacity, coverage, latency, efficiency (in terms of power, frequency spectrum, and other resources), flexibility, compatibility, quality of experience, and silicon convergence of wireless communication systems. Due to their capacity to handle large amounts of data, strong nonlinear mapping, and distribution processing capabilities, machine learning (ML) techniques are viewed as promising solutions to meet this challenge. One of the advantages of deep learning (DL)-based methods is that once the model is trained, the classification (or, regression) task can be completed in one shot without the need for additional iterations, incurring only minimal latency. Modern iterative receivers in wireless communication networks, on the other hand, might lengthen the latency, which makes them impractical in some applications such as ultra-reliable low latency communications (URLLC) [1].

Recently, DL-based symbol detectors have attracted significant interest due

to their relatively low-complexity algorithms compared to traditional and model-based ones. Most signal processing algorithms for communication systems have a solid foundation in statistics and information theory, and many are generally demonstrably optimal for traceable mathematical models. Although real systems have a lot of imperfections and non-linearities, simplified models usually have linear, stationary, and Gaussian statistics. DL-based communication systems do not need a mathematically traceable model and can provide better optimization over practical scenarios. Since NNs are universal function approximators [2], functional operations in different blocks of a communication system (e.g., source/channel coding, modulation, channel estimation, detection, equalization) can be processed using DL-based models. While DL-based models are suitable for optimizing these blocks individually or jointly, traditional designs require computationally complex operations for joint optimization.

Channel detection algorithms, which infer the transmitted symbols from a noisy and distorted version of the transmitted signals observed at the receiver are one of the crucial components in the reliable recovery of data transmitted over a communication channel. One of the common problems in wireless communication is to detect the transmitted symbols received through channels that experience inter-symbol interference (ISI) caused by multipath signal propagation. Some of the traditional detectors and decoders require likelihoods and channel state information (CSI) as side information, they may fail to achieve the required performance when these are missing. ML algorithms, however, directly learn the channel detectors or decoders from training data and are shown to be robust to lack of likelihoods or CSI [3]. Another advantage of DL-based algorithms is that the adaptive signal processing capabilities of these algorithms allow them to function in time-varying wireless communication environments. Although the maximum likelihood (ML) detector is optimal in these situations, it involves an exhaustive search. Suboptimal detection algorithms are being implemented with great interest to reduce the computational complexity of typical maximum likelihood detectors because they offer a more flexible accuracy versus complexity trade-off. DL-based models are suitable for use as some of the suboptimal detection algorithms.

Channel equalization is the primary method used in wireless communications to combat the effect of ISI. There have been various channel equalization techniques suggested and implemented that can be used to address the ISI. Channel equalization approaches can be divided into two categories: linear and nonlinear equalization techniques. The most frequently used equalizers within the family of linear equalizers are zero-forcing (ZF) and minimum mean-square error (MMSE) equalizers. Nonlinear equalizers are used whenever channel distortion cannot be resolved using linear equalizers, for instance, for channels with deep spectral nulls. Nonlinear equalizers typically perform better but suffer from higher complexity. The decision feedback equalizer (DFE), the maximum likelihood symbol detection (MLSD) based equalizer, and the maximum likelihood sequence estimation (MLSE) based equalizer are the three most common nonlinear approaches developed [4], [5]. The MLSE equalizer is optimal in terms of minimizing the sequence error but the complexity of the equalizer grows exponentially with the number of channel taps.

The subject of this thesis, the DL-based solutions as channel equalizers, have been recommended to reduce the computational cost of traditional equalizers. Recurrent neural networks (RNNs), multilayer perceptron (MLP), and neural network-based iterative detectors are proposed as DL techniques for channel equalization for different set-ups. In this thesis, we introduce new DL-based ISI equalizers and demonstrate that they are computationally efficient and capable of performing equalization under a variety of channel conditions with the knowledge of the underlying channel taps.

1.2 Thesis Outline

The thesis is organized into five chapters.

In Chapter 2, we provide an overview of the fundamental basis and current state of machine learning techniques with potential wireless communications applications, with a focus on ISI channels and MIMO communications.

In Chapter 3, we first investigate an uncoded digital communication system over discrete-time dispersive channels with ISI and additive white Gaussian noise (AWGN). We offer models for both fixed and time-varying ISI channels. We formulate the channel equalization for ISI channels as a supervised regression problem within a machine learning framework. We outline the design of three DL-based models for channel equalization: Sli-BiLSTM, Sli-MLP, and Sli-Iterative. In addition to these, we present the Sli-GRU and Sli-MLP networks for the fixed ISI channels. In order to evaluate the robustness of the proposed detection algorithms, we also consider the case of imperfect CSI. Based on the idea that each window weight in the final estimation process should have a different weight, we introduce an NN-based window weight network, and the performance of weighted average and direct average based solutions are compared. In the last part of the chapter, extensive numerical results based on the proposed models are compared with the MMSE and Viterbi Algorithm based solutions for channel equalization.

In Chapter 4, we investigate ISI channels for MIMO communication systems, and extend the proposed models to the case of MIMO communications. The bit error rate results of the proposed DL-based models are compared with the results of the Viterbi and MMSE algorithms, and it is found that the proposed models produce close results with the Viterbi Algorithm while they have superior performance compared to the MMSE equalizer.

Finally, in Chapter 5, we summarize our conclusions and offer suggestions for further research.

Chapter 2

Preliminaries and Literature Review

In this chapter, we provide the necessary preliminaries and a literature review. We provide an overview of the applications of machine learning (ML) approaches in wireless communications. We then present an overview of basic neural network (NN) models for deep learning as different NN models are employed in the thesis. Deep learning (DL)-based solutions for intersymbol interference (ISI) channel and multiple input multiple output (MIMO) communications are investigated separately, as they form the basis of Chapters 3 and 4. We also review DL for end-to-end communications and other applications.

The chapter is organized as follows. In Section 2.1, ML algorithms and their connections to wireless communications are explained. Section 2.2 presents NN models for DL. Section 2.3 reviews DL applications for ISI channels. Section 2.4 reviews DL for MIMO communications, especially DetNet is examined in some detail. Section 2.5 reviews DL for end-to-to communications, mostly autoencoder-based solutions. Section 2.5 investigates other DL applications such as channel coding and decoding and modulation classification. The chapter is concluded with a summary in Section 2.7.

2.1 Machine Learning and Wireless Communications

Machine learning algorithms may be categorized in two different ways. We briefly review these categorizations and express their connections to wireless communications problems. The first approach is to categorize machine learning algorithms into classification and regression problems. Classifiers are networks that map each input to a proper class, i.e., a classifier has a finite number of possible outputs. In wireless communications, classifiers may be applied to design detectors [6], demodulators, or decoders [7], [8], [9] where the task is to map the received signal into one of the constellation points or one of the codewords. This approach is useful when a sufficiently accurate mathematical model for the system does not exist, and therefore, calculation of likelihoods or channel state information estimation is not possible. Since traditional detectors and decoders require likelihoods and CSI as side information, they may fail to achieve the required performance in such scenarios. DL algorithms, however, directly learn detectors or decoders from training data and are shown to be robust to lack of likelihoods and CSI. Unlike classification, in regression, the network outputs a continuous value. Regression may be applied in applications such as user localization [10]. In all the aforementioned cases, a machine learning model is trained to minimize a loss function, e.g., mean squared error or cross-entropy. One favorite feature of machine learning-based methods is that as soon as the model is trained, the classification (or regression) task may be performed at one shot without any further iterations and hence with negligible delay. This is in contrast to the state-of-the-art iterative receivers in wireless networks which potentially increase the delay.

The second approach to categorize machine learning algorithms is to view them as supervised, unsupervised, and reinforcement learning algorithms. Supervised learning refers to learning methods where data is labeled; such as detection problems where the dataset specifies the corresponding transmitted signal for each received signal. Unsupervised learning refers to learning methods where data is

not labeled. Unsupervised learning is very powerful in feature extraction, clustering, and dimensionality reduction and has many potential applications in wireless communications. For example, it is known that data transmitted by users in a wireless network are correlated, and if these correlations are properly extracted, it may be possible to optimize different performance metrics [11]. However, for large heterogeneous networks, it is hard, if not impossible, to mathematically model such correlations. Since deep neural networks are known to be exceptionally good in automatic feature extraction [12], they can be applied for such purposes. Automatic modulation classification is another well-known application of unsupervised learning [13], [14] since it is a clustering problem. Data compression is another possible application of unsupervised learning since data compression can be viewed as a dimensionality reduction problem.

Autoencoders are the most common approach to unsupervised learning in DL. Deep NNs require labeled data for training purposes; however, labels are not provided in unsupervised learning. To overcome this problem, the autoencoder is trained to learn the data itself; i.e., it assumes that each data is labeled by itself. When fully trained, the autoencoder's output is a close estimate of its input (ideally it is the same as the input). In addition, the autoencoder is designed in a way that the number of neurons gradually decreases from the first to the middle layer and reaches its minimum in the middle layer, then gradually increases from the middle to the last layer. The genius behind this idea is that if the autoencoder succeeds to reconstruct the input data at the output with sufficient accuracy, then the compact middle layer contains a compressed version of data [15]. Autoencoders are successfully applied to several compression problems in wireless communications; e.g., to compress the CSI at the receiver and feed it back to the transmitter in massive MIMO systems [16].

The structure of an autoencoder has also inspired researchers to explore a more ambitious goal, named end-to-end communications [6],[17], [18], [19], [20]. The ultimate goal of communications is to reconstruct a transmitted message at some other point, known as the receiver, with sufficient accuracy. Researchers made an interesting observation that the first part of an autoencoder, from the first to

the middle layer, may be viewed as a transmitter that integrates all the transmission tasks, including compression, coding, modulation, etc., and jointly optimizes them as a neural network. Furthermore, the second part of the autoencoder may be viewed as a receiver that is optimized jointly with the transmitter. However, we must note that the effect of the communication channel has to be modeled as an additional stochastic layer, and then the network could be optimized via training for end-to-end communication purposes.

In addition to supervised and unsupervised learning, reinforcement learning is also receiving increasing attention in wireless communications. Reinforcement learning is a learning approach where the neural network is trained via interactions of an agent with an environment [21]. At each time, the agent that is in a specific state takes an action from a possible set of actions. The environment responds by rewarding the agent and taking the agent from its current state to another state. Through training, the agent learns what sets of actions will maximize its cumulative reward. Reinforcement learning is applicable to wireless communications problems such as resource allocation [22], where the users act as agents and are trained to optimize some specific metrics. For example, in power control problems, the user may learn how to pick proper transmit powers at each time to maximize the signal to interference plus noise ratio.

2.2 Neural Network Models For Deep Learning

2.2.1 Multilayer Perceptron

Perceptron is the basic unit of a single-layer artificial neural network. It consists of a single artificial nerve cell that can be trained. It is a supervised learning algorithm. A perceptron consists of four parts: input values, weights and biases, weighted sum, and activation function. Given input and output values, the neural network is expected to learn their relation. The mathematical expression of the perceptron, which is the smallest learning unit of artificial neural networks, is as

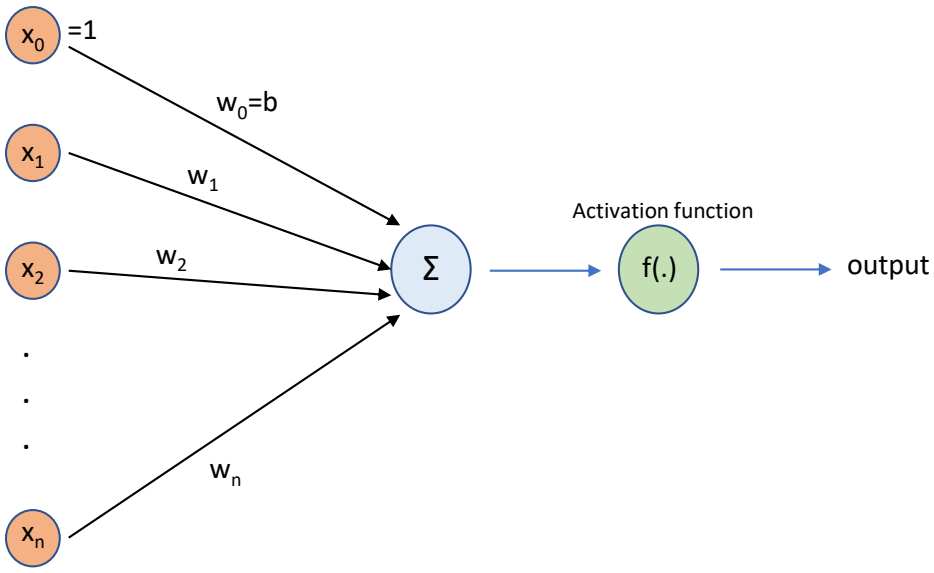

Figure 2.1: Perceptron.

follows:

$$output = f\left(\sum_{i=0}^{n} w_i x_i\right) = f(\mathbf{w} \cdot \mathbf{x}), \tag{2.1}$$

where $\mathbf{w}$ is a vector of real-valued weights, n is the number of features of the input, $\mathbf{w}\cdot\mathbf{x}$ is the dot product, and w_0 is the bias. Fig. 2.1 shows the basic architecture of the perceptron. In order to make an accurate classification using the perceptron model, a threshold value is required. The activation function labels above this threshold value as a class and below it as another class. Perceptron usually allows data to be split into two parts, hence it is also called a linear binary classifier.

The purpose of the perceptron learning algorithm is to create a decision boundary (line) that can correctly classify positive inputs and negative inputs. The aim is to learn weights and bias parameters, which provide the best classification.

We first define the variables to be used in the perceptron learning algorithm:

- $y = f(\mathbf{x})$ denotes the perceptron output when input is vector $\mathbf{x}$.

- $D = \{(\mathbf{x}_1, d_1), \ldots, (\mathbf{x}_s, d_s)\}$ is training set containing s elements, where $\mathbf{x}_j$ is the n-dimensional input vector and d_j is the desired output value (label) for that input.
- $x_{j,i}$ is the value of the ith feature of the j^{th} training input vector, with $x_{j,0}$=1 for bias.
- $w_i(t)$ is the i^{th} value in the weight vector at time t, with $w_0(t)$ being the bias constant.
- r is the learning rate of the perceptron. The learning rate is a tuning variable in an optimization process that determines the step size at each iteration while moving toward the minimum of a loss function.

Algorithm 1: Perceptron learning algorithm

1 Initialize the weights.

2 Perform the following steps over each data input:

(I) Calculate the actual output:

$$\begin{aligned} y_j(t) &= f(\mathbf{w}(t) \cdot \mathbf{x}_j) \\ &= f(w_0(t)x_{j,0} + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}) \end{aligned}$$

(II) Update the weights:

$$w_i(t+1) = w_i(t) + r.(d_j - y_j(t))x_{j,i},\ 0 \leq i \leq n$$

3 The second step should be repeated until the user-specified error threshold is reached or until a predetermined number of iterations have been completed.

$$error = \frac{1}{s}\sum_{j=1}^{s} |d_j - y_j(t)|$$

The multilayer perceptron (MLP) is the most well-known and widely used neural network model. MLP is a feedforward neural network, consisting of three types of layers, namely, the input layer, the output layer, and the hidden layer, as shown in Fig. 2.2. The input layer receives the input signal to be processed.

The hidden layers, located between the input and output layers, are the actual computational engine of the MLP. The neurons in the MLP are trained using the backpropagation learning algorithm. Backpropagation helps in adjusting the weights of the neurons to get an output that is closer to the expected one. MLPs are designed to approximate any continuous function and can also solve problems that are not linearly separable. The main use cases of MLP are pattern classification, recognition, prediction, and approximation.

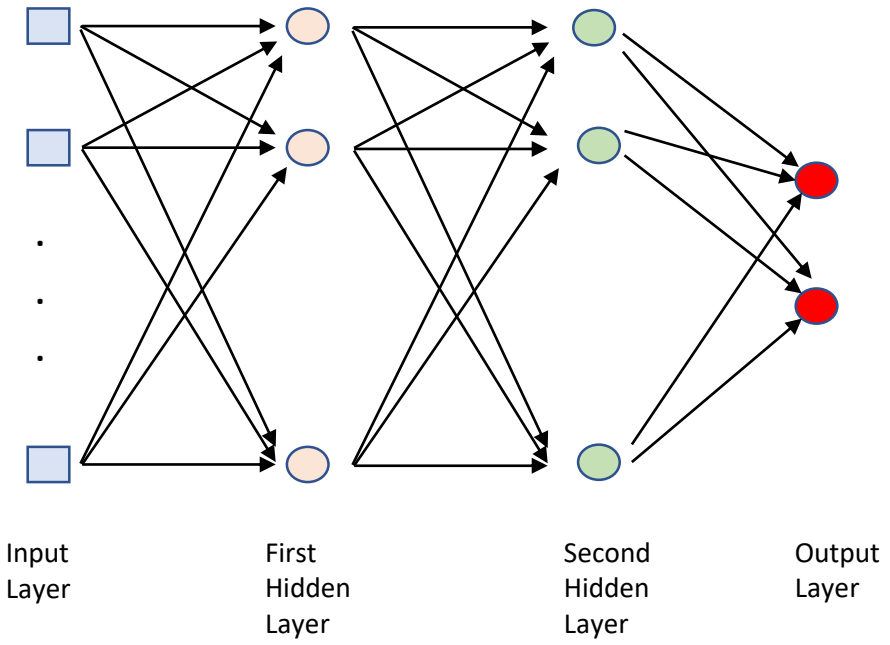

Figure 2.2: An example of an MLP network

2.2.2 Convolutional Neural Networks

Convolutional neural networks are neural networks that use convolution instead of general matrix multiplication in at least one of their layers. Convolutional neural networks are often used to analyze visual information, with common usage areas such as image and video recognition, suggestive systems image classification, medical image analysis, and natural language processing.

2.2.3 Autoencoders

An autoencoder is a type of artificial neural network used for unsupervised learning. The purpose of the autoencoder is to learn a symbolic vector that represents the data. In its general architecture, it includes encoder and decoder modules. While the encoder module creates a representative vector that assimilates the data, the decoder module creates new data again using this representative vector. The problem is to learn the functions $A : \mathbb{R}^n \rightarrow \mathbb{R}^p$ (encoder) and $B : \mathbb{R}^p \rightarrow \mathbb{R}^n$ (decoder) that satisfy

$$\hat{s} = argmin_{A,B}\mathbb{E}[F(x, B \circ A(x))] \tag{2.2}$$

where $\mathbb{E}$ is the expectation over the distribution of x, and F is the reconstruction loss function, which measures the distance between the output of the decoder and the input. Fig. 2.3 illustrates diagram of an autoencoder model.

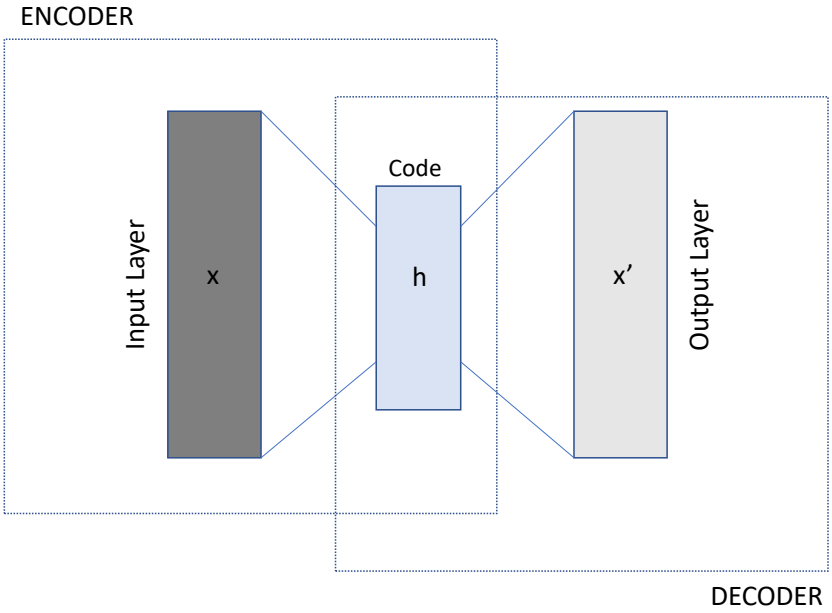

Figure 2.3: Diagram of a simple autoencoder.

2.2.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of artificial neural networks in which the connections among the nodes form a directed loop. This allows the RNN to exhibit dynamic temporal behavior. Unlike the feedforward neural networks, RNNs can use their input memory to process arbitrary sequences of inputs. Fig. 2.4 shows a diagram of an RNN. The main purpose of recurrent neural networks is to process sequential information. RNNs repeat the same task for each element of a sequence, however, the output depends on the previous computations. Another way to think of RNNs is that they contain "memory" that collects information about what has been calculated so far.

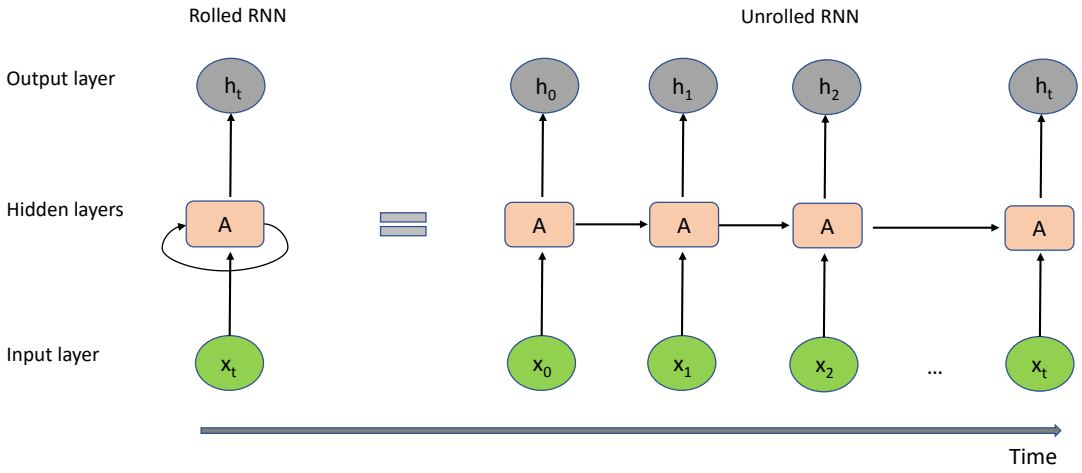

Figure 2.4: Diagram of an RNN.

All RNNs have the form of a chain of repeating neural network modules. In standard RNNs, this repeating module has a very simple structure, such as a single "tanh" layer.

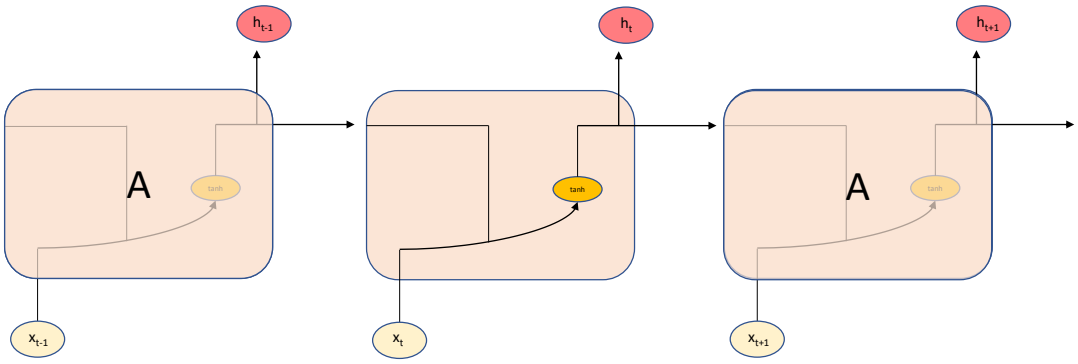

Figure 2.5: The repeating module in a standard RNN.

Fig 2.5 shows the repeating module in a standard RNN. First, x_0 is taken from the input sequence, which results in h_0 as the output. Then, h_0 and x_1 are given as the input for the next step, and so on.

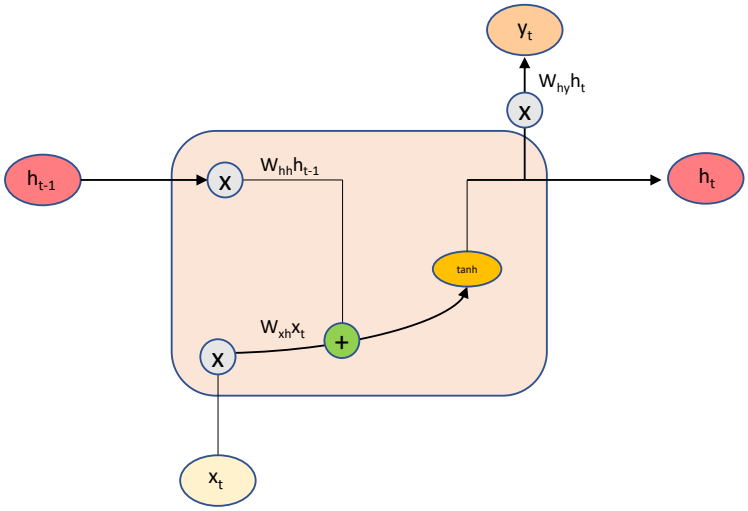

Figure 2.6: Structure of an RNN cell.

The mathematical operations inside an RNN cell are shown in Fig. 2.6, where $\mathbf{W_{hh}}$ contains the weights of the previous hidden state, $\mathbf{W_{xh}}$ contains the weights of the input, $\mathbf{W_{hy}}$ denotes the output weights of the current state, $\mathbf{h}$ is the single hidden vector, $\mathbf{x}$ is the input sequence, $\mathbf{y}$ is the output sequence, *tanh* is the activation function implementing a non-linearity that limits the activations to the range $[-1, 1]$. The output becomes

$$\mathbf{h_t} = tanh(\mathbf{W_{hh}}\mathbf{h_{t-1}} + \mathbf{W_{xh}}\mathbf{x_t}), \tag{2.3}$$

$$\mathbf{y_t} = \mathbf{W_{hy}}\mathbf{h_t}. \tag{2.4}$$

LSTM networks are a modified version of recurrent neural networks that make it easier to remember past data. They were introduced to avoid the problem of long-term dependence. LSTM is well suited for classification, processing, and prediction of time series with an unknown duration such as speech recognition [23], natural language processing [24], image captioning [25], machine translation [26], and time series prediction [27]. Fig. 2.7 shows the repeating module in an LSTM and Fig. 2.8 illustrates the mathematical operations inside the LSTM cell, which includes four neural network layers (dark blue boxes). The notation is as follows:

- $\mathbf{h_t}$, $\mathbf{C_t}$: hidden layer vectors.
- $\mathbf{x_t}$: input vector.
- $\mathbf{b_f}$,$\mathbf{b_i}$,$\mathbf{b_C}$,$\mathbf{b_o}$: bias vector.
- $\mathbf{W_f}$,$\mathbf{W_i}$,$\mathbf{W_c}$,$\mathbf{W_o}$: weight matrices.
- $\odot$: element-wise matrix multiplication.
- $[a, b]$: the concatenation operation.
- σ, $tanh$: activation functions.

In an LSTM cell, three gates are present:

i) Forget gate discovers what details are to be discarded from the block. The sigmoid function decides which values to forget. Its input-output relation can be written as:

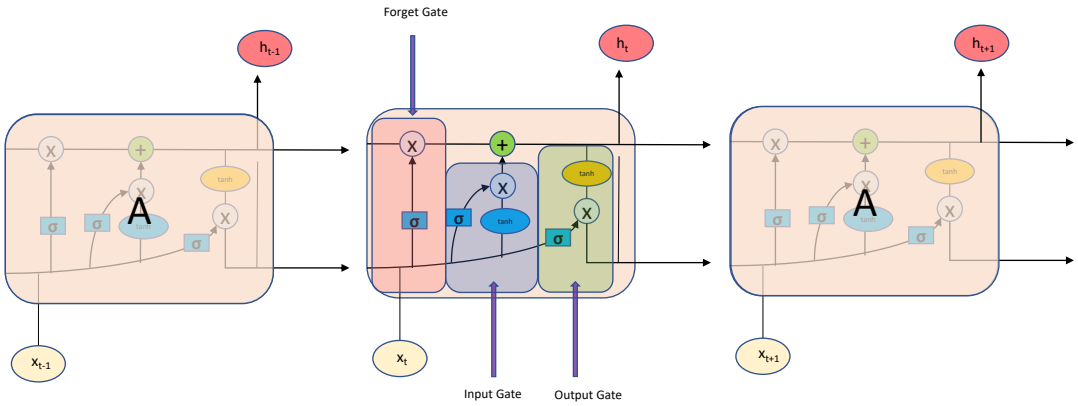

Figure 2.7: The repeating module in an LSTM.

$$f_t = \sigma(\mathbf{W_f} \cdot [\mathbf{h_{t-1}}, \mathbf{x_t}] + \mathbf{b_f}), \tag{2.5}$$

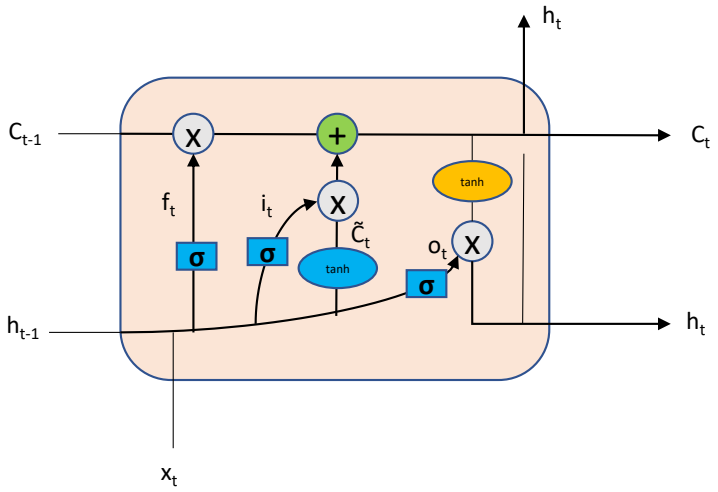

Figure 2.8: Structure of LSTM cell.

ii) Input gate determines which value from the input to use to modify the memory. The sigmoid function decides which values are passed, and tanh function gives weights to the values and decides their importance in the range from -1 to 1, as follows.

$$\mathbf{i_t} = \sigma(\mathbf{W_i} \cdot [\mathbf{h_{t-1}}, \mathbf{x_t}] + \mathbf{b_i}), \tag{2.6}$$

$$\tilde{\mathbf{C}}_\mathbf{t} = tanh(\mathbf{W_C} \cdot [\mathbf{h_{t-1}}, \mathbf{x_t}] + \mathbf{b_C}). \tag{2.7}$$

iii) Output gate filters the cell state as:

$$\mathbf{C_t} = \mathbf{f_t} \odot \mathbf{C_{t_1}} + \mathbf{i_t} \odot \tilde{\mathbf{C}}_\mathbf{t}, \tag{2.8}$$

$$\mathbf{o_t} = \sigma(\mathbf{W_o} \cdot [\mathbf{h_{t-1}}, \mathbf{x_t}] + \mathbf{b_o}), \tag{2.9}$$

$$\mathbf{h_t} = \mathbf{o_t} \odot tanh(\mathbf{C_t}). \tag{2.10}$$

Bidirectional Bi-LSTM networks are based on the idea that the outputs at time t may depend on future elements as well as the previous ones. For example, one would want to look at both left and the right content to guess a missing word in a string. Bidirectional LSTMs can be regarded as two LSTMs stacked on top of each other. The outputs are then calculated based on the latent state of both LSTMs. The Bi-LSTM structure is shown in Fig. 2.9.

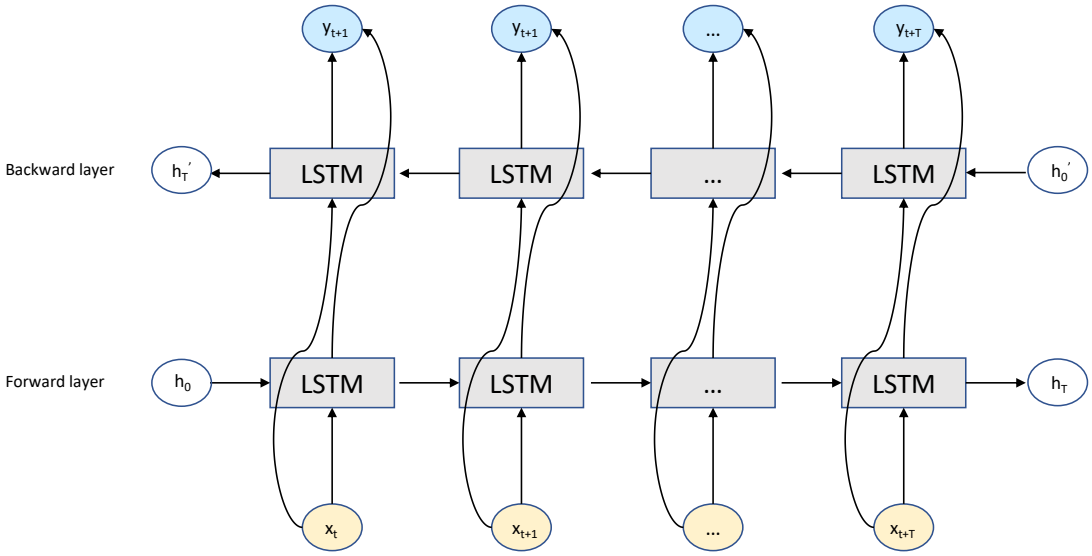

Figure 2.9: Model of a Bi-LSTM network.

2.3 Deep Learning for ISI Channels

In principle, an RNN can map past inputs to each output via a feedback link. This feature can be used for symbol detection over channels inter-symbol interference.

The authors in [3] succeed in detecting the symbols using only the received signal vector over a channel with inter-symbol interference employing a Sliding BiLSTM network. Since the data stream reaching the receiver can be of any length, they propose the idea of using shifting bidirectional RNN (BRNN) to generate the predictions symbol by one symbol as depicted in Fig. 2.10.

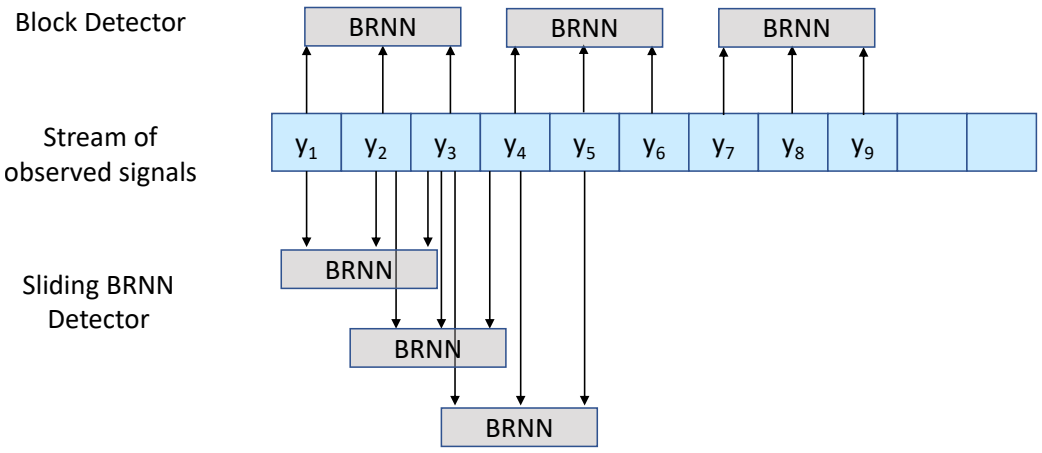

Figure 2.10: The sliding BRNN detector.

The authors in [28] investigate the use of RNNs for symbol detection over magnetic recording channels, which exhibit ISI. They propose a detection method for partial-response equalization, called partial-response neural network (PR-NN). They use Bi-GRUs to recover the ISI channel inputs from noisy channel output sequences, along with a sliding window model as in [3] to process streaming data. The proposed network architecture of PR-NN is depicted in Fig. 2.11.

The authors in [29] propose the so-called ViterbiNet receiver for joint equalization and channel decoding, which simultaneously accounts for both code structure and channel effects to achieve a global optimum with a 3 dB gain compared to the block-based design. ViterbiNet receiver for joint symbol detection and channel decoding is shown in Fig. 2.12. Moreover, a special neural network model is proposed to avoid the need for perfect CSI. This network is shown to be more robust under CSI uncertainty with a gain of 1.7 dB over the Viterbi Algorithm-based receiver using noisy CSI. The ViterbiNet receiver includes three dense layers

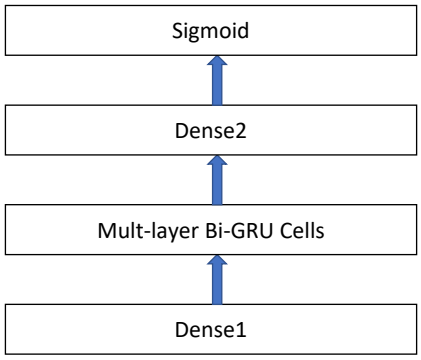

Figure 2.11: The network architecture for proposed PR-NN.

whose activation functions are sigmoid, ReLU, and softmax, respectively. The likelihood function is calculated using the result of ViterbiNet and a probability density function estimator. The channel is assumed to be a tapped delay line model with fixed coefficients.

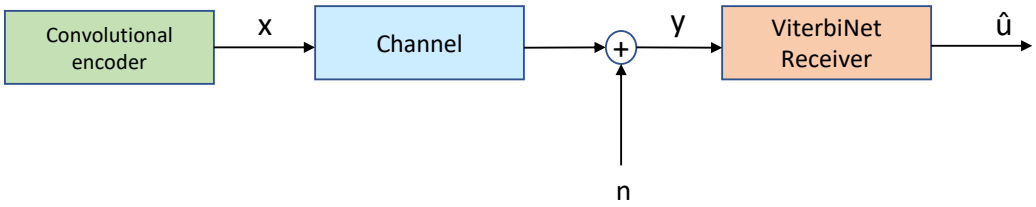

Figure 2.12: ViterbiNet receiver with joint symbol detection and channel decoding.

In many studies, successful predictions are produced for the output by giving the signal vector received through a channel with ISI as input to a BRNN network. The authors [30], [31] evaluate three BRNN models, namely, the bi-LSTM, bi-GRU, and bi-Vanilla-RNN as post-processing units for the compensation of fiber nonlinearities in coherent digital systems carrying polarization. For symbol detection at time t they feed the previous k and following k symbols as inputs to bidirectional RNNs to track inter-symbol dependencies where N stands for the overall length of the sequence with $N = 2k + 1$.

The authors in [32] propose a Bi-LSTM architecture to characterize the ISI

introduced by faster-than-Nyquist (FTN) signaling. Simulation results show that the bit error rate performance of the proposed Bi-LSTM model is close to the theoretically optimal MLSE when the symbol rate is within the Mazo Limit.

The authors in [33] consider the ISI channel $H(z)$, nonlinearities $g(v)$, and AWGN as sources of channel distortion. CNNs are used to recover the transmitted symbol sequence as shown in Fig. 2.13.

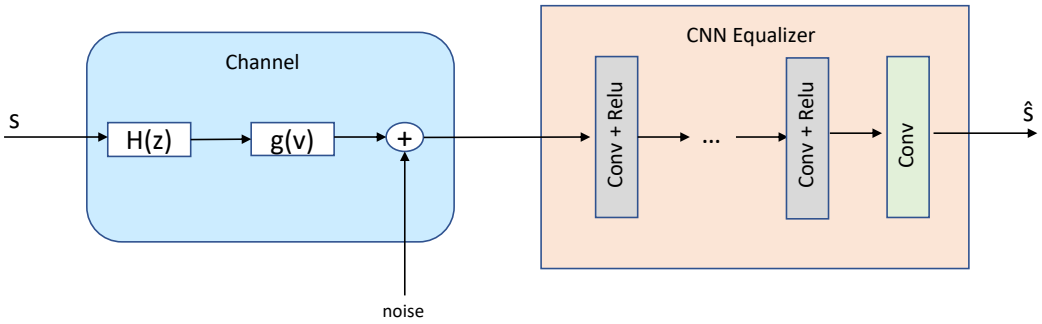

Figure 2.13: The system architecture of CNN equalizer.

The authors in [34] propose autoencoders for end-to-end physical layer communications in the presence of ISI and AWGN. Both the proposed transmitter and the proposed receiver employ Bi-GRU layers. The transmitter learns customized signal constellations for transmission while the receiver performs equalization and demodulation simultaneously.

2.3.1 Sequence Detectors for ISI Channels

The simplest architecture that uses fully connected NN layers for a symbol by symbol detector is shown in Fig. 2.14 (a), and the CNN architecture that can be used to process more complex signals such as images is depicted in Fig. 2.14 (b). In both architectures, The final layer's output has a length of m, (i.e., the cardinality of the symbol set). Note that, the softmax activation is used for the final layer if m is greater than 2. However, the effects of ISI cannot be taken into

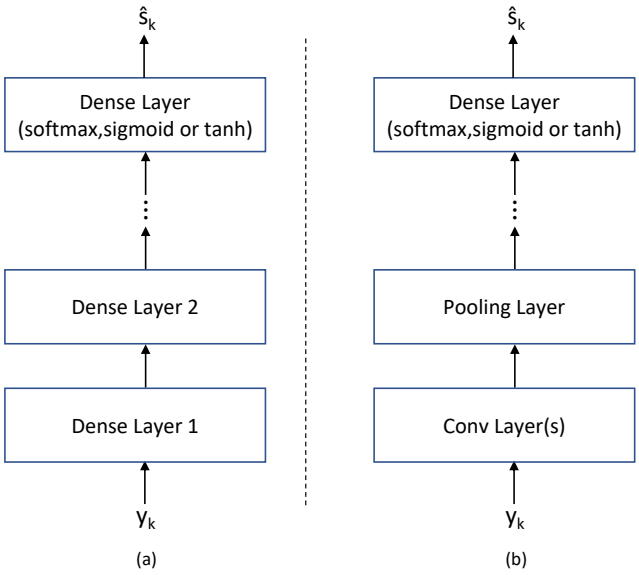

Figure 2.14: Symbol-by-Symbol Detector.

consideration by these symbol-by-symbol detectors. RNNs, which work well for sequence estimation, can be used in this case to perform sequence detection.

Consider a (time) sequence of input data vectors (received vector in this case)

$$\mathbf{y} = \Big[y_1, \quad \cdots, \quad y_T\Big]^T,$$

and a sequence of corresponding output data vectors (transmitted symbols)

$$\mathbf{s} = \Big[s_1, \quad \cdots, \quad s_T\Big]^T,$$

with neighboring data pairs (in time) being correlated. The goal is to learn the rules to predict the output data given the input data using the time sequences and the training data [35].

The block diagram of an RNN is shown in Fig. 2.15. After training, the RNN detector can perform detection on data streams of different lengths, which is one of its key advantages. Since RNNs are feed-forward only, the observations from

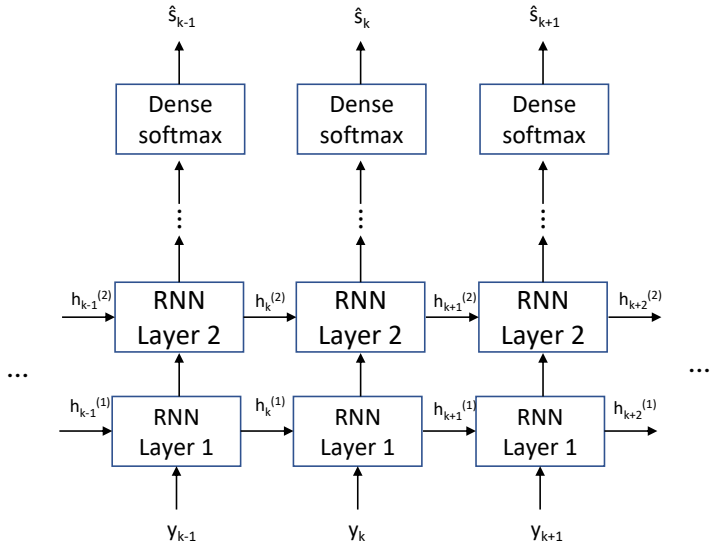

Figure 2.15: RNN architecture for detection.

previous symbols can be used to generate predictions for the k^{th} symbol [3], i.e., one can produce an estimate of s_k as

$$\hat{s}_k = \begin{bmatrix} P_{RNN}(s_k = l_1 \mid y_k, y_{k-1}, \ldots, y_1) \\ P_{RNN}(s_k = l_2 \mid y_k, y_{k-1}, \ldots, y_1) \\ \vdots \\ P_{RNN}(s_k = l_m \mid y_k, y_{k-1}, \ldots, y_1) \end{bmatrix}, \tag{2.11}$$

where P_{RNN} denotes the likelihood of predicting each symbol using the RNN model, and l_i represents the elements in the symbol set. Due to delays in signal arrival as a result of ISI, the received signal during the j^{th} transmission slot, y_j where $j > k$, may contain information about the k^{th} symbol s_k. BRNN can be trained to utilize all available input data in the past and future of a particular time frame to get around the drawbacks of a standard RNN. The concept is to divide a typical RNN's state neurons into two sections: one that is in charge of the positive time direction (forward states), and the other that is in charge of the negative time direction (backward states) [35].

2.4 Deep Learning for MIMO Communications

Samuel et. al. propose deep neural networks for MIMO detection [36],[37]. They design two different types of deep neural networks, a standard fully connected multi-layer network, and a detection network (DetNet) that is designed by unfolding the iterations of a projected gradient descent algorithm into a network. For the varying channel model, their experience with a fully connected MLP network with the channel matrix $\mathbf{H}$, reconfigured as a vector, as the input is reported to be unsuccessful. The dependencies of varying channels were not adequately captured by the network. For this reason, they use the compressed sufficient statistics instead of the received vector directly. That is, the given channel input-output relationship

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{w}, \tag{2.12}$$

where $\mathbf{y}$ is the received vector, $\mathbf{H}$ is the channel matrix, $\mathbf{x}$ is the transmitted symbol vector, and $\mathbf{w}$ is the noise, both sides are multiplied by $\mathbf{H^T}$, i.e.,

$$\mathbf{H}^T\mathbf{y} = \mathbf{H}^T\mathbf{H}\mathbf{x} + \mathbf{H}^T\mathbf{w}, \tag{2.13}$$

is obtained. This suggests that $\mathbf{H}^T\mathbf{y}$ and $\mathbf{H}^T\mathbf{H}\mathbf{x}$ should be the two key components of the architecture. The foundation of their design is a projected gradient descent-like maximum likelihood optimization solution. Such an algorithm would lead to iterations of the form [37]

$$\begin{aligned} \hat{\mathbf{x}}_{\mathbf{k+1}} &= \Pi\left[\hat{\mathbf{x}}_{\mathbf{k}} - \delta_k \frac{\partial ||\mathbf{y} - \mathbf{H}\mathbf{x}||^2}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\hat{\mathbf{x}}_{\mathbf{k}}}\right] \\ &= \Pi[\hat{\mathbf{x}}_{\mathbf{k}} - \delta_k \mathbf{H}^T\mathbf{y} + \delta_k \mathbf{H}^T\mathbf{H}\hat{\mathbf{x}}_{\mathbf{k}}], \end{aligned} \tag{2.14}$$

where $\Pi[.]$ is a nonlinear projection operator, δ_k is a step size, and $\hat{\mathbf{x}}_{\mathbf{k}}$ is the estimate in the k^{th} iteration. Each iteration of DetNet is composed of a linear

combination of the $\mathbf{x}_k$, $\mathbf{H}^T\mathbf{y}$ and $\mathbf{H}^T\mathbf{H}\hat{\mathbf{x}}_\mathbf{k}$ followed by a non-linear projection.

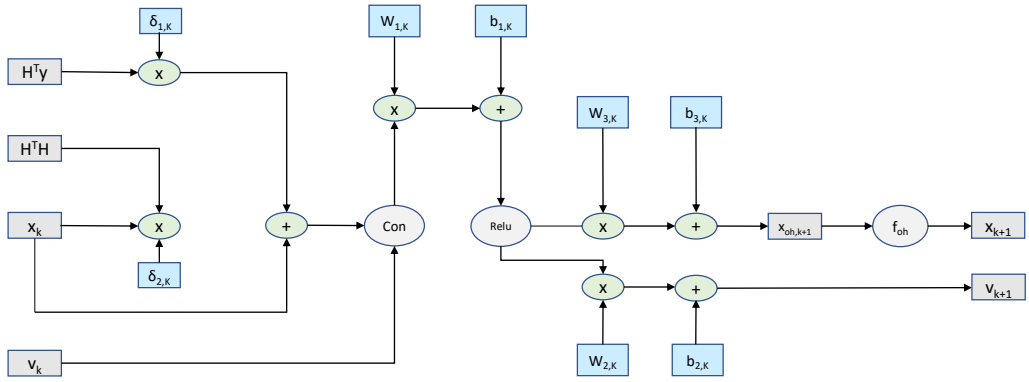

Figure 2.16: A flowchart representing a single layer of DetNet.

Th flowchart of a single layer of DetNet is given in Fig. 2.16, where the learnable parameters are shown in blue boxes. The parameters $\mathbf{W}$ stand for weights and $\mathbf{b}$ stands for biases. The symbol (x) indicates multiplication, (+) indicates the addition operation, f_{oh} indicates the one hot function, con is concatenation, and ReLU is the ReLU activation function. Considering that there are L layers, trainable parameters are as follows:

$$\theta = \{\mathbf{W_{1,k}}, \mathbf{b_{1,k}}, \mathbf{W_{2,k}}, \mathbf{b_{2,k}}, \mathbf{W_{3,k}}, \mathbf{b_{3,k}}, \delta_{1,k}, \delta_{2,k}\}_{k=1}^{L}. \tag{2.15}$$

Simulations demonstrate that DetNet produces sufficiently accurate soft decisions after a single training, with low computational complexity and without any prior knowledge of signal-to-noise ratio (SNR).

Sholev et. al. employ a deep learning method to study the same problem [38]. The structure in a single iteration in DetNet is replaced with layers as in Fig. 3.6, which is explained in detail in Chapter 3. They show that a single trained model is appropriate for the detection of both coded and uncoded data, with or without impairments, for a wide range of SNR levels without prior SNR knowledge [38].

He et. al. investigate the model-driven deep learning for MIMO detection [39]. They design a neural network by unfolding an iterative algorithm. Furthermore,

they investigate joint MIMO channel estimation and signal detection (JCESD), where the detector considers channel estimation impairments. This model significantly improves upon the performance of traditional iterative detectors.

Yan et. al. address the problem of signal detection in MIMO-OFDM with the aid of autoencoders [40]. They employ an autoencoder as a feature extractor along with an extreme learning machine (ELM), for signal classification. Their design obtains higher detection accuracy than several traditional methods while maintaining a similar complexity. O'Shea et. al. introduce a novel physical layer design for single-user MIMO communications based on deep autoencoders [41]. Their design includes a transmitter consisting of an MLP followed by a normalization layer to ensure power constraints, and a receiver including another MLP to decode messages. They discuss how their proposed scheme can be used for open-loop and closed-loop operations in spatial diversity and multiplexing modes with compact binary CSI as feedback. In [42], the authors build autoencoders for both SISO and MIMO systems over flat-fading channels by exploiting the structure of the interference to minimize the symbol error rate. The results are compared with the performance of the standard communication systems with no interference. The performance of the autoencoder-based unsupervised DL communication system is shown to be promising for both single and multi-antenna cases. Wiyaja et. al. employ deep learning to optimize the transmit power levels at the base stations in order to prevent degradation of network performance due to inter-cell interference [43]. Since the capacity of MIMO channels is severely degraded without careful interference management, they suggest employing MLP networks to suppress the resulting inter-cell interference. Their results show that the performance of the proposed network is superior to that of the belief propagation algorithm.

2.5 Deep Learning for End-to-End Communications

O'Shea et. al. discuss several novel applications of deep learning for the physical layer. They model an end-to-end communications system as an autoencoder that jointly learns efficient transmitter and receiver implementations. However, their approach is restricted to short blocklength schemes [17],[6].

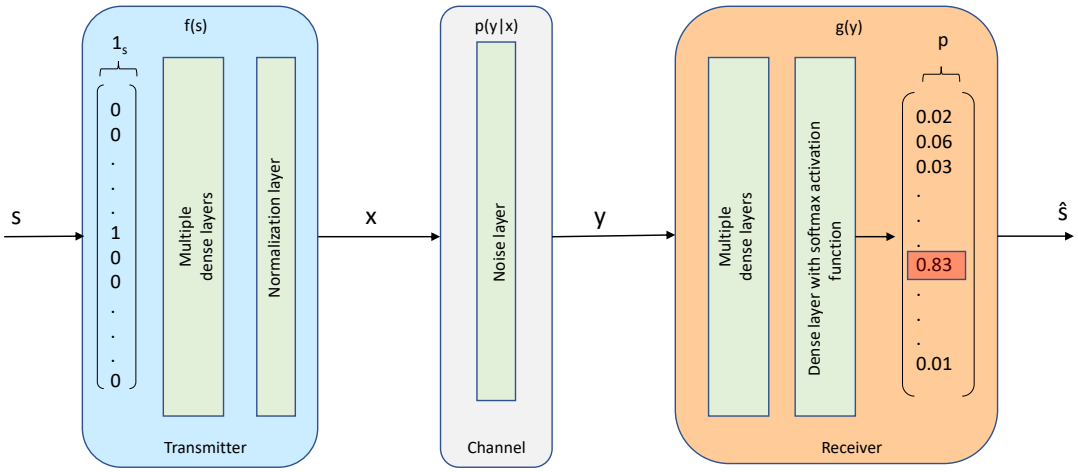

Figure 2.17: A communication system over an AWGN channel is represented as an autoencoder. The input s is encoded as a one-hot vector, and the output is a probability distribution over all possible messages, the most likely of which is chosen as the output $\hat{s}$.

Cammerer et. al. propose a transition from symbol-wise to bitwise autoencoders [18]. Furthermore, they present a fully differentiable neural iterative demapping and decoding structure that achieves promising gains over AWGN channels using a standard 802.11n low-density parity check (LDPC) code.

End-to-end learning of communications systems through NN-based autoencoders requires a differentiable channel model. Hoydis et. al. propose a modified training algorithm that does not require any mathematical model of the channel. The algorithm iterates between the supervised training of the receiver and the

reinforcement learning-based training of the transmitter. They demonstrate that this method works well on AWGN and Rayleigh block fading (RBF) channels. Moreover, their approach can be applied to any type of channel without prior knowledge [19]. Channel modeling is a significant part of the communications systems while evaluating the system performance. As mentioned earlier, most prior designs are based on simplified channel models such as AWGN and Rayleigh fading channels. In order to capture more realistic channel models, O'Shea et. al. utilize generative adversarial networks (GANs) to approximate wireless channel responses. They introduce variational GANs to provide suitable architectures and loss functions that accurately capture stochastic channel behaviors. They also illustrate that a range of different types of stochastic channel models can be accurately learned from measurements [44].

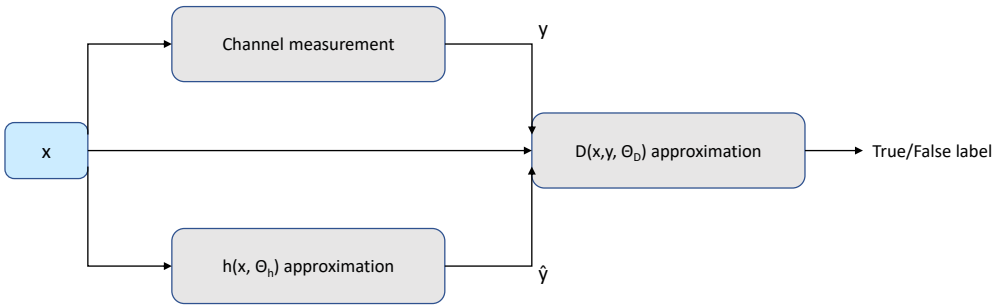

Figure 2.18: Training architecture for conditional variational-GAN based learning of stochastic channel approximation function.

In [20], the transmission medium is modeled as an AWGN channel with additive radar interference. An autoencoder is then applied to design constellations and corresponding receivers for this system, and it is shown that the autoencoder can produce solutions that outperform the results of standard approaches.

2.6 Other Applications

Gruber et. al. use autoencoders for one-shot decoding of random and structured codes, such as polar codes [45]. Their designs demonstrate performances close to

maximum a posteriori (MAP) decoding for short codeword lengths. They further show that when the code is sufficiently structured, neural networks can successfully decode codewords that are not seen during the training process. Therefore, they provide some evidence that NNs can learn a form of decoding algorithm for structured codes. Ninkovic et. al. present a novel autoencoder-based approach for designing codes that provide unequal error protection (UEP) capabilities. Their design is based on switching from a categorical cross-entropy loss function to a weighted categorical cross-entropy loss function. They design codes for both message-wise and bit-wise UEP scenarios. The proposed method is compared with UEP rateless spinal codes and the superposition of random Gaussian codes. In both cases, the autoencoder-based codes show superior performance while providing design flexibility and simplicity [46]. Lyu et al compare the decoding performance of MLPs, CNNs, and RNNs. They conduct experiments using different settings. Their numerical results show that RNNs offer the best decoding performance but require the highest computational burden [47]. Nachmani et. al. introduce neural network architectures for decoding linear block codes with short to moderate lengths. They design RNNs by unfolding belief-propagation (BP) iterations. These architectures yield appreciable improvements over the standard BP and min-sum decoders [48].

O'Shea et. al. perform modulation classification using deep CNNs and LSTM networks, with domain-specific transforms and layer configurations. LSTM is shown to achieve the highest modulation recognition accuracy [49]. Peng et. al. use two different CNN-based pre-trained networks, AlexNet and GoogLeNet, for modulation classification.

2.7 Chapter Summary

In this chapter, we have discussed ML approaches for different communication problems that will be helpful in subsequent chapters. We first examined machine learning techniques and discussed how some (wireless) communication problems can be solved by them. Secondly, we provided a literature review on DL-based

approaches for specific communication problems including transmission over ISI channels and MIMO communications. The rest of the thesis presents our novel studies on this subject.

Chapter 3

Deep Learning Based Channel Equalization for SISO ISI Channels

In this chapter, we use the idea of detecting data streams that can be thousands or even millions in length by processing them using a sliding window. Specifically, three different DL-based methods for ISI channels, namely, Sli-BiLSTM, Sli-MLP, and Sli-Iterative are proposed for time-varying ISI channels based on the sliding window idea. We show that the newly designed ISI detectors are computationally efficient, and can perform detection under perfect or noisy CSI. Based on the idea that each window used in a symbol's estimation process should have a different weight, we further introduce a single-layer NN-based window weight network to learn corresponding window weights. Moreover, we employ Sli-BiGRU and Sli-MLP networks for fixed ISI channels. We evaluate the computational complexity of the proposed models and compare them with the results of the MMSE based equalizers as well as MLSE based solutions.

The chapter is organized as follows. Section 3.1 describes the system model. In Section 3.2, Sliding Bi-LSTM, Sliding MLP, and Sliding Iterative models are

presented for ISI channel equalization. In Section 3.3, we employ Sliding Bi-BiGRU and Sliding MLP, which are more efficient for use with fixed ISI channels. We present our numerical results in Section 3.4, and conclude the chapter in Section 3.5.

3.1 System Model

In this section, we investigate an uncoded digital communication system over discrete-time dispersive AWGN channels with ISI. The corresponding system model is shown in Fig. 3.1, where the transmitter consists of only a modulation module. The input sequence with K information bits is denoted by $x \in \{0,1\}^K$, and s is the modulated signal with BPSK.

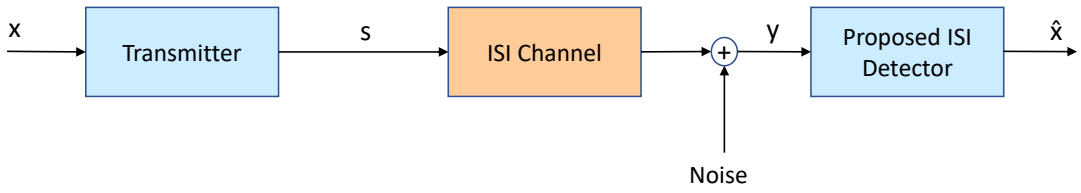

Figure 3.1: System model.

The received vector contains intersymbol interference and AWGN, which can be written as:

$$y_k = \sum_{l=0}^{L-1} g_l s_{k-l} + z_k, \tag{3.1}$$

where z_k is the AWGN distributed as $\mathcal{N}(0, 1/\rho)$, and ρ represents the signal-to-noise ratio (SNR), L is the number of channel taps and g_l is the discrete-time equivalent impulse response of the l^{th} tap. When modeling baseband communication, the channel taps (g_l's) should be modeled as complex random variables. However, we model the channel as real for simplicity and argue that the same

ideas can be extended for complex channels as well. We assume that the l^{th} channel tap has a zero mean Gaussian distribution whose power follows an exponential delay profile, i.e.,

$$g_l \sim \mathcal{N}(0,\, \sigma_l^2), \tag{3.2}$$

with

$$\sigma_l^2 = \frac{e^{-\gamma \times l}}{\sum_{l=0}^{L-1} e^{-\gamma \times l}}, \tag{3.3}$$

where γ is a coefficient that depends on the wireless channel environment as in [50]. If the reflections do not cause much power loss, γ can be chosen small, however, if the reflections result in significant power loss, γ can be chosen large. Note that the selection of this exponential profile is not critical, and the general approach would be applicable for any power delay profile.

The sliding-window model for the received vector is as follows:

$$\mathbf{y_n} = \mathbf{G}\mathbf{s_n} + \mathbf{z_n}, \tag{3.4}$$

where considering the N_1 previous and N_2 following received symbols, we have:

$$\mathbf{y_n} \triangleq \Big[y_{n-N_1}, y_{n-N_1+1}, \ldots, y_{n+N_2-1}, y_{n+N_2}\Big]_{N\times 1}$$

$$\mathbf{s_n} \triangleq \Big[s_{n-N_1-L+1}, s_{n-N_1-L+2}, \ldots, s_{n+N_2-1}, s_{n+N_2}\Big]_{(N+L-1)\times 1} \tag{3.5}$$

$$\mathbf{z_n} \triangleq \Big[z_{n-N_1}, z_{n-N_1+1}, \ldots, z_{n+N_2-1}, z_{n+N_2}\Big]_{N\times 1}.$$

The channel matrix is then defined as:

$$\mathbf{G} = \begin{pmatrix} g_{L-1} & \cdots & g_0 & 0 & \cdots & \cdots & 0 \\ 0 & g_{L-1} & \cdots & g_0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & g_{L-1} & \cdots & g_0 \end{pmatrix}_{N\times(N+L-1)} \tag{3.6}$$

where $N = 1 + N_1 + N_2$ is the window length. Considering the entire sequence of length K, we can write

$$\mathbf{y} = \mathbf{G_{big}}\mathbf{s} + \mathbf{z}, \tag{3.7}$$

where signal vectors are defined as $\mathbf{y} = [y_0, y_1, \ldots, y_{K-1}]$, $\mathbf{s} = [x_0, y_1, \ldots, x_{K-1}]$, $\mathbf{z} = [z_0, y_1, \ldots, z_{K-1}]$, and $\mathbf{G_{big}}$ is given by

$$\mathbf{G_{big}} = \begin{pmatrix} g_0 & \cdots & \cdots & 0 & \cdots & \cdots & 0 \\ g_1 & g_0 & \cdots & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ g_{L-1} & \cdots & g_0 & 0 & \cdots & \cdots & 0 \\ 0 & g_{L-1} & \cdots & g_0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & g_{L-1} & \cdots & g_0 \end{pmatrix}_{K\times K}. \tag{3.8}$$

3.1.1 Fixed vs. Varying Channel Models

We use two different channel models similar to [37] in our work.

i) In the fixed channel scenario, $\mathbf{G}$ is deterministic and constant (or, a realization of a distribution that takes only a single value). The coefficients of the channel (g_l's) do not change over time and all realizations are obtained with the initial channel realization. We take $\{D^{(z)}\}_{z=1}^{Z}$ as a set containing the input symbols, the channel outputs (received vector), and the initial channel taps, i.e., $\{D^{(z)}\} = \{x_{0:K-1}^{(z)}, y_{0:K-1}^{(z)}\, g_{0:L-1}\}$. We suppose that each dataset $D^{(z)}$ is produced from a fixed channel (g_l's are the same for all datasets) with independent additive noise.

ii) In the varying channel scenario, we model the multipath channels with exponential–decay power–delay profiles. We assume that each g_l is a random variable and has a known continuous distribution as given in (3.2). It is either fully known (perfect CSI) or the channel coefficients are estimated at the receiver side with error (noisy CSI). In both cases, delay taps change with each dataset, i.e, $\{D^{(z)}\} = \{x_{0:K-1}^{(z)}, y_{0:K-1}^{(z)}, g_{0:L-1}^{(z)}\}$. Therefore, it is necessary to develop a single detection system for all potential datasets. In other words, the channel is sampled from (3.2) so the network must be able to generalize across the complete distribution of potential channels.

3.1.2 Perfect vs. Noisy CSI

We take into account the following two scenarios to examine the robustness of various detection algorithms against CSI error: (i) perfect CSI, which means that the channel $\mathbf{G}$ is known exactly at the time of detection ($\hat{g}_l = g_l$ for $l = 0, \ldots, L$) (ii) imperfect CSI, where the CSI at the receiver is modeled as [51]

$$\hat{g}_l = g_l + n, \tag{3.9}$$

where n is an additive Gaussian noise distributed as $\mathcal{N}(0, \lambda/\rho)$, where λ coefficient denoting the CSI error level. In this model, as the SNR increases, the CSI is obtained with a smaller error.

3.2 Proposed Deep Learning Based Detectors for Varying Channels

In this section, we formulate the equalization problem for ISI channels as a supervised regression problem in the ML framework. The first step in machine learning is to select a class of possible detectors, also called the architecture. A network architecture is a function $\hat{\mathbf{s}} = f(\mathbf{G}, \mathbf{y}; \theta)$ parameterized by θ. We find the network's parameter θ by minimizing the loss function over the model distribution:

$$\min_{\theta} \mathbb{E}\{loss(\mathbf{s}; f(\mathbf{G}, \mathbf{y}, \theta))\}, \tag{3.10}$$

where the expectation is taken with respect to all the random variables in (3.4), $\mathbf{G}$, $\mathbf{s}$, and $\mathbf{z}$. A loss function that can measure the distance between the network output ($\hat{\mathbf{s}}$) and the true label ($\mathbf{s}$). L_1 and L_2 norms are mostly used in loss functions in machine learning. We use mean squared error (squared L_2 norm) in our simulations. Our goal is to find the best parameters θ of the network architecture which minimize the expected loss over the distribution in (3.4), i.e.,

$$\min_{\theta} \mathbb{E}\{(\mathbf{s} - \hat{\mathbf{s}})^2\}. \tag{3.11}$$

We propose different models for both the fixed and the varying channels. We employ three different NN models for channel equalization. Firstly, we introduce the Sliding Bi-LSTM (Sli-BiLSTM) ISI detector. Secondly, we use the standard multi-layer perceptron (Sli-MLP) rather than Bi-LSTM layers. Finally, Sliding Iterative (Sli-Iterative) is also utilized.

3.2.1 Sliding Bi-LSTM ISI Detector

In this section, we explain the difference between the Bi-LSTM network and the sliding Bi-LSTM network, and describe the use of the Sli-BiLSTM network for

channel equalization. A Bi-LSTM network connects the forward and backward nodes of the LSTM which allows the output to depend on both past and future elements of the received signal vector. The block diagram of such a network was presented in Fig. 2.9. In the Bi-LSTM network, all the information in the received sequence is exploited when generating a prediction for the k^{th} symbol for a sequence of length K, hence $\hat{s}_k$ can be written as

$$\hat{s}_k = f(\mathbf{G}, y_K - 1, y_{K-2}, \ldots, y_0; \theta) \tag{3.12}$$

where $k \leq K$. It is not possible to use this model for data streams whose lengths can be from tens of thousands to millions. Since each element of the received vector contains information for the k^{th} symbol, a re-estimation must be generated for the k^{th} symbol when a new sample of the received signal arrives. Moreover, as the input size increases, the complexity of the model becomes a problem.

For the reasons mentioned above, the sliding Bi-LSTM network is introduced. In this network, we fix the Bi-LSTM length to a certain size, with a maximum length of N. The simplest method would be to detect the stream of data in fixed blocks of length N after training. Although $n \leq N$ values can also be utilized for training, we use the longest value of N in our network design for the best outcome. After detecting N symbols, Bi-LSTM slides ahead by one symbol [3], and the same process is repeated. Fig. 2.10 shows an example of this process with N=3. Note that the N value must be greater than the memory length of the ISI channel. The model evaluates N-length windows on the received vector and generates predictions for the transmitted symbols. One of the disadvantages of this scheme is that the symbols outside the window cannot be used directly, even though they contain information about the symbols within the window. Another disadvantage is that the $(k + N - 1)^{th}$ symbol must be received to complete the estimation of the k^{th} symbol.

We now design an ISI detector to perform channel equalization based on the sliding Bi-LSTM idea. For time-varying channels, the channel coefficients are also given as input, along with the received data sequence. The output of the

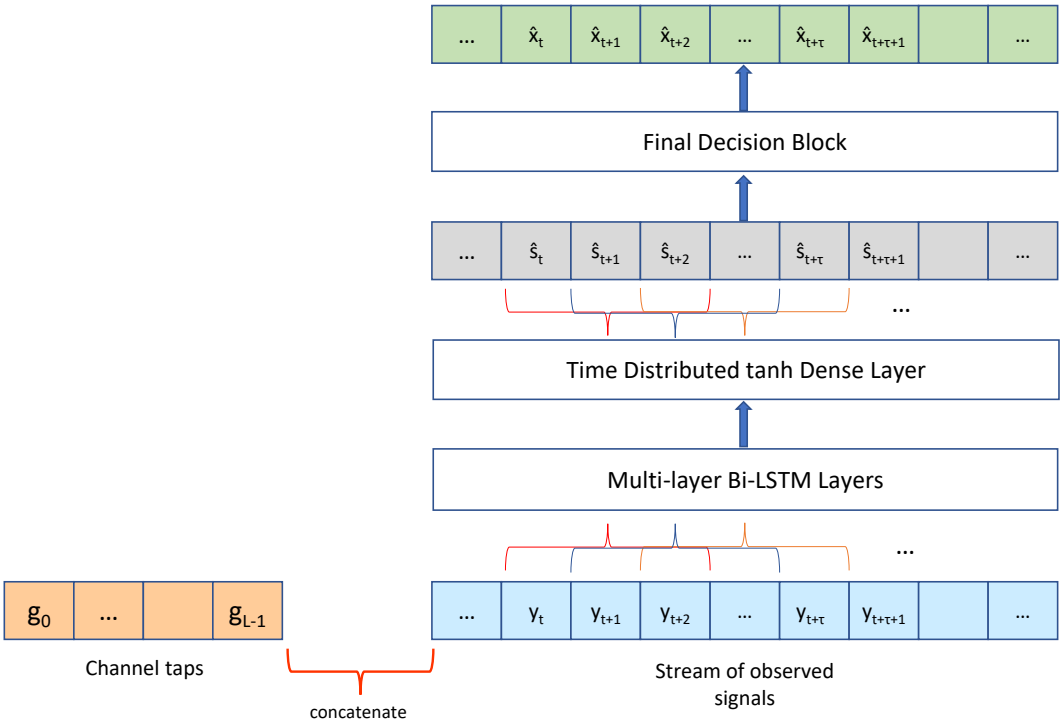

Figure 3.2: The network architecture of Sliding Bi-LSTM ISI detector.

detector is an estimate of the transmitted data sequence. The loss function that the detector is trained on is the mean squared error (MSE) between the transmitted and estimated data sequences. The neural network architecture for our equalizer uses multi-layer Bi-LSTMs. Our network also has a time-distributed dense layer which is used after the Bi-LSTMs to flatten the output by applying the same fully connected dense layer to every time step during LSTM cell unrolling. We consider BPSK modulation which transmits 1 bit per symbol, so there are 2 different classes for each symbol. For this reason, we use tanh activation function instead of the softmax activation function. It is more appropriate to use softmax for higher modulation orders where there are more than two classes. The designed Sliding Bi-LSTM ISI detector is shown in Fig. 3.2.

3.2.1.1 Final Decision Block and Window Weight Network

The set of all feasible beginning positions for a Bi-LSTM detector of length N, such that the detector window overlaps with the k^{th} symbol, is $J_k = \{j \mid j \leq$

$min(k, K-N+1)\, and\, j \geq max(k-N+1, 0)\}$. Let $\hat{s}_k$ be the vector containing the entire predictions generated for s_k:

$$\hat{\mathbf{s}}_k = \begin{bmatrix} \hat{s}_k^{(k-N+1)} \\ \hat{s}_k^{(k-N+2)} \\ \vdots \\ \hat{s}_k^{(k)} \end{bmatrix}, \tag{3.13}$$

where $\hat{s}_k^{(j)}$ the prediction produced by the window starting with $j \in J_k$ and $\hat{w}_k^{(j)}$ is the corresponding window weight for the k^{th} symbol. There are $|J_k|$ different predictions for each bit, whose direct or weighted average produces the final estimate for the k^{th} symbol:

$$\hat{s}_k = sgn\left(\frac{1}{|J_k|}\sum_{j\in J_k} \hat{w}_k^{(j)}\hat{s}_k^{(j)}\right). \tag{3.14}$$

where the sign function is indicated by sgn(.). It seems more reasonable to take the weighted average of these values than to take the direct average because it is expected that the windows that place the k^{th} symbol closer to the center can give better results. We use a single layer at the last stage to find these weights. For a sequence of length K, provided that $N \leq k \leq K-N+1$, the vector containing the predictions of each symbol is given as input to the network, as shown in Fig. 3.3, and the weights of the corresponding windows are learned.

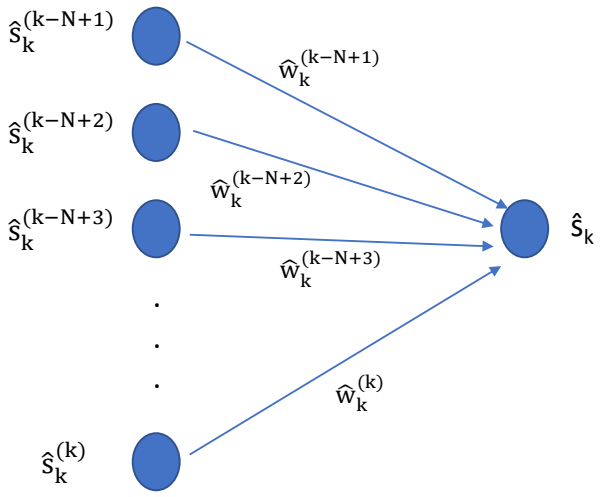

Figure 3.3: The neural network design to find corresponding window weights.

In the case where we take the direct average, all weights are considered equal to one, so the estimate can be written as:

$$\hat{s}_k = sgn\left(\frac{1}{|J_k|}\sum_{j\in J_k}\hat{s}_k^{(j)}\right). \tag{3.15}$$

The final estimate of $\hat{x}_k$ is produced by simply using

$$\hat{x}_k = \begin{cases} 1, & \text{if } \hat{s}_k = 1, \\ 0, & \text{if } \hat{s}_k = -1. \end{cases} \tag{3.16}$$

3.2.1.2 Training Methodology

In this subsection, we describe the training methodology for a specific ISI channel length and a window size to explain how the hyperparameters are selected. Specifically, we perform the hyperparameter search for $L = 5$ and $N = 10$, and we use the selected model for other parameters as well. We use a hyperparameter search [52] since the design space of the hyperparameters is very large. 1920 different models (which are all possible combinations of the search space, i.e., for our

case $5 \times 4 \times 2 \times 4 \times 4 \times 3 = 1920$) need to be tried when grid search is used in the search space even though it is narrowed down beforehand. Random search differs significantly from grid search in that not all the combinations are evaluated, and those that are tested are chosen at random. We only test 30 different models due to the computation time problem, i.e., the random search will randomly sample 30 values to test. We do not give a uniform distribution for each value in the search space because we have an initial idea due to the results in [28], [50] about which values might work better. Although random search does not attempt every possible combination of hyperparameters, it provides a reasonably good performing model in a noticeably shorter amount of time. The search space and the best hyperparameters are summarized in Table 3.1. Although many models give very close results, the choice of training SNR plays an important role in the results of the test part. If it is trained at a low SNR, the model has difficulty in the learning part and cannot produce successful results at high SNR values in the test part. If it is trained at a high SNR, it cannot learn the effect of the AWGN channel sufficiently well. Training at different SNR values gives the most successful results.

Table 3.1: Hyperparameter search for Sli-BiLSTM Model

Parameter	**Search Space**	**Chosen**
# Bi-LSTM layers	2-6	4
# forward-backward states	{25,50,100,200}	100
activation function in the last layer	{sigmoid,tanh}	tanh
training SNR	$\{5,7,\mathrm{U}(7,14),14\}_{dB}$	$\mathrm{U}(7,14)_{dB}$
learning rate	$\{10^{-2}, 10^{-3}, 10^{-3} - 10^{-5}, 10^{-5}\}$	$10^{-3} - 10^{-5}$
batch size	{250,1000,2000}	250

The inputs are fed into four layers of a Bi-LSTM network. In our specific design example, these four layers have sizes of 100, 100, 50, and 100 units for each forward and backward state. The outputs of forward and backward states are concatenated together as a merge mode, it gives twice the state size as input to the next layer. This is then fed into time-distributed dense layers which have the tanh activation function.

The training set includes 5,000,000 different sequences to learn from many different channel distributions. In the training, we start the window from the L^{th} symbol to see the ISI effect in the leading symbols. If the sequence length is K, there are $K-N-L+2$ windows for each sequence. We set the sequence length as $N+L-1$ in the training part, $(\{D^{(z)}\}_{z=1}^{5000000} = \{x^{(z)}_{0:N+L-2}, y^{(z)}_{0:N+L-2}, g^{(z)}_{0:L-1}\})$. In this case, there is only one window that needs to be trained for the sequence in each dataset.

In Fig. 3.4, we show how the training dataset is obtained. For each sequence, SNR in dB was sampled from a uniform distribution $U(7, 14)$ for training. Training is done using the Adam optimizer with a piecewise constant decay scheduler, assigning a learning rate of 0.001 to the first 75% of steps and a learning rate of 0.00001 to the subsequent steps.

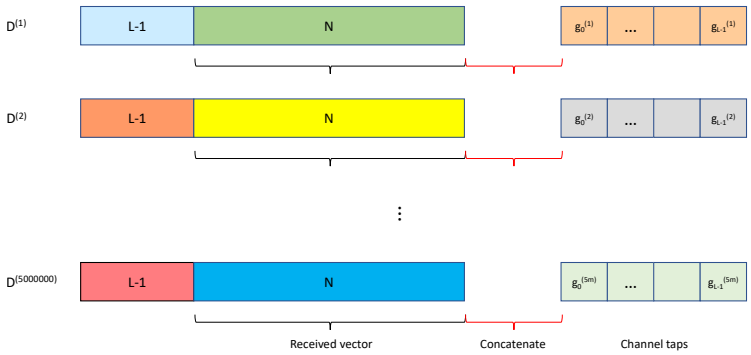

Figure 3.4: Training dataset.

3.2.2 Sliding MLP ISI Detector

In this part, we use MLP instead of the multi-layer Bi-LSTM for channel equalization. The size of the input vector is the sum of the window length and the number of delay taps $(N+L)$, while the length of the output vector is only the length of the window (N) as in the Sli-BiLSTM model. The resulting network architecture of the Sli-MLP model is shown in Fig. 3.5. The Sli-MLP network contains seven fully connected dense layers and activation functions are as shown

in Fig. 3.5.

The reason we use this model is that MLP has a relatively small number of parameters to optimize. For the specific networks designed for $L = 5$ and $N = 10$, the search space and the chosen hyperparameters are summarized in Table 3.2. As the memory length of the ISI channel (L) increases, the deeper MLP model gives better results up to a point. When we deepen the model shown in Fig. 3.5, the network converges to almost the same point, so we decide to use this layer arrangement with 200 neurons in each hidden layer . Other training settings and creation of training dataset ($\{D^{(z)}\}_{z=1}^{5000000} = \{x^{(z)}_{0:N+L-2}, y^{(z)}_{0:N+L-2}, g^{(z)}_{0:L-1}\}$) are the same as those in the previous section.

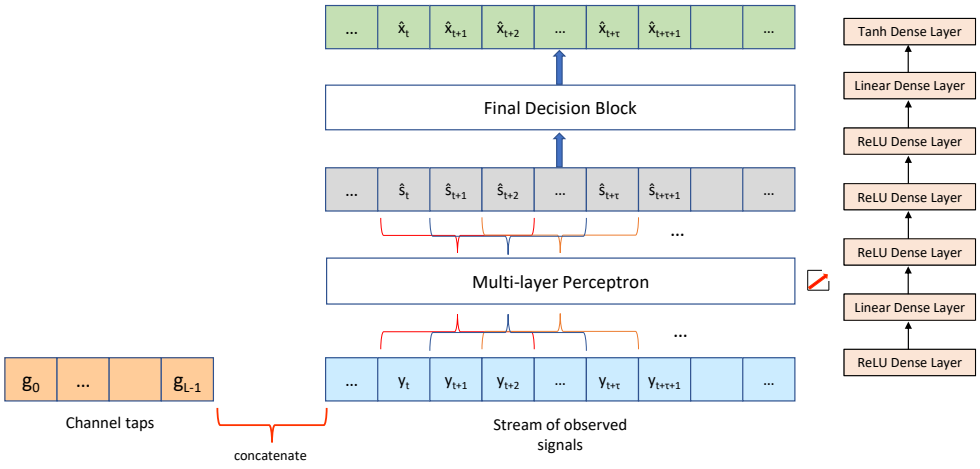

Figure 3.5: The network architecture of Sliding MLP ISI detector.

3.2.3 Sliding Iterative ISI Detector

We take a similar approach as in [37], [38] and use an iterative model as well. The idea is to use a projected gradient approach to solve the detection problem in (3.4). Even though we model the channel as real in our simulations, here we consider a more general case and show how this can be accomplished for complex channel taps. Computing the gradient of (3.4) with respect to $\mathbf{s}$ results in

Table 3.2: Hyperparameter search for Sli-MLP Model

Parameter	**Search Space**	**Chosen**
# MLP layers	3-9	7
# neurons in layers	{25,50,100,200}	200
activation function	{ReLU,linear,sigmoid}	ReLU
training SNR	$\{5,7,\mathrm{U}(7,14),14\}_{dB}$	$\mathrm{U}(7,14)_{dB}$
learning rate	$\{10^{-2}, 10^{-3}, 10^{-3}-10^{-5}, 10^{-5}\}$	$10^{-3}-10^{-5}$
batch size	{250,1000,2000}	250

$$\frac{\partial||\mathbf{y}-\mathbf{G}\mathbf{s}||^2}{\partial \mathbf{s}} = -[\mathbf{G}^*(\mathbf{y}-\mathbf{G}\mathbf{s})], \tag{3.17}$$

where $\mathbf{G}^*$ indicates the conjugate transpose operation. The design is based on mimicking the projected gradient descent as a solution to maximum likelihood optimization. Such an algorithm suggests building a solution in an iterative way, using

$$\mathbf{s}_{k+1} = \Pi(\theta_1 \hat{\mathbf{s}}_k + \theta_2 \hat{\mathbf{G}^*\mathbf{y}} + \theta_3 \mathbf{G}^*\mathbf{G}\hat{\mathbf{s}}_\mathbf{k}), \tag{3.18}$$

where θ_1,θ_2,θ_3 are learnable parameters and Π is a nonlinear function. The unknown value $\hat{\mathbf{s}}_\mathbf{0}$ is started randomly to calculate the following input features:

$$\begin{pmatrix} Re\{\hat{\mathbf{s}}_\mathbf{0}\} \\ Im\{\hat{\mathbf{s}}_\mathbf{0}\} \end{pmatrix}, \begin{pmatrix} Re\{\mathbf{G}^*\mathbf{y}\} \\ Im\{\mathbf{G}^*\mathbf{y}\} \end{pmatrix}, \begin{pmatrix} Re\{\mathbf{G}^*\mathbf{G}\hat{\mathbf{s}}_\mathbf{0}\} \\ Im\{\mathbf{G}^*\mathbf{G}\hat{\mathbf{s}}_\mathbf{0}\} \end{pmatrix}. \tag{3.19}$$

We use a real-valued neural network model so we split each input feature into its real and imaginary parts and concatenate them. The iterative model outputs soft probabilities for the real and imaginary parts of each of the transmitted symbols in each iteration:

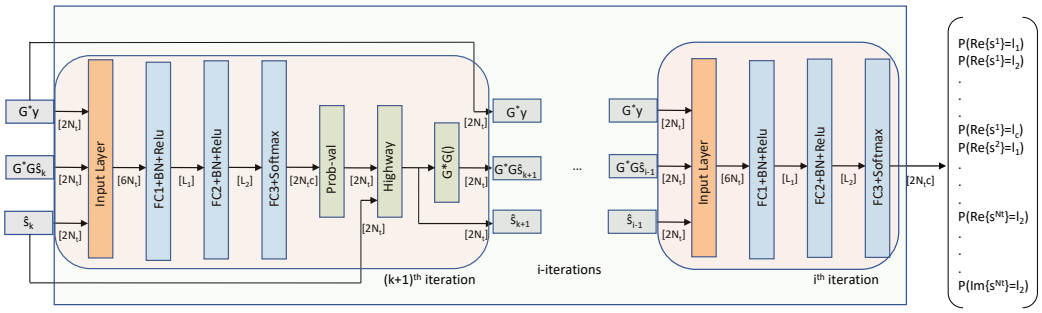

Figure 3.6: The network architecture of Iterative Detector.

$$\begin{pmatrix} P(Re\{s^1\} = l_1) \\ P(Re\{s^1\} = l_2) \\ \vdots \\ P(Re\{s^1\} = l_c) \\ P(Re\{s^2\} = l_1) \\ \vdots \\ P(Re\{s^{Nt}\} = l_c) \\ \vdots \\ P(Im\{s^{Nt}\} = l_c) \end{pmatrix}. \tag{3.20}$$

The values that the real and imaginary part may take are symbolized as l_1,l_2,...l_c where c = $\sqrt{|M|}$. As we mentioned before, when we only take the real parts, vector dimensions indicated by $2N_t$ ($N_t = N + L - 1$) in Fig. 3.6 will be replaced by N_t.

At the k^{th} iteration, the model performs the following operations:

(i) The input values enter two fully connected layers with batch normalization and ReLU activation function.

(ii) The outputs from the ReLU layer enter a fully connected softmax layer and the result in (3.20) is obtained.

(iii) The output probabilities go into a block called "probabilities to values", which performs the following operations:

$$Re\{\hat{s}^n\} = \sum_{d=1}^{c} l_d P(Re\{s^n\} = l_d), n = 1, 2, ..., N_t. \tag{3.21}$$

$$Im\{\hat{s}^n\} = \sum_{d=1}^{c} l_d P(Im\{s^n\} = l_d), n = 1, 2, ..., N_t.$$

(iv) Highway layers have been used to facilitate gradient flows in very deep neural networks as in [53]. A new estimate ($\hat{\mathbf{s}}_\mathbf{k}$) is obtained by combining the estimate from the previous iteration ($\hat{\mathbf{s}}_{\mathbf{k-1}}$) with the soft output from the "probabilities to values" block. That is, the highway layers perform the following operations:

$$\hat{\mathbf{s}}_\mathbf{k} = \Pi(\hat{\mathbf{s}}_{\mathbf{k-1}}, \mathbf{G}^*\mathbf{y}, \mathbf{G}^*\mathbf{G}\hat{\mathbf{s}}_{\mathbf{k-1}}, \mathbf{W_k}) \cdot \sigma(\hat{\mathbf{s}}_{\mathbf{k-1}}, \mathbf{W_k^1}) + \hat{\mathbf{s}}_{\mathbf{k-1}} \cdot (1 - \sigma(\hat{\mathbf{s}}_{\mathbf{k-1}}, \mathbf{W_k^2})), \tag{3.22}$$

where $\mathbf{W_k^1}$ and $\mathbf{W_k^2}$ are all learnable parameters of the k^{th} iteration.

(v) After $\hat{\mathbf{s}}_\mathbf{k}$ is obtained, it is multiplied with $\mathbf{G}^*\mathbf{G}$ to obtain the missing input feature. Finally, the input features $\mathbf{G}^*\mathbf{y}$, $\mathbf{G}^*\mathbf{G}\hat{\mathbf{s}}_\mathbf{k}$, $\hat{\mathbf{s}}_\mathbf{k}$ are concatenated and fed-back to the $(k+1)^{th}$ iteration as shown in Fig. 3.6.

3.2.3.1 Training Methodology

In this subsection, we consider the same set-ups as in Section 3.2.1.2 and describe the training methodology for a specific ISI channel length and a window size to explain the optimization of hyperparameters. That is, we perform the hyperparameter search for $L = 5$ and $N = 10$, and we use the selected model for other parameters as well. The search space and the chosen hyperparameters for training are summarized in Table 3.3. The network parameters L_1 and L_2 in Fig. 3.6 are chosen as $L_1 = 200$ and $L_2 = 100$. Batch normalization was applied before the ReLU layers and a 0.2 rate dropout layer was applied after the first ReLU layer in each iteration. In cases where the number of channel taps is not high (e.g., $L = 3, 5$), we found that 10 iterations with 200 neurons in the first layer and

100 neurons in the second layer provide the best performance-complexity trade-off. Training set includes 5,000,000 different $(N + L - 1)$ length sequences. As in (3.6), 5 million different $\mathbf{G}$ matrices are produced and the training dataset is obtained, $\{D^{(z)}\}_{z=1}^{5000000} = \{x_{0:N+L-2}^{(z)}, y_{0:N-1}^{(z)}, \mathbf{G^{(z)}}\}$. For each sequence, SNR in dB is sampled from the $U(7, 14)$ distribution for training. Training is done using an Adam optimizer with a piecewise constant decay scheduler as in the Sli-BiLSTM model. The MSE loss function given by

$$\left[\sum_{j=1}^{N_t=N+L} \sum_{q=1}^{c} (\mathbb{1}\{Re\{s^j\} = l_q\} - P(Re\{\hat{s}^j\} = l_q))^2 + \sum_{j=1}^{N_t=N+L} \sum_{q=1}^{c} (\mathbb{1}\{Im\{s^j\} = l_q\} - P(Im\{\hat{s}^j\} = l_q))^2 \right] \tag{3.23}$$

.

is used. Here $\mathbb{1}$ denotes the indicator function which takes on the value of 1 when its argument is true and 0 when it is false.

Table 3.3: Hyperparameter search for Sli-Iterative Model

Parameter	**Search Space**	**Chosen**
# iterations	{5,10,20,50}	10
# neurons (L_1)	{25,50,100,200}	200
# neurons (L_2)	{25,50,100,200}	100
training SNR	$\{5,7,\text{U}(7,14),14\}_{dB}$	$\text{U}(7,14)_{dB}$
learning rate	$\{10^{-2}, 10^{-3}, 10^{-3} - 10^{-5}, 10^{-5}\}$	$10^{-3} - 10^{-5}$
batch size	{250,1000,2000}	1000

3.2.3.2 Test Methodology

In this subsection, we examine an example for $N = 3$, $L = 3$, and $K = 10$ to show the algorithm of the Sli-Iterative model in the test part. For these values, the $\mathbf{G}$ and $\mathbf{G_{big}}$ matrices become

$$\mathbf{G_{big}} = \begin{pmatrix} g_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_1 & g_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_2 & g_1 & g_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & g_2 & g_1 & g_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_2 & g_1 & g_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g_2 & g_1 & g_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_2 & g_1 & g_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & g_2 & g_1 & g_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_2 & g_1 & g_0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g_2 & g_1 & g_0 \end{pmatrix}_{10\times 10}, \tag{3.24}$$

$$\mathbf{G} = \begin{pmatrix} g_2 & g_1 & g_0 & 0 & 0 \\ 0 & g_2 & g_1 & g_0 & 0 \\ 0 & 0 & g_2 & g_1 & g_0 \end{pmatrix}_{3\times 5}. \tag{3.25}$$

When we shift the values in (3.5) to the right by one symbol, the $\mathbf{G}$ matrix does not change. For the new $\mathbf{y_n}$ and $\mathbf{s_n}$ values in (3.5), we have to generate predictions for the $\mathbf{s_n}$ vector by running the model in Fig. 3.6. It is necessary to perform this operation $(K - N - L + 2) = 6$ times to make predictions for that part of the sequence. Corner samples seem to be a problem for this short block length, but in reality, there are $N + L - 1$ estimates for almost all the samples as the sequence length increases.

The set of all feasible beginning positions for a Sli-Iterative detector of length $(N + L - 1)$, such that the detector overlaps with the k^{th} symbol, is $J_k = \{j \mid j \leq min(k + L - 1, K - N)\ and\ j \geq max(k - N + 1, L - 1)\}$. Moreover, $\hat{s}_k^{(j)}$ indicates the prediction produced by the window starting with $j \in J_k$ $P(Re\{\hat{s}^k\} = 1)$, $\hat{w}_k^{(j)}$ is the corresponding window weight for the k^{th} symbol, x_k indicates the final predictions for transmitted symbols in Fig. 3.1 for $k = 0, 1, ..., K - 1$. The final decision block performs the following operation:

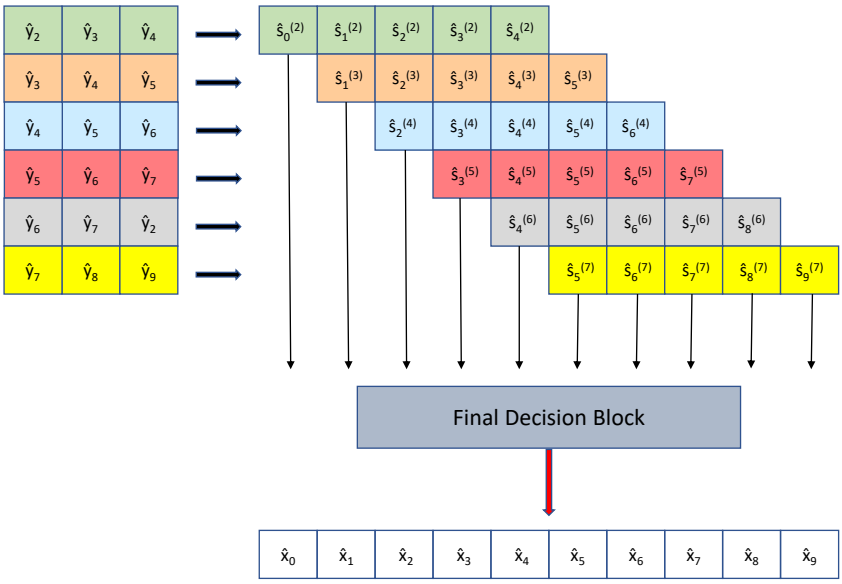

Figure 3.7: Sliding estimates.

$$\hat{s}_k = sgn\left(\left(\frac{1}{|J_k|}\sum_{j_k}\hat{w}_k^{(j)}\hat{s}_k^{(j)}\right) - 0.5\right). \tag{3.26}$$

3.3 Proposed Deep Learning Based Detectors for Fixed Channels

For the fixed channel model, the channel coefficients do not need to be given as the input. The complexity of the problem and model is less than the varying channel case so we use a simpler Bi-GRU network rather than a Bi-LSTM network. The network architecture of the Sliding Bi-GRU ISI detector is shown in Fig. 3.8. We use three Bi-GRU layers with the state sizes of 100, 50, and 100 units, respectively, for each forward and backward state. Moreover, we use the Sli-MLP model, which is also used for the varying channel model, without giving the channel taps as input. The training settings are the same as those of the models used for the varying channels.

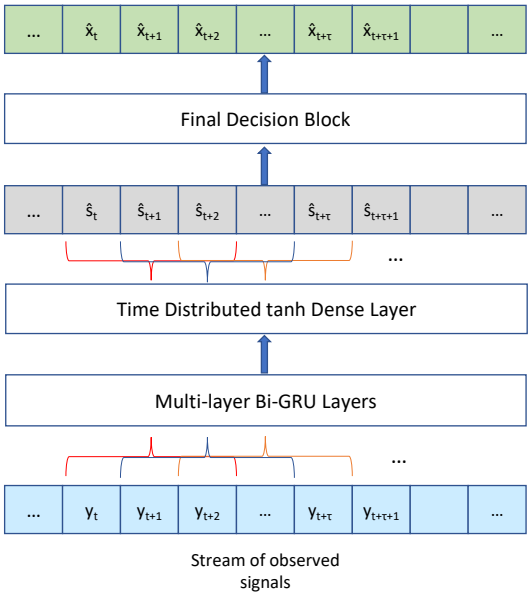

Figure 3.8: The network architecture of Sliding Bi-GRU ISI detector for fixed channels.

3.4 Numerical Experiments

3.4.1 MMSE and Viterbi Algorithm Results

The proposed ISI detectors are compared with the results of MMSE equalization and the maximum likelihood sequence detection implemented via the Viterbi Algorithm.

Consider the model

$$\mathbf{y} = \mathbf{G}\mathbf{s} + \mathbf{z}, \tag{3.27}$$

- MMSE solution is as follows:

$$\hat{\mathbf{s}} = argmin_{\mathbf{s}} ||\mathbf{y} - \mathbf{G}\mathbf{s}||^2, \tag{3.28}$$

$$\hat{\mathbf{s}} = \mathbf{G}^{\mathbf{H}}(\mathbf{G}\mathbf{G}^{\mathbf{H}} + \mathbf{I}/\rho)^{-1}\mathbf{y}. \tag{3.29}$$

- The Viterbi Algorithm is the MLSE detector, which minimizes the probability of sequence error (assuming equiprobable transmitted symbols) with independent and identically distributed Gaussian noise at the receiver. Namely, it efficiently computes

$$\hat{\mathbf{s}} = argmin_{(\mathbf{s}\in S^{N_t})}||\mathbf{y} - \mathbf{G}\mathbf{s}||^2, \tag{3.30}$$

  where the minimization is over $\mathbf{s} \in S^{N_t}$; i.e., over all possible transmitted vectors.

  In the Viterbi Algorithm, the number of cost values to be stored is equal to the number of nodes in the trellis, which is equal to the number of possible states 2^L times the number of time instances, which is equal to the sequence length for a single-input single-output (SISO) system. Therefore, the complexity of the Viterbi Algorithm is $O(K2^L)$ without including floating point operation calculations, which are linear in the number of time instances K but grow exponentially with memory length L. One of the main issues with the Viterbi Algorithm is that we might run out of memory when writing the program for this decoder if there are a large number of channel taps, which causes a storage complexity issue.

  On the other hand, the computational complexity of the proposed models (Sli-BiLSTM, Sli-MLP, and Sli-Iterative) is $O(N(K - N - L + 2))$ because the sliding detector of length N needs to process $(K - N - L + 2)$ times to scan the entire sequence as in (3.24). As a result, the computational complexity for the conventional Viterbi Algorithm increases exponentially with memory length L; however, the proposed DL-based detectors are linear in window length and sequence length.

3.4.2 Complexity Analysis of Proposed Detectors

We can determine the total amount of calculations the model will need to carry out in order to estimate the inference time for that model. The term floating point operation (FLOP) is used here. Any operation involving a floating point value, including addition, subtraction, division, and multiplication, is included in this category. The total number of FLOPs will give us the complexity of the proposed detectors. We calculate FLOPs approximately for each model. The explanation of how we count the number of FLOPs is as follows:

- Any operation involving a floating point value, including addition, subtraction, division, and multiplication are counted as 1 FLOP.
- The number of flops in a fully dense layer network with input dimension A and output dimension B is $B(2A + 1)$.
- The matrices are of the form $(n \times p)$ and $(p \times m)$, the number of FLOPs is equal to $nm(2p - 1)$.
- The LSTM cell contains 4 fully dense layers and 4 element-wise matrix multiplications, the total number of FLOPs is $(4B(2(A+B)+1)+4A)$, where A is the number of hidden units of LSTM and B is the input dimension. The number of FLOPs in Bi-LSTM is also twice that.
- The GRU cell contains 3 fully dense layers and 4 element-wise matrix multiplications, the total number of FLOPs is $(3B(2(A + B) + 1) + 4A)$, where A is the number of hidden units of GRU and B is the input dimension. The number of FLOPs in Bi-GRU is also twice that.

In Table 3.4, we calculate the total number of FLOPs of the proposed models for $L = 3$ and $N = 10$. Since the final decision block is almost the same for all models, we neglect these FLOPs. When we compare the models, we see that the Sli-MLP model has the lowest complexity. On the other hand, the Sli-BiLSTM model is the most complex detector. We use the Sli-BiGRU detector only for

Table 3.4: Detection complexity comparison

Detector	**Calculations**	**# FLOPs**
Sli-BiLSTM	$[100(2(100+1)+1)\times 4) + 100 \times 4] \times 2+$ $[100(2(200+100)+1)\times 4) + 100 \times 4] \times 2+$ $[50(2(200+50)+1)\times 4) + 50 \times 4] \times 2+$ $[100(2(100+100)+1)\times 4) + 100 \times 4] \times 2$	1.167.200
Sli-MLP	$200(2\times 13 + 1) + 200(200 \times 2 + 1) \times 5+$ $10(200\times 2 + 1)$	410,410
Sli-Iterative	$10[200(2\times 36 + 1) + 100(2 \times 200 + 1)+$ $24(2\times 100 + 1) + 12(2 \times 24 - 1) + 3 \times 12]$	601,240
Sli-BiGRU	$[100(2(100+1)+1)\times 3) + 100 \times 4] \times 2+$ $[50(2(200+50)+1)\times 3) + 50 \times 4] \times 2+$ $[100(2(100+100)+1)\times 3) + 100 \times 4] \times 2$	514,700

fixed channels and it has almost half the FLOPs compared to the Sli-BiLSTM model.

3.4.3 Simulation Results

In this section, we evaluate the performance of the proposed DL-based models for ISI channels. We model the multipath channels with exponential–decay power–delay profiles as given in (3.2) and $\gamma = 5$ in (3.3). Then, we numerically evaluate the performance of the proposed algorithms. The parameters used in the test part are shown in Table 3.5. We can increase the sequence length (over thousands or millions) as long as computer memory allows, but since it takes time to produce the simulation results and we want to see the error rate for many different g_l realizations, we set $K = 200 + 2(L-1)$. We do not include the first and the last $(L-1)$ time instances in the error calculation, since the amount of error in the corner observations is high in a short sequence. This is not necessary in cases where the sequence length is chosen long enough. The evaluation metric is the bit error rate (BER) between input $\mathbf{x}$ and detection result $\hat{\mathbf{x}}$, which is defined

Table 3.5: Test set parameters

Parameter	**Selected Value**
K	200+2(L-1)
γ	1/5
λ	0.4
test set size	5000
test SNR in dB	{0,1,2,. . . ,20}

as

$$BER = \frac{1}{K - 2(L-1)} \sum_{m=L-1}^{K-L} \mathbb{1}_{\hat{x_m} \neq x_m}. \tag{3.31}$$

In Fig. 3.9, we perform simulations for the Sli-BiLSTM network using different window lengths (N) to decide on the optimal value to be used in the models. As the window length increases, the results get closer to the Viterbi Algorithm based equalization. However, as more time instances will be involved, it becomes difficult for the model to resolve the relationship among them, which shows the trade-off between the complexity of the model and the bit error rate. The number of channel taps (L) also plays an important role in the selection of the window length (N), as can be seen in Fig. 3.9, if we choose N more than twice of L ($N > 2L$), we do not observe much difference. With this observation, we perform our simulations for $N = 10$ unless otherwise stated.

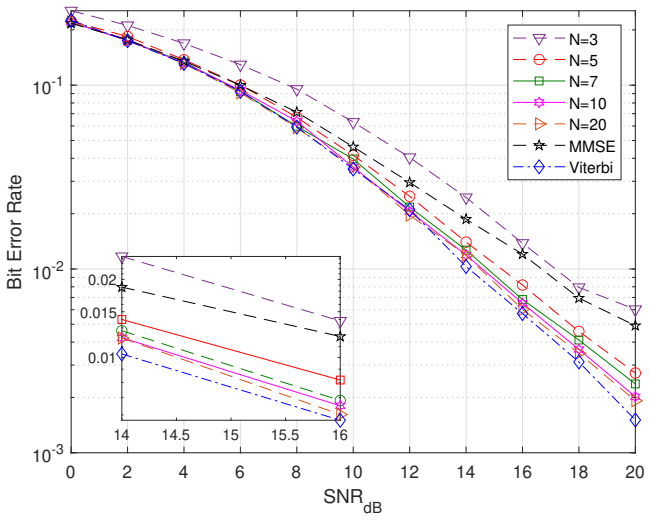

Figure 3.9: BER results of the Sli-BiLSTM model for the varying channels, $L = 3$, and different window lengths.

In Fig 3.10, we compare the bit error rates when using weighted average or direct average in the final decision block. The learned window weights for (3.14) can be seen in Fig. 3.11 to be bell-shaped. The weight of the window becomes larger for the symbols close to the center of the window. Although the weighted average beats the direct average, in our extensive experiments, we observed no significant difference. Even in some simulation results, the weighted average was slightly worse. For this reason, we will use the direct average in the rest of the simulations.

Fig. 3.12 illustrates the bit error rates of all three models. All the proposed models outperform linear MMSE equalizers and give results close to the Viterbi Algorithm based equalization in the SNR range of $0 - 6$ dB. Although a clear distinction cannot be made between the models for the $0 - 6$ dB SNR range, the Sli-BiLSTM models have the best performance. The Sli-Iterative model performs worse than Sli-MLP even though its structure is more complex. When the window length (N) is selected as 10 for all the models, the Sli-BiLSTM model gives better results than the others. As we mentioned before, if we increase the window length (N), the result approaches those of the Viterbi Algorithm. For this reason,

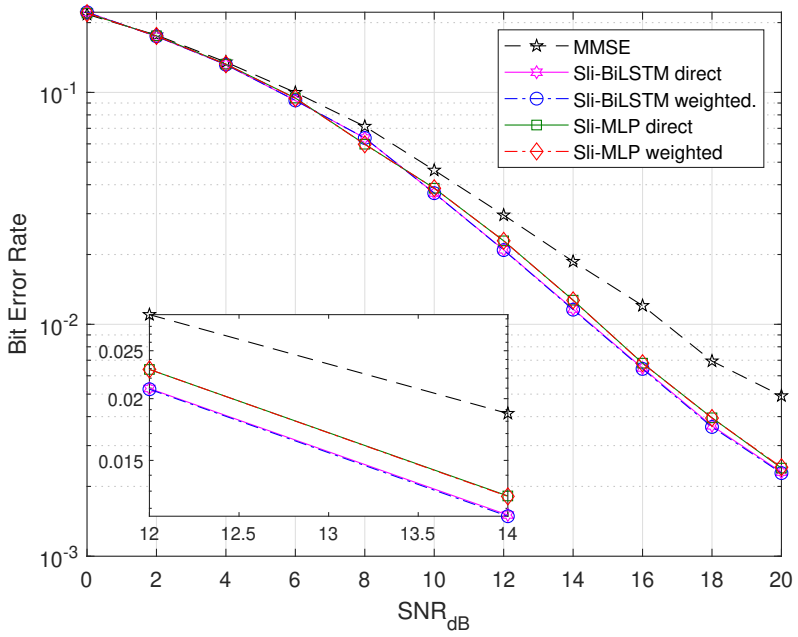

Figure 3.10: BER results of the Sli-BiLSTM and Sli-MLP model for the varying channels, $L = 3$, $N = 10$, and the weighted or direct averages are used.

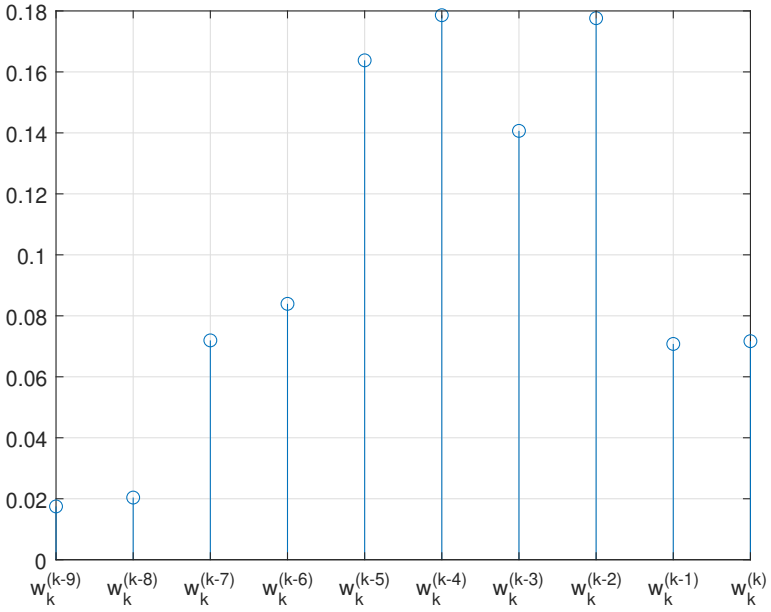

Figure 3.11: Learned window weights $\hat{w}_k^{(j)}$.

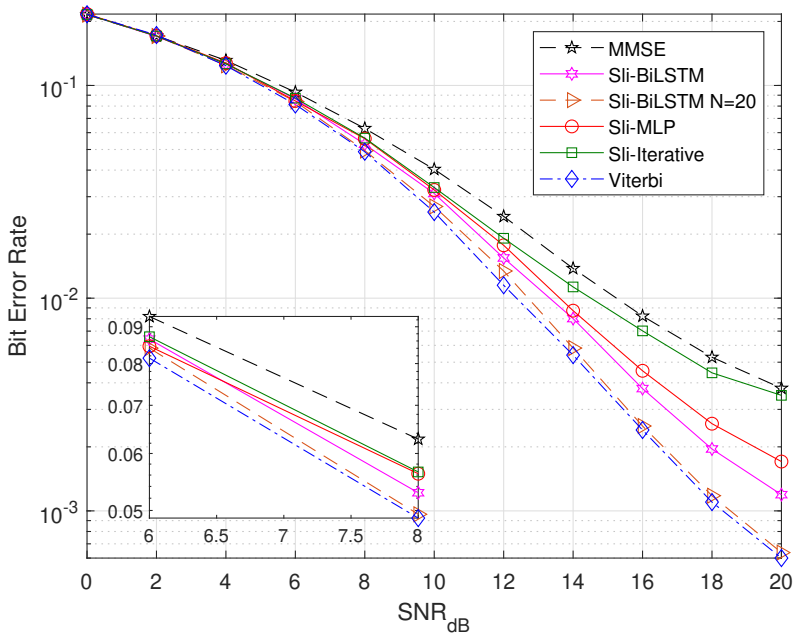

Figure 3.12: BER results of the Sli-BiLSTM, Sli-MLP, and Sli-Iterative model for the varying channels, $L = 5$ and $N = 10$.

we also show the BER performance when $N = 20$ for the Sli-BiLSTM model, and observe the result is the same as the one with the Viterbi Algorithm based equalization. Evaluating the models among themselves, we conclude that the Sli-BiLSTM network gives better results as the number of channel taps increases.

We next consider the effects of CSI error. In Fig. 3.13, we show how the BER changes as the CSI error level increases for the Sli-BiLSTM model. Coefficient λ denotes the CSI error level in (3.9). It is observed that the BER increases as the error in the CSI increases as expected. Fig. 3.14 illustrates the performance of all three proposed models when $\lambda = 0.4$. Since the Sli-Iterative model uses a function of channel taps ($\mathbf{G}^*\mathbf{y}$ and $\mathbf{G}^*\mathbf{G}\hat{\mathbf{s}}_k$) instead of the direct channel taps as input, it shows a smaller performance degradation compared to the other models and offers slightly superior performance compared to other models in the $0-10$ dB SNR range. In the $10-20$ dB SNR range, the Sli-BiLSTM model gives better results than the other proposed models. In the $12-20$ dB SNR range, the Sli-BiLSTM model performs quite close to the Viterbi Algorithm based equalization, while outperforming the linear MMSE equalizers by 2 dB, at a BER level of 10^{-2}.

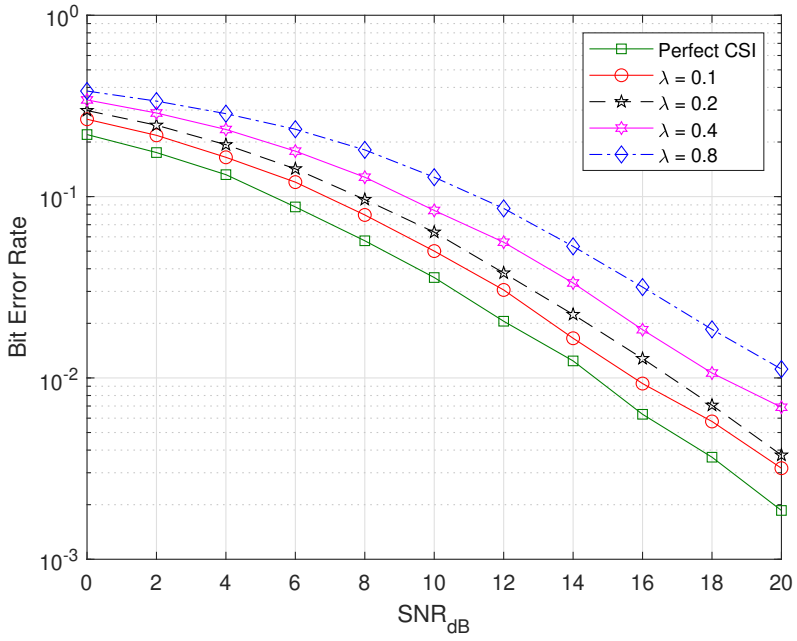

Figure 3.13: BER results of the Sli-BiLSTM model for the varying channels $L = 3$, $N = 10$, and different channel state information error levels.

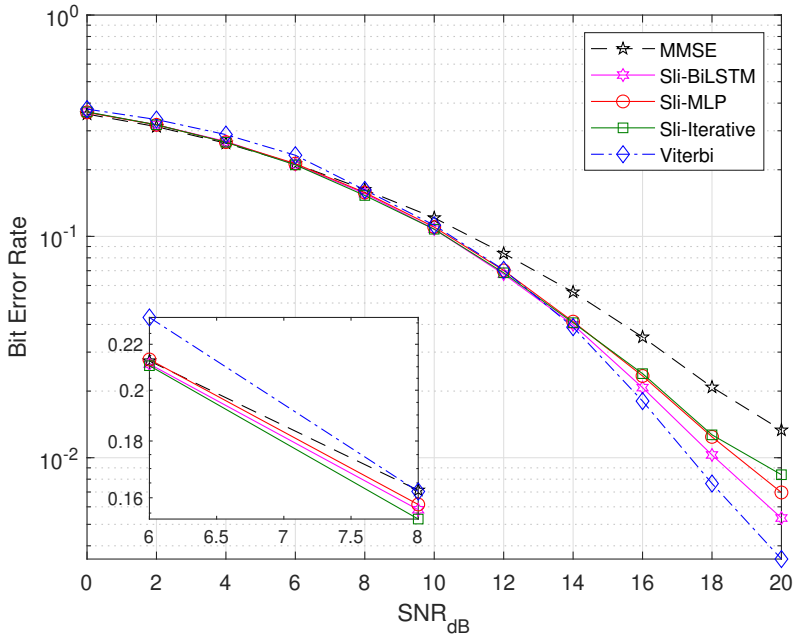

Figure 3.14: BER results of the Sli-BiLSTM, Sli-MLP, and Sli-Iterative model for the varying channels, $L = 5$, $N = 10$, and $\lambda = 0.4$.

In Fig 3.15, we produce results for a fixed channel realization with channel taps: $g_0 = 0.273$, $g_1 = 0.156$, $g_2 = -0.094$, $g_3 = -0.0026$, $g_4 = -0.427$, $g_5 = -0.206$, $g_6 = -0.149$. In this case, we obtain very close results with the Viterbi Algorithm based equalization. We can also produce results for a larger number of channel taps (e.g., $L = 7$) compared to varying ISI channel examples. Since we use a longer ISI length $L = 7$ compared to previous simulations, we produce bit error rates for $N = 20$ as well. The performance gap between the proposed models and the linear MMSE equalizers after 12 dB widens, providing a gain of more than 4 dB at a BER level of 10^{-2}. Moreover, the proposed models produce results as close as 1 dB to the Viterbi Algorithm based equalization for the same SNR range.

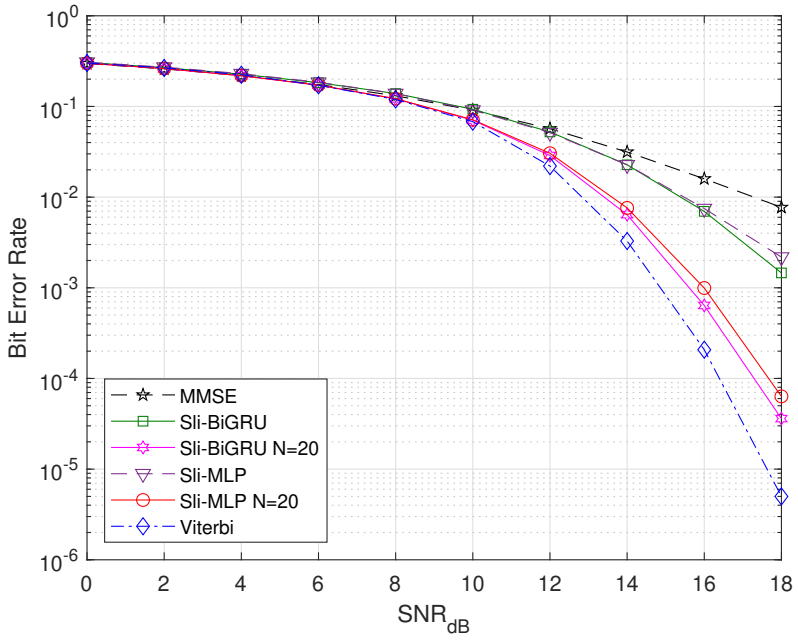

Figure 3.15: BER results of the Sli-BiGRU and Sli-MLP model for a fixed channel with $L = 7$.

3.5 Chapter Summary

In this chapter, we use sliding window based DL models to detect data streams that received through an ISI channel. Specifically, we propose three different DL-based methods, namely Sli-BiLSTM, Sli-MLP, and Sli-Iterative for time-varying ISI channels. Based on the idea that each window used in a symbol's estimation process should have a different weight, we further introduce a single-layer NN-based window weight network to learn the corresponding window weights. Furthermore, we employ the Sli-BiGRU and Sli-MLP networks for fixed ISI channels. We show that the newly designed ISI detectors are computationally efficient, and can perform equalization under perfect or noisy CSI in an effective manner. We evaluate the performance of the proposed models for different settings, and the results show that the proposed models produce close results with the Viterbi Algorithm based equalization while they have a superior performance compared to linear MMSE equalizers.

Chapter 4

Deep Learning Based Channel Equalization for MIMO ISI Channels

In this chapter, we examine DL-based equalization techniques for multiple-input multiple-output (MIMO) ISI channels, by extending the proposed models in the previous chapter to the MIMO ISI case. We also discuss the complexities and limitations of the newly developed models. To assess the performance of the proposed solutions, the bit error rate results of the proposed DL-based models are compared with those of the Viterbi Algorithm and linear MMSE equalizer.

The chapter is organized as follows. Section 4.1 describes the system model. The models presented in Chapter 3 are extended for equalization of MIMO ISI channels in Section 4.2. We present our numerical results in Section 4.3, and conclude the chapter in Section 4.4.

4.1 System Model

We focus on transmission over a MIMO-ISI channel. We consider an $n_T \times n_R$ MIMO system with n_T transmitting and n_R receiving antennas. When the number of transmitting and receiving antennas is one, the model becomes single-input single-output (SISO), which is the setup considered in the previous chapter.

The received vector contains intersymbol interference and AWGN, which is white in both time and space (i.e., independent across antennas). Then the received sequence at the m^{th} receiver and at time $k = 0, \ldots K-1$ becomes

$$y_k^{(m)} = \sum_{n=1}^{n_T} \sum_{l=0}^{L-1} g_l^{(m,n)} s_{k-l}^{(n)} + z_k^{(m)}. \tag{4.1}$$

where $s_k^{(n)}$ is the symbol sequence at the transmitter n, $g_k^{(m,n)}$ is the impulse response from the transmitter n to the receiver m.

The received signal vector can also be written as

$$\begin{bmatrix} y_k^{(1)} \\ \vdots \\ y_k^{(n_R)} \end{bmatrix} = \sum_{l=0}^{L-1} \mathbf{G_l} \begin{bmatrix} s_{k-l}^{(1)} \\ \vdots \\ s_{k-l}^{(n_T)} \end{bmatrix} + \begin{bmatrix} z_k^{(1)} \\ \vdots \\ z_k^{(n_R)} \end{bmatrix}, \tag{4.2}$$

where $z_k^{(1)}, \ldots . z_k^{(n_R)}$ are AWGN samples, and $\mathbf{G_l}$ is the discrete-time equivalent impulse response of the l^{th} tap, where $\mathbf{G_l}$ is defined as

$$\mathbf{G_l} = \begin{pmatrix} g_l^{(1,1)} & \cdots & g_l^{(1,n_T)} \\ \vdots & \ddots & \vdots \\ g_l^{(n_R,1)} & \cdots & g_l^{(n_R,n_T)} \end{pmatrix}_{n_R \times n_T}. \tag{4.3}$$

We assume that the l^{th} channel taps have a zero mean Gaussian distribution whose power follows an exponential delay profile, i.e.,

$$g_l^{(m,n)} \sim \mathcal{N}(0,\, \sigma_l^2). \tag{4.4}$$

with

$$\sigma_l^2 = \frac{e^{-\gamma \times l}}{\sum_{l=0}^{L-1} e^{-\gamma \times l}} \tag{4.5}$$

where γ is a coefficient that depends on the wireless channel environment, i.e., following an exponentially decaying power delay profile, as also adopted in the previous chapter. The sliding-window model for the received sample is as follows:

$$\mathbf{y_n} = \mathbf{G}\mathbf{s_n} + \mathbf{z_n}, \tag{4.6}$$

where, for considering the N_1 previous and N_2 following received symbols, we have:

$$\mathbf{y_n} \triangleq \left[y_{n-N_1}^{(1)}, ..., y_{n-N_1}^{(n_R)}, ..., y_{n+N_2}^{(1)}, ..., y_{n+N_2}^{(n_R)} \right]_{(n_R N)\times 1}$$

$$\mathbf{s_n} \triangleq \left[s_{n-N_1-L+1}^{(1)}, ..., s_{n-N_1-L+2}^{(n_T)}, ..., s_{n+N_2}^{(1)}, ..., s_{n+N_2}^{(n_T)} \right]_{n_T(N+L-1)\times 1} \tag{4.7}$$

$$\mathbf{z_n} \triangleq \left[z_{n-N_1}^{(1)}, ..., z_{n-N_1}^{(n_R)}, ..., z_{n+N_2}^{(1)}, ..., z_{n+N_2}^{(n_R)} \right]_{(n_R N)\times 1}.$$

The channel matrix is then defined as:

$$\mathbf{G} = \begin{pmatrix} \mathbf{G_{L-1}} & \cdots & \mathbf{G_0} & 0 & \cdots & \cdots & 0 \\ 0 & \mathbf{G_{L-1}} & \cdots & \mathbf{G_0} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & \mathbf{G_{L-1}} & \cdots & \mathbf{G_0} \end{pmatrix}_{n_R N \times n_T(N+L-1)} . \tag{4.8}$$

where $N = 1 + N_1 + N_2$.

4.2 Proposed Deep Learning Based Models

We extend the three models proposed in Chapter 3 for the MIMO setup. We first examine the Sliding Bi-LSTM detector, and then Sliding MLP. Finally, we also use the idea of DetNet [37] and Iterative model [38] which are designed for MIMO detection to solve the channel equalization problem in MIMO ISI channels.

4.2.1 Sliding Bi-LSTM Equalizer MIMO ISI Channels

In this section, we introduce the sliding Bi-LSTM equalizer for MIMO ISI channels. The input size of the network is different from the SISO case. Each channel matrix contains $n_R \times n_T$ elements, hence a total of $L \times n_R \times n_T$ elements should be concatenated with part of the received vector of length $N \times n_R$ as the input. Therefore, the input length becomes $(L \times n_R \times n_T) + (N \times n_R)$ and the output length is $N \times n_T$. The resulting network architecture is shown in Fig. 4.1.

We perform the hyperparameter search for $L = 3$, $N = 10$, and n_R=n_T=2, and we use the selected model for other parameters as well. In our specific design example, the inputs are fed into four layers of a Bi-LSTM network, these four layers have sizes of 100, 100, 50, and 100 units for each forward and backward state, which is the same as the SISO model considered in the previous chapter.

By using an $N \times n_R$ symbols long window and performing detection for $N \times n_T$

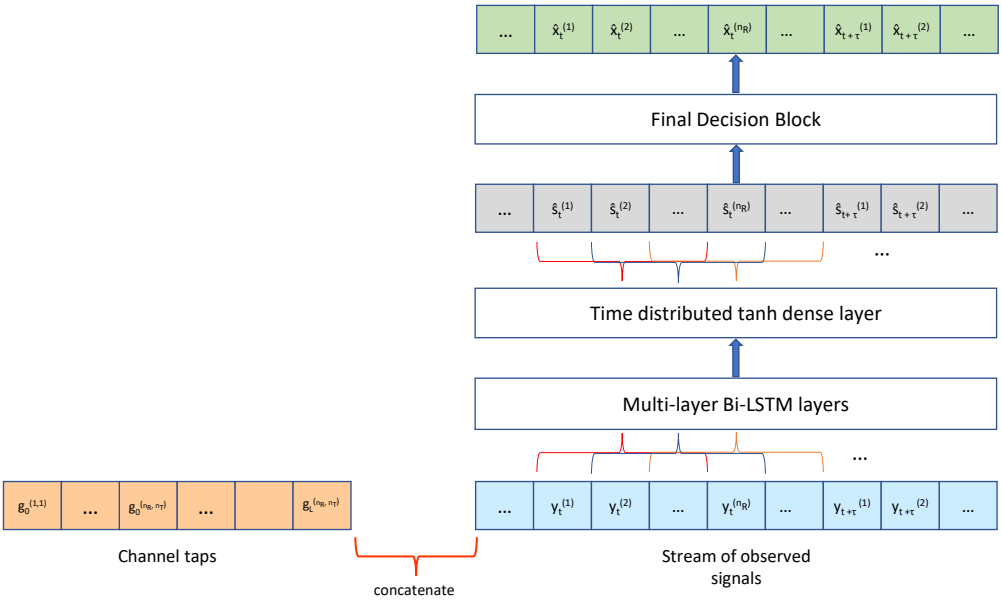

Figure 4.1: The network architecture of Sliding Bi-LSTM MIMO ISI detector.

symbols, the Sli-BiLSTM network slides ahead by n_R symbols, and the same process is repeated. In Fig. 4.2, we show how the detector is shifted for $N = 3$, and n_R=2.

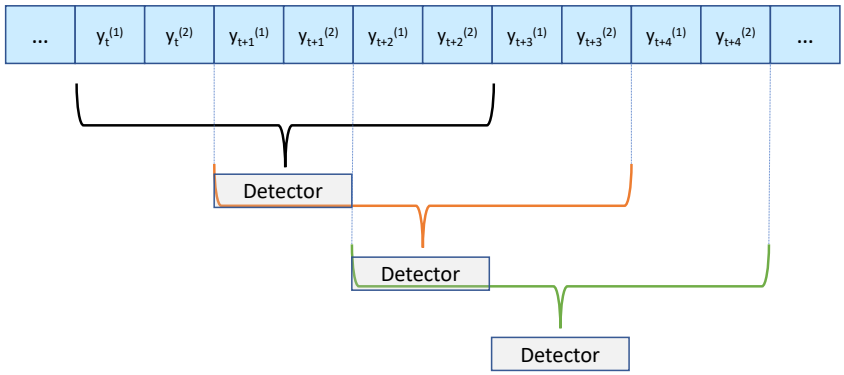

Figure 4.2: The Sliding Bi-LSTM Detector for N=3 and n_R=2.

4.2.2 Sliding MLP Equalizer MIMO ISI Channels

In this model, we use MLP instead of the multi-layer Bi-LSTM. We use the model structure employed in the previous chapter, however, by doubling the number of neurons. The remaining parameters, and training and test methodologies are as described in the previous chapter.

4.2.3 Sliding Iterative Equalizer MIMO ISI Channels

We introduce the Sli-Iterative model for MIMO ISI channels as well. We examined this model in detail in the previous chapter; here we only review the modifications required for the case of MIMO ISI channels. For the MIMO ISI modification, the $\mathbf{y}$ and $\mathbf{s}$ variables in (3.5) need to be replaced with the expressions in (4.7). In addition, the matrix given in (4.8) should be used as the $\mathbf{G}$ matrix. The input size for the MIMO ISI model, in the case of BPSK modulation, is $3N_t$ (where N_t=$n_T(N+L)$), since the real part of three input features, each with a dimension N_t, $\mathbf{G}^*\mathbf{y}$, $\mathbf{G}^*\mathbf{G}\hat{\mathbf{s}}_\mathbf{k}$, $\hat{\mathbf{s}}_\mathbf{k}$ are concatenated. After the mentioned changes are made for the input vector, it is fed to the network in Fig. 3.6.

4.3 Simulation Results

In this section, we evaluate the BER performance of the proposed DL-based models for MIMO ISI channels. We model the multipath channels with exponential–decay power–delay profiles as given in (4.4). The parameter γ in (4.5) is taken as $\gamma = 5$. The parameters used in the test part are shown in Table 4.1.

As the parameters L, n_T, and n_R increase, the proposed models become more complex, hence their training becomes more difficult. For this reason, we limit the number of antennas and delay taps in our simulations.

Fig. 4.3 and 4.4 illustrate the bit error rates of all three proposed models for

Table 4.1: The test set parameters for MIMO setup

Parameter	**Selected Value**
K	$n_T(200 + 2(L-1))$
γ	$1/5$
λ	0.4
N	10
test set size	5000
test SNR in dB	$\{0,1,2,\ldots,20\}$

the cases with $L = \{2, 3\}$ and $n_R = n_T = 2$. We observe that all proposed models outperform the linear MMSE equalizer for the case with $L = 2$ and $n_R = n_T = 2$. The Sli-Iterative and Sli-MLP models show similar results at low SNRs ($0 - 6$ dB). We also observe that the Sli-BiLSTM detector performs better than other models. We produced results for different numbers of Bi-LSTM layers and the number of states in these layers as well. However, we did not observe significantly better results, hence they are not reported here.

One of the problems with the Sli-Iterative detector is that as the input size increases, the model has difficulty in learning and cannot produce good results for intermediate and high SNRs (between $10 - 20$ dB). The Sli-Iterative model performs worse than Sli-MLP for the $10 - 20$ dB SNR range, even though its structure is more complex. In cases where the number of channel taps is not excessive, we obtain results close to the Viterbi algorithm based equalization. As the number of channel taps increases, the BER results of the proposed models move away from the Viterbi Algorithm based equalization.

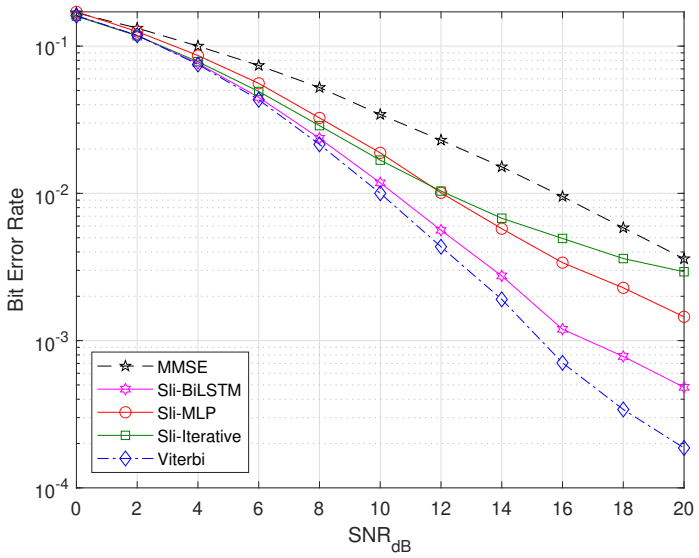

Figure 4.3: BER results of the Sli-BiLSTM, Sli-MLP, and Sli-Iterative detectors for MIMO ISI, varying channels, $L = 2$ and n_R=n_T=2.

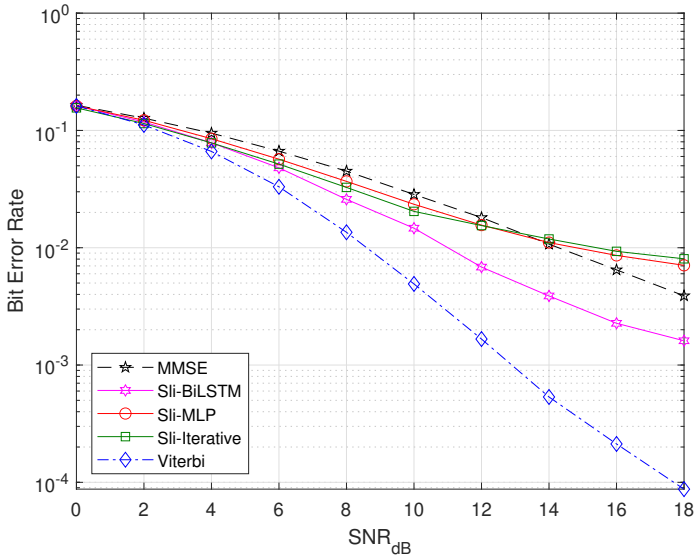

Figure 4.4: BER results of the Sli-BiLSTM, Sli-MLP, and Sli-Iterative detectors for MIMO ISI, varying channels, $L = 3$ and n_R=n_T=2.

In Fig. 4.5, we show the performance of all three proposed models for the same MIMO ISI configuration with CSI error. We take $\lambda = 0.4$. The Sli-Iterative

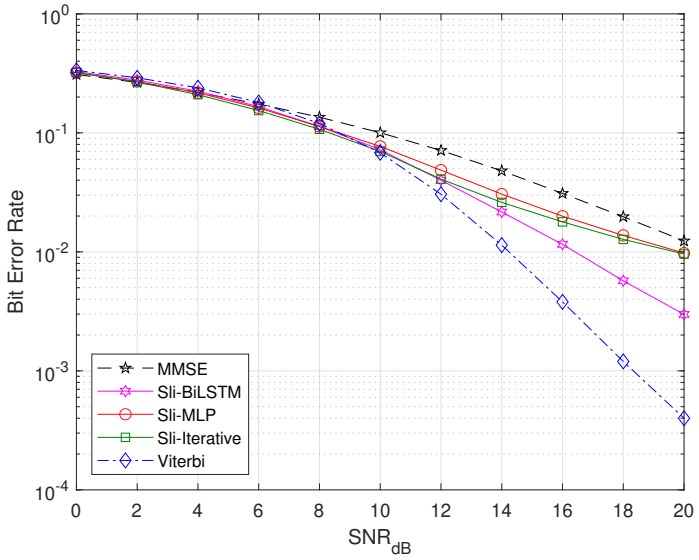

Figure 4.5: BER results of the Sli-BiLSTM,Sli-MLP and Sli-Iterative detectors for MIMO ISI, varying channels, $L = 3$, n_R=n_T=2 and $\lambda = 0.4$.

model shows close results with the Viterbi Algorithm based equalization and the Sli-BiLSTM model in the $0 - 12$ dB SNR range. Since the Sli-Iterative uses a function of channel taps ($\mathbf{G}^*\mathbf{y}$ and $\mathbf{G}^*\mathbf{G}\hat{\mathbf{s}}_k$) instead of the direct channel taps, it shows less performance degradation compared to the other models. In the $12-20$ dB SNR range, the Sli-BiLSTM gives more successful bit error rates compared to the other models. In the $12-20$ dB SNR range, the Sli-BiLSTM model performs better than the linear MMSE equalizer by more than 4 dB at a BER of 10^{-2}. We observe that the results of the proposed models are closer to the Viterbi Algorithm based equalization than those in Fig. 4.4, which are obtained with perfect CSI.

In Fig. 4.6, we produce results for a fixed channel realization with channel tap matrices

$$\mathbf{G_0} = \begin{pmatrix} -0.273 & -0.496 \\ -0.690 & 1.018 \end{pmatrix}, \mathbf{G_1} = \begin{pmatrix} 0.618 & -0.291 \\ -0.187 & 0.778 \end{pmatrix}, \mathbf{G_2} = \begin{pmatrix} -0.558 & 0.604 \\ -0.116 & 0.700 \end{pmatrix}.$$

In this case, we use Bi-GRU or MLP instead of Bi-LSTM as the complexity of the

model is smaller, so it would be more accurate to call the models Sli-BiGRU and Sli-MLP. These proposed models produce close results to the Viterbi Algorithm based equalization. We produce results for longer ISI channels using the proposed models in this case compared to examples given for time-varying ISI channels. Since computational complexity of the Viterbi Algorithm grows exponentially with L and n_T, it is not preferable to use it for such cases. We note that in cases where L and n_T values are selected larger, Sli-BiGRU or Sli-MLP can be used which outperform the linear MMSE equalizer by a large margin.

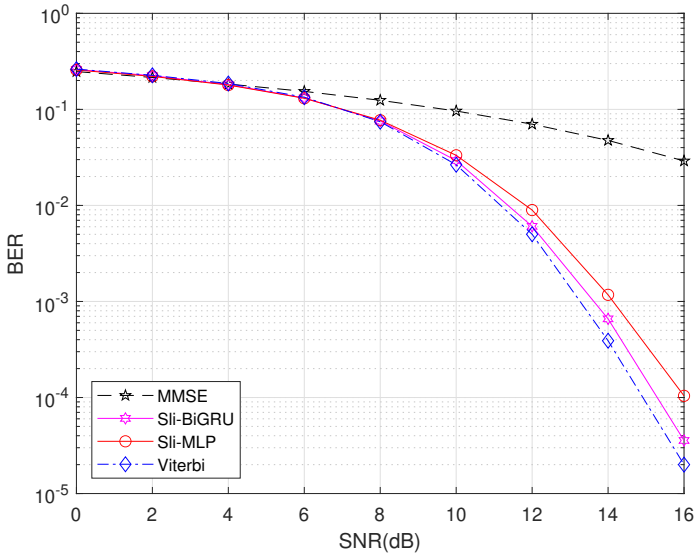

Figure 4.6: BER results of the Sli-BiGRU and Sli-MLP detectors for a fixed channel MIMO ISI channel, $L = 3$, and n_R=n_T=2.

4.4 Chapter Summary

In this chapter, we focus on transmission over a MIMO-ISI channel. We propose three different DL-based channel equalization methods, namely Sli-BiLSTM, Sli-MLP, and Sli-Iterative for the use of time-varying channels, obtained as extensions of the solutions developed in the previous chapter. We perform simulations with perfect and imperfect CSI, and compare results with those of the linear MMSE

equalizer and the Viterbi Algorithm based equalization. The results show that the proposed models produce close results with the Viterbi Algorithm while they have a superior performance compared to linear MMSE equalizers as long as the ISI length and the number of transmitting antennas are not excessive. Some modifications and new ideas are needed to get closer results with those of the Viterbi Algorithm based equalization for larger numbers of antennas or longer ISI lengths.

Chapter 5

Conclusions and Future Work

In this thesis, we propose DL-based equalization techniques for channels with ISI. Three different DL-based methods, namely Sli-BiLSTM, Sli-MLP, and Sli-Iterative are proposed for time-varying ISI channels, and it is demonstrated that they are computationally efficient and capable of performing equalization under a variety of channel conditions with the knowledge of the channel state information. Based on the idea that each window used in a symbol's estimation process should have a different weight, we further introduce a single-layer NN-based window weight network to learn corresponding window weights. We make the final estimation of a symbol with the weighted or direct average of these different predictions. Furthermore, we employ the Sli-GRU and Sli-MLP networks for fixed ISI channels. Through numerical simulations with perfect and noisy CSI, we investigate the effects of the parameters (ISI length, processing window lenght, and CSI error level) on BER and we observe that the proposed models produce very close results with the Viterbi Algorithm based channel equalization with much less computational complexity, while they offer superior results compared to the linear MMSE equalizer.

We also extend our study to transmission over MIMO ISI channels by extending the proposed SISO equalizer networks. Numerical results demonstrate that the proposed models produce close results with the Viterbi Algorithm based

equalization while they have a superior performance compared to linear MMSE equalizers as long as the ISI length and the number of transmitting antennas are not excessive.

One possible future research direction is to modify the networks and evaluate the bit error rates by using different settings, e.g., different modulation techniques (such as QAM and QPSK) instead of BPSK. For example, if the modulation technique is changed to QAM, the imaginary part of the received signal can be given as a second feature in the same time instance for the proposed Sli-BiLSTM model but this is not the case for the Sli-MLP model. The reason is that time-step and feature are two separate dimensions in the Bi-LSTM networks but not in the MLP networks. In the Sli-MLP model, the imaginary part should be concatenated with the real part as in the Sli-Iterative model.

Another possible future research direction is to evaluate the performance of the newly proposed models under additional impairments. In this thesis, we produced results for the noisy channel estimates as the only impairment other than ISI and noise. In addition to this, the effects of impairments such as carrier frequency offset, power amplifier distortion, and IQ imbalance on DL based equalization can also be investigated.

Finally, as we mentioned before, the results for larger numbers of antennas and longer ISI lengths are open for further research. Different neural network architectures can be designed or the proposed models can be potentially modified to obtain DL-based MIMO ISI detectors with better detection capabilities or computational complexities for this purpose.

Bibliography

[1] A. Zappone, M. D. Renzo, and M. Debbah, "Wireless networks design in the era of deep learning: Model-based, AI-based, or both?" *IEEE Transactions on Communications*, vol. 67, pp. 7331–7376, June 2019.

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, p. 359–366, July 1989.

[3] N. Farsad and A. Goldsmith, "Neural network detection of data sequences in communication systems," *IEEE Transactions on Signal Processing*, vol. 66, no. 21, pp. 5663–5678, Aug. 2018.

[4] F.-L. Luo, *Machine Learning for Future Wireless Communications*. Wiley-IEEE Press, 2020, p. 213–241.

[5] J. Proakis, "Adaptive equalization for TDMA digital mobile radio," *IEEE Transactions on Vehicular Technology*, vol. 40, no. 2, pp. 333–341, May 1991.

[6] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, July 2017.

[7] I. Ahmed, W. Xu, R. Annavajjala, and W.-S. Yoo, "Joint demodulation and decoding with multi-label classification using deep neural networks," in *2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2021, pp. 365–370.

[8] B. He, Z. Wu, and F. Wang, "Rethinking: Deep-learning-based demodulation and decoding," June 2022.

[9] T. Wu, "CNN and RNN-based deep learning methods for digital signal demodulation," Feb. 2019, pp. 122–127.

[10] F. Lemic, V. Handziski, and J. Famaey, "Toward regression-based estimation of localization errors in fingerprinting-based localization," Feb. 2019.

[11] T. Omomule, O. D.O., C. Ugwu, and F. T.A., "QoS performance metrics for analyzing wireless network usability," Dec. 2019.

[12] F. Shaheen, B. Verma, and M. Asafuddoula, "Impact of automatic feature extraction in deep learning architecture," in *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Dec. 2016, pp. 1–8.

[13] A. Ali and F. Yangyu, "Unsupervised feature learning and automatic modulation classification using deep learning model," *Physical Communication*, vol. 25, Sep. 2017.

[14] S. Ramjee, S. Ju, D. Yang, X. Liu, A. E. Gamal, and Y. C. Eldar, "Fast deep learning for automatic modulation classification," *ArXiv*, vol. abs/1901.05850, Jan. 2019.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," Jan. 1986.

[16] S. Ravula and S. Jain, "Deep autoencoder-based massive MIMO CSI feedback with quantization and entropy coding," in *2021 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2021, pp. 1–6.

[17] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, Mar. 2016, pp. 223–228.

[18] S. Cammerer, F. A. Aoudia, S. Dörner, M. Stark, J. Hoydis, and S. ten Brink, "Trainable communication systems: Concepts and prototype," *IEEE Transactions on Communications*, vol. 68, no. 9, pp. 5489–5503, Sep. 2020.

[19] F. A. Aoudia and J. Hoydis, "End-to-end learning of communications systems without a channel model," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, Dec. 2018, pp. 298–303.

[20] F. Alberge, "Deep learning constellation design for the AWGN channel with additive radar interference," *IEEE Transactions on Communications*, vol. 67, no. 2, pp. 1413–1423, Feb. 2019.

[21] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, Feb. 2011, pp. 1143–1146.

[22] A. Kwasinski, W. Wang, and F. Shah-Mohammadi, *Reinforcement Learning for Resource Allocation in Cognitive Radio Networks*, Dec. 2019, pp. 27–44.

[23] J. Li, R. Zhao, H. Hu, and Y. Gong, "Improving RNN transducer modeling for end-to-end speech recognition," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Sep. 2019, pp. 114–121.

[24] A. F. Ganai and F. Khursheed, "Predicting next word using RNN and LSTM cells: Stastical language modeling," in *2019 Fifth International Conference on Image Information Processing (ICIIP)*, Nov. 2019, pp. 469–474.

[25] M. Wang, L. Song, X. Yang, and C. Luo, "A parallel-fusion RNN-LSTM architecture for image caption generation," in *2016 IEEE International Conference on Image Processing (ICIP)*, Sep. 2016, pp. 4448–4452.

[26] C. Su, H. Huang, S. Shi, P. Jian, and X. Shi, "Neural machine translation with gumbel tree-LSTM based encoder," *J. Vis. Commun. Image Represent.*, vol. 71, p. 102811, Aug. 2020.

[27] Y. Chen and K. Wang, "Prediction of satellite time series data based on long short term memory-autoregressive integrated moving average model (LSTM-ARIMA)," in *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*, July 2019, pp. 308–312.

[28] S. Zheng, Y. Liu, and P. H. Siegel, "PR-NN: RNN-based detection for coded partial-response channels," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 1967–1982, July 2021.

[29] H.-M. Ou, C.-F. Teng, W.-C. Tsai, and A.-Y. A. Wu, "A neural network-aided viterbi receiver for joint equalization and decoding," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, Sep. 2020, pp. 1–6.

[30] S. Deligiannidis, C. Mesaritakis, and A. Bogris, "Performance and complexity analysis of bi-directional recurrent neural network models versus volterra nonlinear equalizers in digital coherent systems," *Journal of Lightwave Technology*, vol. 39, no. 18, pp. 5791–5798, June 2021.

[31] S. Deligiannidis, A. Bogris, C. Mesaritakis, and Y. Kopsinis, "Compensation of fiber nonlinearities in digital coherent systems leveraging long short-term memory neural networks," *Journal of Lightwave Technology*, vol. 38, no. 21, pp. 5991–5999, Sep. 2020.

[32] S. Lai and M. Li, "Recurrent neural network assisted equalization for FTN signaling," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, June 2020, pp. 1–6.

[33] W. Xu, Z. Zhong, Y. Be'ery, X. You, and C. Zhang, "Joint neural network equalizer and decoder," in *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, Aug. 2018, pp. 1–5.

[34] H. Wu, Y. Zhang, X. Zhao, N. Zhu, and M. Coates, "End-to-end physical layer communication using bi-directional GRUs for ISI channels," in *2020 IEEE Globecom Workshops (GC Wkshps)*, Dec. 2020, pp. 1–6.

[35] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[36] N. Samuel, T. Diskin, and A. Wiesel, "Deep MIMO detection," in *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, June 2017, pp. 1–5.

[37] ——, "Learning to detect," *IEEE Transactions on Signal Processing*, vol. 67, no. 10, pp. 2554–2564, Feb. 2019.

[38] O. Sholev, H. H. Permuter, E. Ben-Dror, and W. Liang, "Neural network MIMO detection for coded wireless communication with impairments," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, June 2020, pp. 1–8.

[39] H. He, C.-K. Wen, S. Jin, and G. Y. Li, "Model-driven deep learning for MIMO detection," *IEEE Transactions on Signal Processing*, vol. 68, pp. 1702–1715, Feb. 2020.

[40] X. Yan, F. Long, J. Wang, N. Fu, W. Ou, and B. Liu, "Signal detection of MIMO-OFDM system based on auto encoder and extreme learning machine," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 1602–1606.

[41] T. J. O'Shea, T. Erpek, and T. C. Clancy, "Deep learning based MIMO communications," *CoRR*, vol. abs/1707.07980, July 2017. [Online]. Available: http://arxiv.org/abs/1707.07980

[42] T. Erpek, T. J. O'Shea, and T. C. Clancy, "Learning a physical layer scheme for the MIMO interference channel," in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–5.

[43] M. A. Wijaya, K. Fukawa, and H. Suzuki, "Intercell-interference cancellation and neural network transmit power optimization for MIMO channels," in *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*, Jan. 2015, pp. 1–5.

[44] T. J. O'Shea, T. Roy, and N. West, "Approximating the void: Learning stochastic channel models from observation with variational generative adversarial networks," in *2019 International Conference on Computing, Networking and Communications (ICNC)*, Aug. 2019, pp. 681–686.

[45] T. Gruber, S. Cammerer, J. Hoydis, and S. t. Brink, "On deep learning-based channel decoding," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, Jan. 2017, pp. 1–6.

[46] V. Ninkovic, D. Vukobratovic, C. Häger, H. Wymeersch, and A. Graell i Amat, "Autoencoder-based unequal error protection codes," *IEEE Communications Letters*, vol. 25, no. 11, pp. 3575–3579, Aug. 2021.

[47] W. Lyu, Z. Zhang, C. Jiao, K. Qin, and H. Zhang, "Performance evaluation of channel decoding with deep neural networks," in *2018 IEEE International Conference on Communications (ICC)*, Jan. 2018, pp. 1–6.

[48] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, Jan. 2018.

[49] T. J. O'Shea, L. Pemula, D. Batra, and T. C. Clancy, "Radio transformer networks: Attention models for learning to synchronize in wireless systems," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, May 2016, pp. 662–666.

[50] M. J. Park, J. Ok, Y.-S. Jeon, and D. Kim, "MetaSSD: Meta-learned self-supervised detection," in *2022 IEEE International Symposium on Information Theory (ISIT)*, May 2022, pp. 480–485.

[51] N. Shlezinger, N. Farsad, Y. C. Eldar, and A. J. Goldsmith, "Data-driven factor graphs for deep symbol detection," in *2020 IEEE International Symposium on Information Theory (ISIT)*, Jan. 2020, pp. 2682–2687.

[52] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. null, p. 281–305, Feb. 2012.

[53] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *CoRR*, vol. abs/1505.00387, Nov. 2015. [Online]. Available: http://arxiv.org/abs/1505.00387