Machine Learning for Reliable Communication and Improved Tracking of Dynamical Systems

by

Kirty Prabhakar Vedula

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Electrical and Computer Engineering

by

October 2020

APPROVED:

Professor D. Richard Brown III, Major Advisor, Department of ECE, WPI

Professor Randy Paffenroth, Department of Mathematics, Department of Computer Science, and Data Science Program, WPI

Professor Ziming Zhang, Department of ECE, WPI

Abstract

Reliable communication systems and optimal tracking of dynamic systems are subjects that have been studied for several decades. In recent years, however, there is a renewed interest in these subjects from the perspective of machine learning. This dissertation applies machine learning techniques to develop new insights and specific algorithms for two important problems: (i) achieving reliable communications in noisy channels using autoencoders and (ii) improving tracking of dynamical systems using machine learning.

The first problem considers a joint coding and modulation scheme for an end to-end communication system design using an autoencoder architecture in short blocklength regime. Unlike the classical approach of separately designing error correction codes and modulation schemes for a given channel, the approach here is to learn an optimal mapping directly from messages to channel inputs while simultaneously learning an optimal mapping directly from channel outputs to estimated messages. Additive white Gaussian noise (AWGN) channels are considered first and the performance of the autoencoder is compared against various coding and modulation schemes for linear block codes. An analysis on conditional block error rate is presented with interesting insights on the geometry of the codewords generated by the autoencoder. For AWGN channels, numerical results show that the autoencoder can achieve better block error rate (BLER) performance than BPSK modulated Hamming codes with maximum likelihood decoding.

We extend it to other non-canonical channels which have no-known good codes such as Bernoulli-Gaussian Impulsive Noise (BGIN) channel. A family of autoencoders is developed for different probabilistic parameters in BGIN channels. Numerical results show the autoencoder achieves uniformly better BLER performance than conventional block codes. The proposed architecture is general and can be modified for comparison against other block coding schemes and higher-order modulations.

The second problem considers tracking dynamical systems under parametric mismatch and model switching. The Kalman filter (KF) is the optimal state estimator for linear and Gaussian dynamic systems. Optimality of the KF, however, requires exact knowledge of covariances and the system dynamics. Otherwise, the KF will be "mismatched" and, consequently, the state estimates and predictions will not be optimal. We develop a machine learning approach to accurately predict the dynamic states without knowledge of the system dynamics or noise statistics. Considering an application of oscillator phase predictions, we demonstrate that the machine learning approach achieves performance close to optimal irrespective of the amount of parametric mismatch.

Then, model switching is considered in the context of (i) maneuvering target-tracking and (ii) tracking dynamical systems over Gilbert-Elliott Channels. An Interacting Multiple Model (IMM) is a commonly used state estimator that utilizes hypotheses from a filter bank of KFs instead of a single KF to handle changeable and uncertain maneuvering movements. We develop an alternate machine learning method using a Temporal Convolutional Network (TCN) to demonstrate better stability robustness to model switching.

We also develop a hybrid model, Autoencoder Interacting Multiple Model (AEIMM) filter, as an extension to Autoencoder Kalman Filter (AEKF). AEIMM embeds an IMM within an autoencoder framework to learn measurements and their associated measurement covariances for multiple dynamic models. We demonstrate that AEIMM outperforms state-of-the-art maneuvering target-tracking algorithms such as IMM and machine learning models like Long-Short Term Memory Networks.

Acknowledgements

First, I would like to express my sincere appreciation and gratitude to my Ph.D. advisor, Prof. Rick Brown for his support, guidance, enthusiasm and encouragement throughout my graduate studies and research progress. His attention to detail, clarity of thought and way of elucidating abstract concepts is beyond comparison.

I would like to thank Prof. Randy Paffenroth for teaching an amazing course on unsupervised learning and for sparking a life-long fascination for the subject. I would like to thank you for promptly agreeing to be my collaborator and for sharing your vision of research, your continuous encouragement and invaluable feedback throughout my Ph.D. at WPI.

I would also like to thank my collaborators Matthew Weiss, Arick Grootveld, Prof. Andrew Klein. I would also like to thank Prof. Paffenroth and Prof. Zhang for serving as members of my Ph.D. committee and providing valuable feedback on my thesis and defense.

I would like to thank my wife, best friend (and my *illustrator*) Anoosha Papireddy for her infinite love, support and encouragement. Thank you for pushing me to be my best and for being my everyday inspiration! I could not have done this without you.

I would also like to thank my younger brother Sanketh Vedula, a Ph.D. candidate at Technion - Israel Institute of Technology for introducing me to deep learning, medical imaging, finance and for all the valuable and insightful discussions over all these years. I wish him the very best for his Ph.D.

I would like to express my deep gratitude to my parents. Their constant belief in me throughout my life paved my path and helped me strive and work hard to reach greater heights.

I thank my in-laws for providing their moral support. I would also like to thank my cousins and friends for making this journey a smooth sailing. I am grateful for every little help that I was provided on my way up.

I deeply appreciate opportunities that I received through my internships with Philips Research and Witricity. I give my sincere thanks to Francois Vignon and Jun-Seob Shin from Philips and Karl Twelker from Witricity for their valuable discussions, for challenging my growth as a team player and an independent researcher.

Biography

Kirty Vedula was born on June 26, 1990 in Hyderabad, India. He grew up in Vijayawada, India finishing his schooling leading to a Bachelors of Technology in Electronics and Communication Engineering at Amrita University in Coimbatore. In 2012, he received an admission for graduate studies from Rutgers University and completed his Masters of Science in Electrical and Computer Engineering in 2014.

After an internship at Mathworks and working at a Silicon Valley startup for a few months, he joined the doctoral program at Worcester Polytechnic Institute in Fall 2015. As a Ph.D. candidate in the Electrical and Computer Engineering, Kirty worked on several projects funded by DARPA and NSF, all under the supervision of his advisor and mentor Prof. Donald Richard Brown III. In addition to teaching assistantships, Kirty also worked as an intern in R&D teams at Philips Research and Witricity in the summer of 2017 and 2019 respectively.

His current work aims at bridging traditional techniques and machine learning algorithms with a focus on communication and tracking systems. His research interests are in wireless communication, machine learning, reinforcement learning and dynamical systems.

Kirty joins the Qualcomm Wireless R&D team in November 2020 and hopes to continue to make significant contributions to wireless communication and machine learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we briefly introduce the problems discussed in this dissertation and provide motivation to study them. Section 1.2 discusses the relevant prior art for this dissertation. In Section 1.3, we list the chapter-wise contributions and overview of this dissertation and list the publications in Section 1.4.

1.1 Problem Statement

Reliable communication systems and optimal tracking of dynamical systems have been studied for several decades. However, in recent years, there is a renewed interest in these subjects from the perspective of Machine Learning (ML).

The unifying theme in this dissertation is ML as shown in Figure 1.1. ML plays a useful role in settings where there is either a *model deficit* or *algorithmic deficit* [1]. We consider *algorithmic deficit* in this dissertation and use ML to

1. improve end-to-end communication systems with Additive White Gaussian Noise (AWGN) channel and a few non-AWGN channels with no known good codes,
2. improve tracking in situations where the Kalman Filter (KF) is not optimal. We consider cases with parametric and model mismatch and discuss three applications - oscillator phase predictions, tracking in Gilbert-Elliott channels and maneuvering

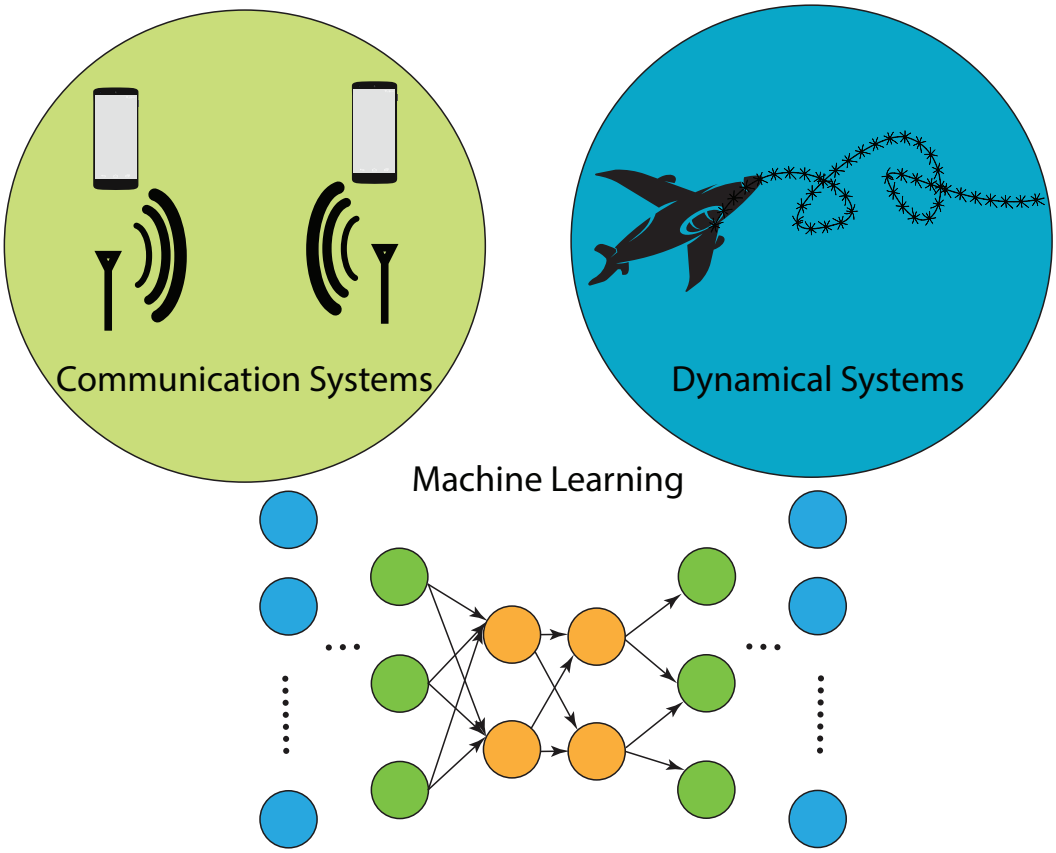

Figure 1.1: A fresh perspective on communication and tracking systems using machine learning.

target tracking - to address different issues and demonstrate the advantages of ML over traditional methods using numerical examples.

1.2 Relevant Prior Work

In this section, we discuss details on specific problems considered in this dissertation and review the most relevant prior work.

1.2.1 Machine Learning

Traditional algorithmic signal processing models involve solving numerical optimization which typically entails longer iterations to converge. Incorporating domain knowledge into the ML architectures could potentially expedite the training process and convergence

and mainly improves the performance of the model. Techniques such as deep unfolding algorithms [2] and other hybrid models [3], [4] are gaining traction. Future communication and tracking systems will involve these deployments and it is slowly leading into the zone of federated learning [5], where devices make autonomous decisions while also interacting with other devices [6]. Furthermore, this improves energy-efficiency and longevity of the devices.

Deep Learning methods had a breakthrough in the ImageNet challenge in the early 2010s after a period of AI Winter. *AlexNet* architecture was the seed for the resurgence in deep learning [7]. With an improved hardware technology using graphics processing unit (GPU) architectures, the training speed increased and also, the amount of training data. In 2017, deep learning methods were first introduced in the context of communication systems [8] and dynamical systems [9]. More details on these are provided in Chapters 2 and 4.

Neural networks are generally used to approximate other functions by selecting the parameters to minimize the approximation error. In particular, fully-connected feed-forward networks are capable of approximating any continuous function arbitrarily well by utilizing a large but finite number of parameters. This can describe many real-world tasks, e.g. classification of objects on a image [10], transcription of a spoken sentence [11] or translation of a written sequence [12]. The approximation power of deep neural networks is the reason for their current success in a variety of different applications.

ML based solutions are increasingly being applied to solve complex signal processing tasks such as massive Multiple-Input Multiple-Output (MIMO) channel estimation and detection [13], beamforming, Forward Error Correction (FEC) decoding [1], solving partial differential equations [14]. We envision such model-driven approaches might have significant influence in future 6G networks with immense performance gains and ease of implementation.

We can use ML approaches as an alternative to the standard engineering design when there is justification for its suitability, scalability and other advantages [15]. This is

usually determined case-by-case whether we have a model or an algorithmic deficit [1].

- **Model Deficit**: This happens when there are no existing mathematical models due to insufficient domain knowledge, making a conventional model-based design inapplicable.
- **Algorithmic Deficit**: This happens when a well-defined mathematical model is available, but the existing algorithms optimized on the basis of such model are (i) either too complex to be implemented for the given application or (ii) the search space to find an optimal algorithm is too large.

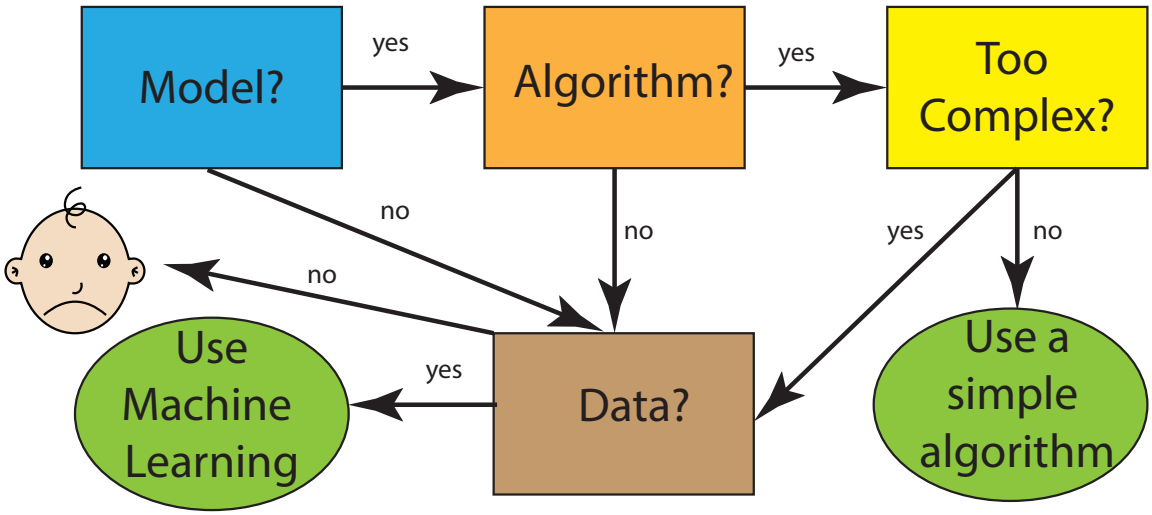

Figure 1.2: Model and algorithmic deficit.

We have three possible scenarios in this context as shown in the flowchart in Figure 1.2

1. where we do not have a good model.
2. where the model exists, but the algorithm is not easily computable.
3. where we know the model and algorithm, but the algorithm is too complex.

In addition to model and algorithmic deficit, we have few more motivational factors to use ML to solve communication system and tracking problems such as:

1. **Limiting functional block-structure**: A traditional system operates as a chain of blocks in which each block is optimized for its performance. However, in an ML-based approach, we have an end-to-end optimized system that considers all blocks

implicitly and designs a model to minimize the overall loss. We see this in action in Chapter 2.

2. **Potential for Online Learning**: ML methods can provide flexibility and reconfigurability since the training and testing phases are separate. It also provides scope for online learning methods such as meta learning and active learning which
3. **Hardware-friendly Computation**: Matrix inversion is computationally expensive, and also causes numerical instabilities. With GPUs, it is possible to implement matrix multiplication with similar complexity as multiplication.

1.2.2 Communication Systems

Communication systems have traditionally been designed by considering each block, i.e., channel encoding, modulation, demodulation, and channel decoding, separately [16]. For example, coding theorists typically design error correction codes by abstracting and lumping the effects of modulation, physical channel impairments, and demodulation into an "effective channel" with certain statistical properties. Similarly, communication theorists typically ignore error correction coding and develop efficient and robust modulation schemes to overcome physical channel impairments. This approach has been successful, especially for longer blocklength codes in AWGN channels, but is also known to not necessarily be optimal in other settings.

More recently, researchers have considered the use of an unsupervised learning framework called *autoencoders* for jointly designing coding and modulation schemes to overcome channel impairments [8], [17]. Generally speaking, autoencoders can be used to find a low-dimensional representation of the input while facilitating reconstruction at the output with minimal error [18].

Autoencoders have been successfully applied for end-to-end communication system design in the binary-input additive white Gaussian noise (bi-AWGN) channel [19]–[22]. More recently, autoencoders have been used to automate the discovery of decoding algo-

rithms for channels that do not have known good codes, e.g., the feedback channel [1]. This work demonstrates the strong generalization capability of classical algorithms like Viterbi and Bahl, Cocke, Jelinek and Raviv (BCJR) on convolutional and turbo codes, with near-optimal performance on AWGN channels. The adaptability and flexibility of neural networks allow them to operate in situations where some simplifying assumptions of standard coding, modulation, demodulation and decoding techniques are not fulfilled [12]. We focus on *algorithmic deficit* - the situation when the model is understood, but solutions are difficult to find in a large search space in this chapter.

Joint channel coding and modulation was first proposed in the context of trellis coded modulation and constellation shaping [23] and more recently in probabilistic amplitude shaping [24] for low-density parity-check codes. We tackle this from an ML standpoint in Chapter 3.

Impulsive noise is considered the main cause for burst errors in data transmission which causes temporary loss in signal. It is prevalent in interference from machines or any other kind of electronic devices, which are sources of random and high power noise [25]. It is non-stationary and comprises of irregular pulses of short duration and energy spikes with random amplitude and spectral content [26].

This can be characterized by the occurrence of large noise samples, which results in a heavy-tailed distribution. Due to their non-stationary nature and high peak power, they can significantly degrade the performance and reliability of communication systems [27], [28]. In this case, a single Gaussian noise model is not correct and many distributions such as Bernoulli-Gaussian model, Middleton Class A, B and Symmetric Alpha Stable (SαS) distributions are more suitable to model these impulsive channels. The Bernoulli-Gaussian model has been widely applied on impulsive noises.

Currently, little is known about channel coding in this setting [29]. In fact, soft decision decoding, while optimal in the AWGN channel, can perform worse than hard decision decoding in impulsive noise channels. This has led to the development of impulsive noise mitigation techniques such as blanking and clipping [30]–[32]. This is handled

in Chapter 4.

1.2.3 Tracking Dynamical Systems

Modeling dynamic random processes is essential in applications like oscillator phase synchronization, channel state prediction, flight tracking, autonomous driving and robotics. State-space models are commonly used for tracking targets over the state observation sequences at each step as originating from internal states of the system [33].

Alternative approaches have been proposed for dealing with these scenarios like conditional random fields, conditional state space models [34]. However, they are model-driven and heavy-handed in their implementation.

Deep learning has been successfully applied in numerous tasks in signal processing such as automatic speech recognition, audio denoising, and audio source separation. Convolutional Neural Networks (CNNs) have shown success in tasks ranging from detection, segmentation and object recognition [12]. More recently, deep learning approaches are being used in conjunction with KFs, specifically related to Kalman smoother [35], [36], discriminative state estimators [9]. Inspired by the success of these methods, we propose a few deep learning based approaches for predicting dynamic random processes. In this dissertation, we consider three settings:

1. **Oscillator Phase Predictions**: Oscillator stability has been traditionally characterized by the Allan variance and stochastic models originally developed for high precision, high cost sources such as atomic clocks. Knowledge of model parameters allows development of tracking and prediction techniques which enable accurate prediction of and compensation for oscillator drift.

   The KF [37] provides the minimum mean squared error solution to linear system when the process under observation is completely represented by the state model. However, its success relies on the knowledge of the system models. When the models are unknown or partially known, it is hard to determine the covariances of

the process and measurement noises [38].

This can happen when we do not have high-end oscillators for tracking. The traditional model-based approach does not adapt to the inherent parametric mismatch. We approach this from a machine learning point-of-view and develop methods that can solve this problem.

2. **Maneuvering Target Tracking**: Target tracking is used in many practical applications to accurately track objects with trajectories that have significant position derivatives of several orders. When not detected and compensated, the maneuvers can degrade the performance of the tracker and might lead to filter divergence [39]. A Kalman Filter (KF), or its non-linear variants such as Extended Kalman Filter (EKF) or particle filter, is commonly employed for tracking maneuvering targets [40].

   Accurate tracking is possible only when we model the target motion appropriately. Some common dynamic models are constant velocity (CV) model and a coordinated turn (CT) model [41]. However, a single model is not always adequate to accurately describe the target's motion. In an Interacting Multiple Model (IMM) filter, two or more models are considered, and the model inaccuracy is addressed by facilitating an interaction between them for different modes at the beginning of each filter cycle [39]. These are weighed accordingly by the conditional probabilities of switching between model modes. However, the IMM is a heuristic algorithm designed to yield a good performance only within the class of a fixed structure algorithms. It is not known to be generally optimal and does not guarantee robust performance and may give unsatisfactory results for particular scenarios such as nonlinear target dynamics during a turn [42] and sudden starts and stops of maneuvers such as model switching [43].

   We consider this problem of tracking maneuvering targets and approach it in two ways

   (a) **Hybrid Algorithms**: Developing algorithms that *assist* Kalman filters and

IMMs in tracking and predicting states. We demonstrate a dominance of hybrid models over traditional model-based methods and time-series forecasting methods such as LSTMs.

(b) **Machine Learning Algorithms**: Developing algorithms that *replace* Kalman filters. We specifically resort to a temporal convolutional network (TCN) here.

3. **Tracking Dynamical Processes on Gilbert-Elliott Channels**: Switching dynamical processes can be seen as an extension of hidden Markov models (HMMs) in which each HMM state, or mode, is associated with a dynamical process. Existing methods for learning switching processes rely on fixing the number of HMM modes [39]. We consider a time-varying linear state-space system, typically driven by a discrete Markov chain. Such systems are typically used to describe system dynamics subject to sudden shifts or modal changes governed by stochastic switching. They are modeled as an ensemble dynamical system where the modes of operation are governed by a Markov chain [44]. While the transition probabilities between the modes are often assumed to be known, the exact location of the jumps is not guaranteed to be known.

   We consider an application that demonstrates this: channel tracking for Gilbert-Elliott channels [45] and apply a TCN to handle model switching in Chapter 7.

1.3 Overview of the Dissertation

This section gives an overview of the dissertation and details the chapter-wise contributions starting with Chapter 3 to 7. Chapter 2 provides the necessary background for this dissertation and reviews digital communication systems and dynamical systems. The main body of this dissertation is divided into two parts:

1.3.1 Part I: Machine Learning for Communication Systems

Chapter 3

Chapter 3 introduces an arbitrary autoencoder architecture for joint channel coding and modulation for an end-to-end digital communication system in AWGN channels. This is compared against various coding schemes such as Hamming and Golay codes with modulations with Phase Shift Keying (PSK) such as Binary PSK (BPSK), Quadrature PSK (QPSK), 8-PSK and Quadrature Amplitude Modulations (QAM) such as 16-QAM and 64-QAM. We also delve further into conditional block error rate for Hamming$(7, 4)$+BPSK and Hamming$(15, 11)$+BPSK to gather new insights about the geometry of the codewords generated by the autoencoder. Here are the contributions:

1. We discover that the transmitter side of the autoencoder can just be an embedding and for AWGN channels, the receiver side can just be a matched filter. This can improve training speed, reduce model size and increase the overall efficiency of block codes. It also helps us to solve an end-to-end optimization problem for longer blocklength codes.
2. We also discover a parsimonious matched-filter based receiver architecture for AWGN channels with efficient parameterizations for linear block codes that reduces the number of paramaters.
3. We discover that minimum distance is not a good predictor of BLER. The codes learned by autoencoder have worse minimum distance than classic codes with similar modulations, but they perform better because most of the codewords have better distance properties.
4. We discover that we need to train at an SNR with sufficient numbers of block errors. Otherwise, the training will be slowed down. However, if we train at one SNR, the code will work at all SNRs. Hence, there is no need to have a family of codes for different SNRs.
5. We discover that training at different SNRs may lead to different codes. This is

not because different codes are better at different SNRs, but because the objective function is multimodal and we converge to local maxima for each random initialization.

6. We present a detailed analysis on conditional block error rates for linear block codes and present insights on the geometry of autoencoders and answer the question on how have an edge over a typical decoding approach.

Chapter 4

Chapter 4 extends our autoencoder-based approach to other non-canonical channels such as Bernoulli-Gaussian Impulsive Noise (BGIN) channel for the coding and modulation schemes considered in Chapter 3. Here are the contributions:

1. We propose a family of autoencoders rather than employing heuristic techniques in impulsive noise channels to minimize the BLER.
2. We also discover a parsimonious architecture for BGIN channels with efficient parameterizations for linear block codes that reduces the number of parameters.
3. Numerical results show that the trained autoencoder uniformly outperforms classical block codes with BPSK modulation in the BGIN channel even when impulsive noise mitigation techniques such as blanking and clipping are employed.

1.3.2 Part II: Machine Learning for Dynamical Systems

Chapter 5

Chapter 5 introduces the oscillator phase prediction problem under parametric mismatch. We propose alternative solutions based on ML and compare it with Riccati equation and least squares fit. Numerical results show that by using a CNN, we can achieve a performance similar to steady state. Here are the contributions:

1. We formulate a dynamical system as a time-series forecasting problem and design and develop data-driven approach to predict dynamic random processes where we

use state observations to predict the next internal state.

2. We show a proof-of-concept demonstration with oscillator phase predictions and study the performance of a Kalman filter that uses mismatched parameters to better understand the sensitivities of the Kalman filter to parameter mismatches.
3. We use a *circular mean-squared-error* loss function to predict phases as opposed to a regular mean squared error function.

Chapter 6

Chapter 6 focuses on tracking with Kalman filter model switching and considers a specific application: maneuvering target tracking, where the target switches between a linear constant velocity mode and a nonlinear coordinated turn mode. We develop neural network algorithms that dynamically adapt to maneuvering target tracking. Here are the contributions:

1. We propose an alternate approach using a Temporal Convolutional Network (TCN) and demonstrate its performance with numerical results.
2. We also propose a surrogate hybrid model by appending autoencoders with Kalman filter, specifically introducing Autoencoder Interacting Multiple Model (AEIMM) and demonstrate its performance with numerical results.
3. We apply domain randomization for designing a robust learning model and avoid overfitting.
4. We show numerical demonstrations of AEIMM outperforming both model-based and learning-based approaches for the system model 6.2.

Chapter 7

Chapter 7 focuses on tracking with Kalman filter model switching and considers a different application: a Gilbert-Elliott burst noise communication channel that switches between two different modes, each modeled as a linear system. Here are the contributions:

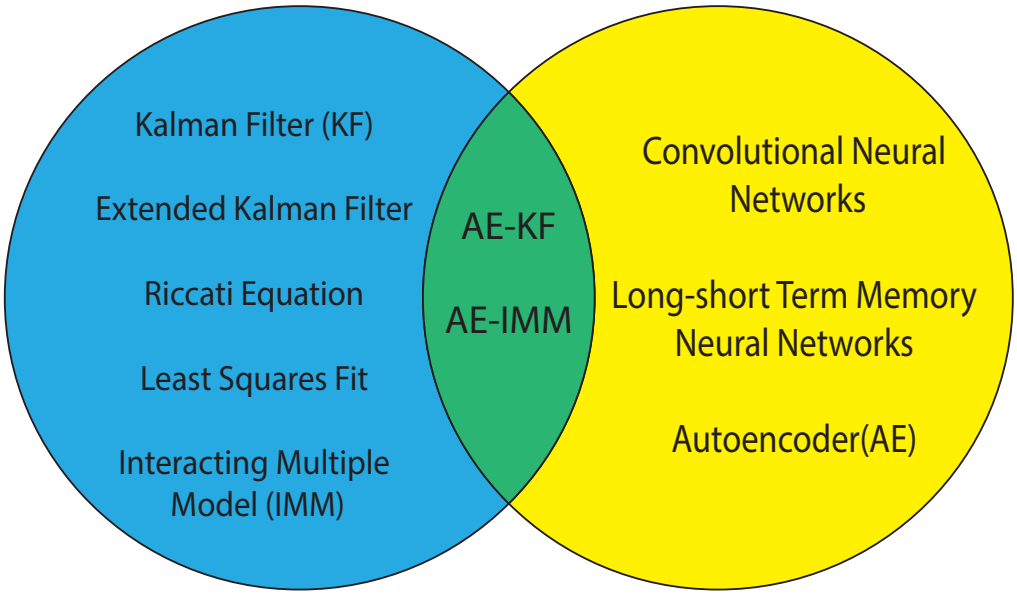

Figure 1.3: Hybrid models in Chapter 6.

1. Numerical simulations demonstrate that a TCN, which is considered in Chapter 6, outperforms classical tracking methods.
2. We discover that the TCN tends to identify a mode switch faster than an IMM and that, in some cases, the TCN can perform almost as well as an omniscient Kalman filter with perfect knowledge of the current mode of the dynamical system.

This is followed by a conclusion and a discussion of potential research for both parts of the dissertation.

1.4 Publications

1. A. Grootveld, **K. Vedula**, V. Bugayev, L. Lackey, D.R. Brown III, and A.G. Klein. *Tracking of Dynamical Processes with Model SwitchingUsing Temporal Convolutional Networks.* Submitted to the 2021 IEEE Aerospace Conference (AEROCONF 2021), Big Sky, Montana, Mar 6-13, 2021. In review.
2. **K. Vedula**, M.L. Weiss, R.C. Paffenroth, J.R. Uzarski, D.R. Brown III *Maneuvering Target Tracking using Autoencoder Interacting Multiple Model Filter*, 54th Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA,

November 3-6, 2020.

3. **K. Vedula**, R.C. Paffenroth, and D.R. Brown III. *Joint Coding and Modulation in the Ultra-Short Blocklength Regime for Bernoulli-Gaussian Impulsive Noise Channels Using Autoencoders*, 45th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2020). Barcelona, Spain, May 4-8, 2020.
4. **K. Vedula**, D.R. Brown III *Deep Learning for Predicting Dynamic Random Processes with Parametric Mismatch*, Technical Report, June, 2018.
5. J. McNeill, S. Razavi, **K. Vedula**, and D.R. Brown III. *Experimental Characterization of Oscillator Stability for Carrier Phase Synchronization.* Proceedings of the 2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Torino, Italy, May 22-25, 2017.

Chapter 2

Background: Communication Systems and Dynamical Systems

In this chapter, we provide the required background on communication systems and dynamical systems. The ideas discussed in this chapter are used in parts I and II of this dissertation. We divide this chapter into two parts:

1. Section 2.1 provides the relevant background information on traditional communication systems. We compare our machine learning approaches against some traditional techniques like hard decision and soft decision decoding and asymptotic finite blocklength coding bounds. We also briefly discuss different kinds of modulations such as Pulse Amplitude Modulation, Phase Shift Keying and Quadrature Amplitude Modulation.
2. Section 2.2 discusses the basics of dynamical systems and presents a few methods to solve for optimal solutions such as Kalman filter, least squares fit, discrete-time Riccati equation and Interacting multiple model.

2.1 Communication Systems

We introduce channel coding in Section 2.1.1, energy and power constraints in Section 2.1.2. Then, we discuss about Hamming codes and signal detection techniques such as hard decision and soft decision decoding. Then, we discuss finite blocklength coding bounds in Section 2.1.5. Section 2.1.6 briefly discusses different kinds of modulation. Finally, in Section 2.1.2, we discuss different kinds of energy constraints.

Figure 2.1 shows a traditional approach for a typical communication system that wishes to send a message from point A to B. Each communication system block is optimized individually for a different functionality. The classical approach at the transmitter is to provide a block of k bits at the input of the channel encoder, map these bits to an n-bit codeword, and then map this codeword to m real-valued symbols for transmission through the channel. This is done by upsampling and pulse shaping. The channel can add various kinds of impairments such as additive or multiplicative noise. Similarly, at the receiver, the noisy symbols are first filtered and synchronized so that they are aligned with the transmitted symbol. Then they are demodulated and channel decoding is performed either on the soft demodulator outputs or on the hard decisions from the demodulator. We briefly detail each step in the following sections. For a detailed treatment, we refer the reader to a standard communication systems textbook [16].

2.1.1 Channel Coding

Source coding is used for compression and removing the redundancy from sources, such as compressing into JPEG pictures. However, we also need channel coding to add redundancy to a message to make it more robust against noise and reliably communicate to the receiver. This provides large gains in system efficiency. For a binary code, the code rate R is defined as

$$R = \frac{k}{n} \tag{2.1}$$

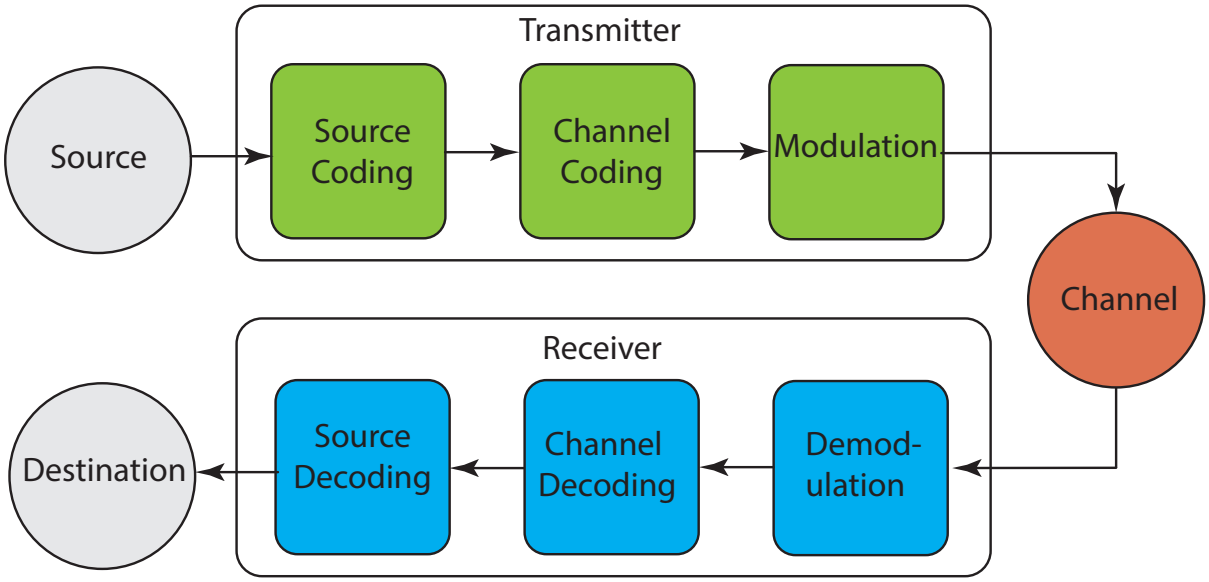

Figure 2.1: Traditional block-by-block approach for a communication system.

where n is the number of bits in the codeword, and k is the number of bits in the message.

As an example, we can take a repeat code which takes a 1-bit message $u \in \{0, 1\}$and maps it to codeword of length 5 by repeating the message bit. This gives a binary codeword $00000, 11111$. The code rate for this is $R = \frac{1}{5}$.

As in Figure 2.2,our goal is to send a message - in this case, 1001 to the destination. To make sure that the message would reach correctly, we send each bit thrice. This is called *repetition coding*, and it is one of the simplest forms of channel coding. If 1 is the message, 111 is the *codeword*. Similarly, if we list all the codewords for all the possible messages, we get a *codebook*. We also use terms like *embedding* and *constellation* to mean the same concept. The process of modulation turns into a cosine wave, where 1 would be represented by a sinusoidal wave with a positive magnitude, 0 would be represented by a sinusodial wave with a negative magnitude. Typically, when we add Gaussian noise to the vector that goes through the channel, it makes it blurry. That is, it puts the symbol on a point cloud and modulates it onto a BPSK constellation i.e. we modulate it to a sinusoidal wave with $+1$, another sinusoidal wave with -1 as magnitude and adds noise to it. If we compare the sent and received codewords - we see that only 3 out of 4 bits arrived properly in one block, and there is an error in one bit, hence error in the block.

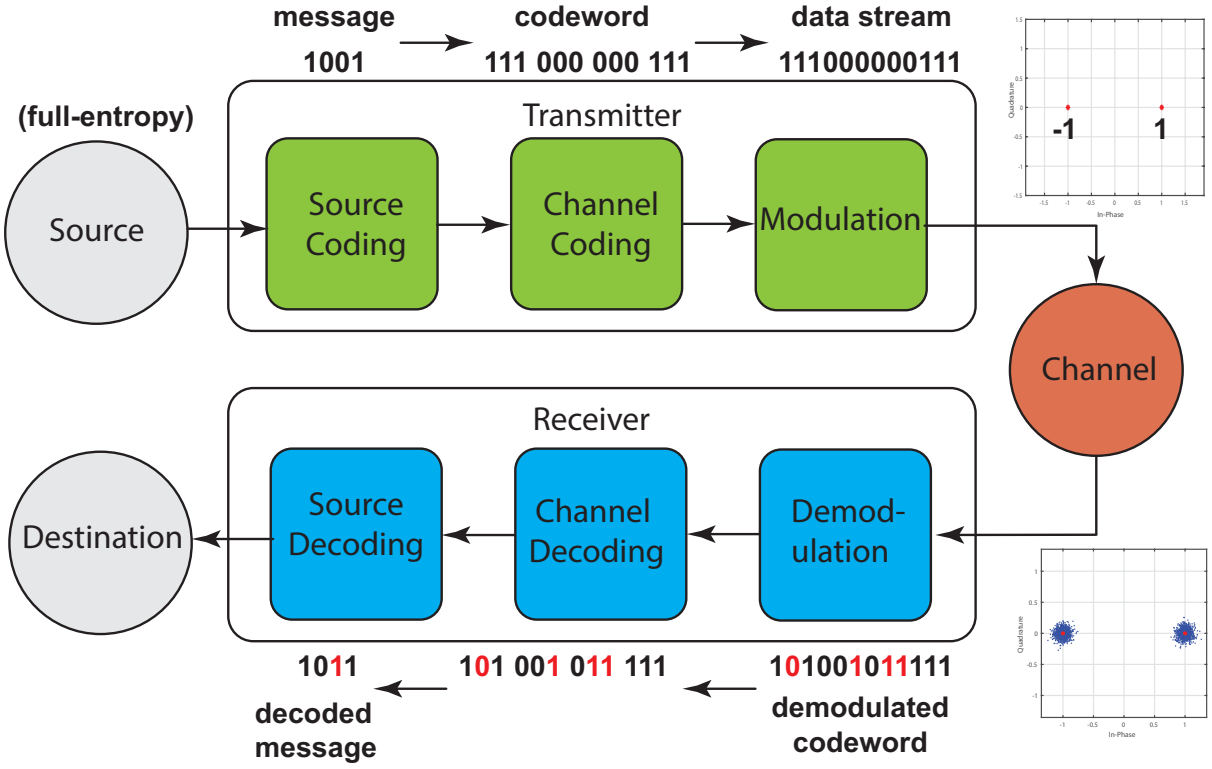

Figure 2.2: Working example for an end-to-end traditional communication system.

So the block error rate (BLER) is 1/4. Similarly, if we look at the sequence on the whole, we only lose a bit - so the bit error rate (BER) is 1/16.

2.1.2 Energy and Power of Codewords

The signal-to-noise ratio (SNR) of communication system can be defined in terms of the *energy per information bit* $\mathcal{E}_b$, and the average *energy per transmitted symbol* $\mathcal{E}_s$.

The modulator in a standard digital communication system takes k bits of information (binary symbols) and maps to a set of corresponding waveforms $s_m(t), 1 \leq m \leq M, M = 2^k$. Let $\mathcal{E}_m$ be the energy of the waveform $s_m(t)$. Now, the average signal energy is given by

$$\mathcal{E}_s = \sum_{i=1}^{M} p_m \mathcal{E}_m \tag{2.2}$$

where p_m indicates the probability of the m^{th} signal. Assuming all messages are

equiprobable, we can set $p_m = 1/M$. Then,

$$\mathcal{E}_s = \frac{1}{M} \sum_{i=1}^{M} \mathcal{E}_m \tag{2.3}$$

The average energy for transmission of one bit of information, or average energy per bit, when the signals are equiprobable is given by

$$\mathcal{E}_b = \frac{\mathcal{E}_s}{\log_2 M} \tag{2.4}$$

If a communication system is transmitting an average energy of $\mathcal{E}_b$ per bit, and it takes T seconds to transmit this average energy, then the average power sent by the transmitter [16] is given by

$$P = \frac{\mathcal{E}_b}{T} = R\mathcal{E}_b \tag{2.5}$$

where R is the code rate of the signal.

2.1.3 Hamming Code

Hamming codes were one of the earliest block codes used to detect errors in the calculations of the relay-based computers at the time. They are characterized by the structure $(n, k) = (2^n - 1, 2^n - 1 - m)$ where $m = 2, 3, \ldots$. They can detect up to all combinations of 2 or fewer errors within a block. For example, the $(7, 4)$ binary Hamming Code has $n = 7$, $M = 16$, and $d_{min} = 3$. The code can be defined in terms of a Venn diagram showing three partially overlapping sets. Each of the seven subregions represent a code bit and the three circles represent even parity constraints. Any single error can be corrected by observing each bit error gives a unique pattern of parity violations. The codewords

can be listed as follows:

$$\begin{array}{cccc} 0000000 & 0100110 & 1000011 & 1100101 \\ 0001111 & 0101001 & 1001100 & 1101010 \\ 0010101 & 0110011 & 1010110 & 1110000 \\ 0011010 & 0111100 & 1011001 & 1111111 \end{array} \tag{2.6}$$

2.1.4 Optimal Symbol Detection

Let $H_0, H_1, \ldots, H_{m-1}$ be m different hypotheses associated with a random observation Y. The probability of a hypothesis before the observation, $\Pr(H_i)$, is called the *a priori probability*. For each hypothesis, the connection with Y is defined by the *observation probability* $\Pr(Y = y \,|\, H_i)$. The goal is to choose a decision function $D(y)$ which minimizes the decision error probability for any observation. This is equivalent to maximizing the probability that the decision is correct. If $Y = y$, then the probability that hypothesis H_i is correct is given by its *a posteriori probability* $\Pr(H_i \,|\, Y = y)$. Therefore, one finds that the optimal choice is the *maximum a posteriori probability* (MAP) decision rule.

Hard Decision Decoding

Hard Decision Decoding sets a threshold on the received signal and decodes each bit by considering it as 1 or 0. Suppose a BPSK signal is transmitted through our discrete-time AWGN channel model. The detector must decide whether a 0 or 1 was transmitted. So, the decision region $D(y)$ is

$$D(y) = \begin{cases} 0 & \text{if } y \geq 0 \\ 1 & \text{if } y < 0 \end{cases}. \tag{2.7}$$

This detector is optimal if 0s and 1s are transmitted with equal probability. In general, if the code bits are transmitted over an AWGN channel using BPSK followed by a hard-

decision detector, then we have

$$\Pr(Y = y \,|\, H_i) = Q\left(\sqrt{\frac{2\mathcal{E}_s}{N_0}}\right) = Q\left(\sqrt{\frac{2R\mathcal{E}_b}{N_0}}\right). \tag{2.8}$$

which we can use to make fair comparisons between coding systems with different rates.

Soft Decision Decoding

Soft Decision Decoding determines the maximum likelihood estimate by computing the correlations on the signal. We consider the decision function

$$D(y) = \underset{i\in\{0,\ldots,m-1\}}{\arg\max} \Pr(H_i \,|\, Y = y). \tag{2.9}$$

and compute the probabilities with Bayes' rule using the a priori probabilities and observation probabilities. This gives

$$\Pr\left(H_i|Y = y\right) = \frac{\Pr(H_i)\Pr(Y = y|H_i)}{\sum_{j=0}^{m-1}\Pr(H_j)\Pr(Y = y|H_j)}. \tag{2.10}$$

The denominator of this expression is the same for all i, the MAP rule can be simplified to

$$D(y) = \underset{i\in\{0,\ldots,m-1\}}{\arg\max} \Pr(H_i)\Pr(Y = y|H_i). \tag{2.11}$$

We can also define a *maximum likelihood* (ML) decision rule

$$D(y) = \underset{i\in\{0,\ldots,m-1\}}{\arg\max} \Pr(Y = y|H_i), \tag{2.12}$$

which ignores the a priori probability. *When all the hypotheses have the same a priori probability, ML and MAP are identical.*

Now, for a system which transmits BPSK over an AWGN channel, let H_0 be the hypothesis that 0 was sent and H_1 be the hypothesis that 1 was sent. For binary hypothesis

problems, the MAP decision rule can be written as

$$\Pr(H_0)\Pr(Y=y|H_0) \underset{H_1}{\overset{H_0}{\gtrless}} \Pr(H_0)\Pr(Y=y|H_0), \tag{2.13}$$

If $\Pr(H_0) = 1-p$ and $\Pr(H_1) = p$, then one can substitute to rewrite this as

$$(1-p)\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(y-1)^2/(2\sigma^2)} \underset{H_1}{\overset{H_0}{\gtrless}} p\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(y+1)^2/(2\sigma^2)} \Rightarrow y \underset{H_1}{\overset{H_0}{\gtrless}} \frac{\sigma^2}{2}\ln\frac{p}{1-p}. \tag{2.14}$$

2.1.5 Finite Blocklength Coding Bounds

A theoretical analysis of the interplay between block-error probability, communication rate, and block size is required in some applications. Hence, nonasymptotic achievability and converse bounds on the *maximum coding rate* for several channel models that are relevant for wireless communication systems, such as the AWGN channel and the Rayleigh block-fading channel. In our case, we want to compare the performance of the autoencoder approach with the finite blocklength coding bounds.

In this section, we briefly review some of these finite length bounds. They will serve as benchmark curves on the codes designed by the autoencoder. We also consider the converse and achievability bounds on block error rate (BLER) based on finite-blocklength information theory to benchmark the Hamming codes considered here and to characterize the maximum code rate achievable for a given blocklength. First, we introduce Shannon's sphere packing bound (SPB) [46].

Sphere Packing Bound

Shannon introduces the sphere packing argument that a randomly picked Voronoi cell does not exhibit a better probability of error than a circular cone of the same solid angle. This claim is based on propositions stating that among the cones of a given solid angle, the circular one provides the lowest probability of error and it is best to share the total solid angle evenly between all Voronoi cells [46].

This helps us to evaluate the performance limits of block codes over an AWGN channel by providing a lower bound on the block error rate for any code whose codewords lie on a spherical shell. Typically, during detection, an error occurs if the received sequence falls outside the Voronoi region on the plane of detection that corresponds to the transmitted signal point.

However, sphere packing bound just assumes that the signals have equal energy and does not take into account of their modulation. Tighter bounds such as Gallager's random coding bound (RCB) and random coding union bound (RCU) are built upon this for quantifying the sub-optimality of error-correcting codes associated with their decoding algorithms.

Random Coding Union Bound

The achievability bounds are based on the random-coding union bound (RCU) [47] that denotes the upper bound on the average probability of error attained by an arbitrary codebook using a maximum likelihood decoder. This can be obtained by analyzing the average behavior of random coding and maximum-likelihood decoding.

Normal Approximation

We also consider a normal approximation on BLER based on the Berry-Esseen theorem by calculating the channel SNR $2R\mathcal{E}_b/N_0$ and then computing the dispersion

$$V = \frac{2R\mathcal{E}_b/N_0(2 + 2R\mathcal{E}_b/N_0)}{2(1 + 2R\mathcal{E}_b/N_0)^2} \tag{2.15}$$

and then, for n is the codeword length and R is the code rate, the probability of error for the normal approximation comes to

$$P_e = Q\left(\sqrt{\frac{n(C-R)}{V\log_2 e + \frac{\log_2 n}{n}}}\right) \tag{2.16}$$

Metaconverse Bound

The metaconverse bound is based on the metaconverse theorem [48] in channel coding, where one of the M equiprobable messages is sent through the channel with a codebook. Since there is a codeword for each message, the input distribution induces a new distribution at the encoder and an estimate of an input distribution at the decoder. Posing this as an M-ary hypothesis testing problem, the meta-converse bounds gives a lower bound on BLER. It provides a simultaneous generalization for previously known converse bounds in the literature and yields the best converse bound known for channels without feedback [49].

2.1.6 Signal Constellations

After forward error correction encoding, the bits are mapped to constellation symbols. An M-ary constellation is a set of M points that are used for the pulse shaping. The number M of points is usually chosen as a power of two. Once a channel model has been defined, the next step is choosing how to transmit digital data through the channel. *Modulation* converts a string of bits into a signal suitable for transmission over a communication channel. *Demodulation* transforms the information symbols are extracted from the received signal.

Modulating to a high-frequency carrier allows two independent signals to be modulated onto the same carrier frequency - one onto the sine wave and the other onto the cosine wave. This allows us to treat the transmitted value x_n and received value y_n as points in 2-dimensional space. The set $\mathcal{X}$ of possible transmitted points in 2-dimensional space is called the *symbol constellation* or *symbol embedding* or a *symbol codebook*.

Constellations are typically defined by choosing the set of channel input values $\mathcal{X}$, and then choosing the mapping function $M : \mathcal{U} \to \mathcal{X}$. These are represented by complex numbers $\mathbb{C}$ and the constellation is a subset $\mathcal{X} \subset \mathbb{C}$. Likewise, the transmitted symbol is $x_n \in \mathbb{C}$ and the received value is $Y_n \in \mathbb{C}$.

Since we are transmitting complex signals, the noise term z_n consists of two i.i.d. Gaussian random variables, one in each direction. The joint probability distribution is given by

$$p(y_n^{(re)}, y_n^{(im)}) = \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(y_n^{(re)} - x_n^{(re)}\right)^2/(2\sigma^2)}\right)\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(y_n^{(im)} - x_n^{(im)}\right)^2/(2\sigma^2)}\right) \tag{2.17}$$

$$= \frac{1}{2\pi\sigma^2} e^{-|y_n - x_n|^2/(2\sigma^2)}. \tag{2.18}$$

So, the probability of receiving a y_n value is simply a function of its Euclidean distance $\sqrt{|y_n - x_n|}^2$ from the actual transmitted symbol. This leads to a geometric characterization of the optimal decision regions for the detector.

We now discuss three kinds of modulation schemes which give distinct constellations.

Pulse Amplitude Modulation

Pulse amplitude modulation (PAM) embeds data in the amplitude of a single waveform $u(t) = u\phi(t)$. We can segment the data into blocks of k bits and each block is mapped into one of $2^k = M$ possible real numbers within the constellation set

$$\mathcal{A} = \left\{-\frac{d(M-1)}{2}, \ldots, -\frac{d}{2}, \frac{d}{2}, \ldots, \frac{d(M-1)}{2}\right\}. \tag{2.19}$$

where d is the distance between adjacent points. The sent message is recovered by taking the inner product of $u(t)$ with the basis element $\phi(t)$,

$$u = \langle u(t), \phi(t)\rangle = \int_{\mathbb{R}} u(t)\phi^*(t)dt, \tag{2.20}$$

Generalizing to an M-PAM system where $x \in \{(-M+1)c, \ldots, -c, c, \ldots, (M-1)c\}$. Each x is has a different amount of energy and is also associated with $\log_2 M$ bits. We

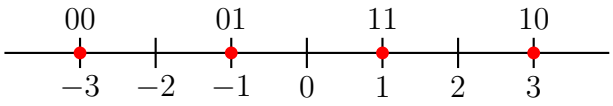

Figure 2.3: An example of a 4-PAM constellation.

can compute the *average energy per symbol* as

$$\mathcal{E}_s = \frac{1}{M} \sum_{m \in \mathcal{M}} c^2 m^2 = \frac{c^2(M^2-1)}{3} \tag{2.21}$$

where $\mathcal{M} = \{-M+1, -M+3, \ldots, -1, 1, \ldots, M-3, M-1\}$. The energy per bit then follows as

$$\mathcal{E}_b = \frac{\mathcal{E}_s}{\log_2 M} = \frac{c^2(M^2-1)}{3\log_2 M} = \frac{c^2(4^k-1)}{3k}. \tag{2.22}$$

Given $\mathcal{E}_b$ and M (or k), we can find c so that the M-PAM constellation is scaled properly and has the correct $\mathcal{E}_b$. Different symbols have different energies, but they are transmitted with equal probability. c is chosen so that the *average* energy per symbol is $\mathcal{E}_s = k\mathcal{E}_b$. Roughly speaking, for large PAM constellations, the energy consumption per bit doubles with every additional bit. Figure 2.3 shows an example constellation.

For 4-PAM, we use four symbols $\mathcal{X} = \{-3, -1, 1, 3\}$. Typically, we center them around 0 to minimize the transmitted energy. The decision function is

$$D(y) = \begin{cases} 00 & \text{if } y < -2 \\ 01 & \text{if } -2 \le y < 0 \\ 10 & \text{if } 0 \le y < 2 \\ 11 & \text{if } y \ge 2 \end{cases}. \tag{2.23}$$

Quadrature Amplitude Modulation

QAM modulates baseband waveforms with sinusoid carriers. The in-phase $\cos(2\pi f_c t)$ and quadrature $\sin(2\pi f_c t)$ components of the bandpass signal correspond to the real and

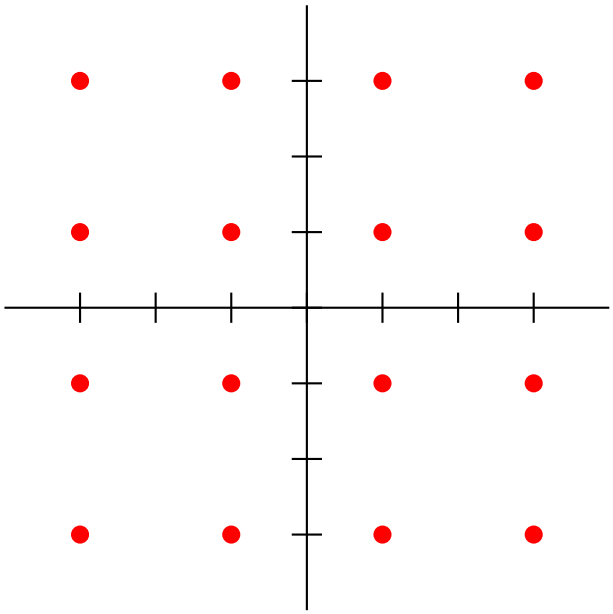

Figure 2.4: Symbol constellations $\mathcal{X}$ for 16-QAM.

imaginary parts of the baseband waveform. The standard QAM constellation with 16 points (known as 16-QAM) is given by

$$\mathcal{X} = \left\{ a + bi \,\middle|\, a, b \in \{-3, -1, 1, 3\} \right\} \subset \mathbb{C}. \tag{2.24}$$

The average energy of this constellation is given by

$$\begin{aligned} \mathcal{E}_s &= \frac{1}{16} \sum_{a,b \in \{-3,-1,1,3\}} (a^2 + b^2) && (2.25) \\ &= \frac{8}{16} \sum_{a \in \{-3,-1,1,3\}} a^2 && (2.26) \\ &= 10. && (2.27) \end{aligned}$$

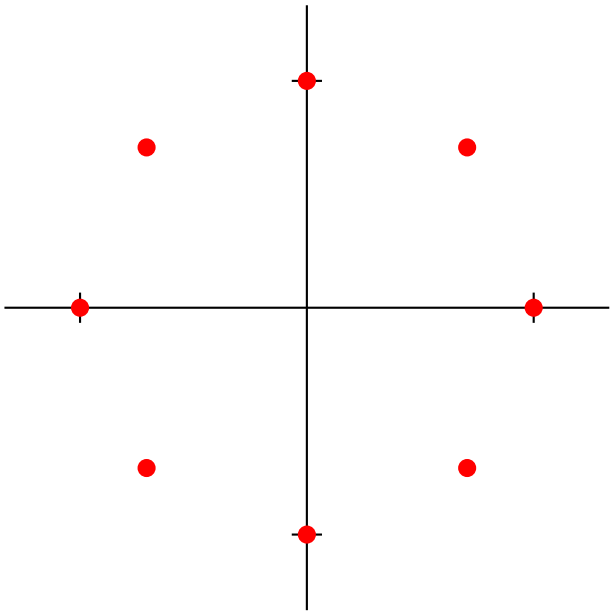

Figure 2.5: Symbol constellations $\mathcal{X}$ for 8-PSK.

Phase Shift Keying

PSK maps the constellation points to be equally spaced around a circle as shown in Figure 2.5. For M points, we have

$$\mathcal{X} = \left\{ e^{2\pi i k/M} \,\middle|\, k \in \{0, 1, \ldots, M-1\} \right\} \subset \mathbb{C}. \tag{2.28}$$

where the data is embedded in the sinusoidal phase θ.

2.1.7 Constellation Shaping

We also consider constellation shaping in this disseration in the context of per-codebook energy constraint imposed on the autoencoder. Constellation shaping describes the optimization of a modulation format with equidistant and equiprobable signal points towards some shape that is tailored to the transmission channel. Geometric shaping optimizes the location of constellation points. However, a few alternate approaches exist in the literature namely:

- **Adaptive modulation and coding (AMC)**: This is widely employed in modern wireless communication systems to improve the transmission efficiency by adjusting the transmission rate according to channel conditions. It can provide very efficient use of channel resources especially over fading channels [50].
- **Minimum energy coding (MEC)**: This maps the messages to codewords such that the average codeword energy is minimized under asymmetric modulation assumption. Here, channel symbols with smaller energy are mapped to 0 leading to a reduced energy dissipation because transmitting 0 requires less energy than 1. Codeword weights and original message-codeword mappings are chosen such that the expected code weight is minimized at the cost of increased codeword length [51].
- **Probabilistic Amplitude Shaping (PAS)**: This modifies the probability of the constellation symbols, which remain on a square grid. Classic PS schemes are for example based on many-to-one mappings, trellis shaping, and shell mapping [52].

2.1.8 Types of Energy Constraints

Normalization can be enforced in a number of ways such as putting the constraint on mean amplitude, mean power, maximum power, or other similar constraint, yielding quite different results for each in some cases. This can also be done on a per-symbol or per-batch level. Here, we consider three possible kinds of energy constraints.

A codebook $\boldsymbol{C} \in \mathbb{R}^{M \times M}$ is a map from the set $\mathcal{C}$. The codewords of the block code represented by the matrix $\boldsymbol{C}$ are in the rows of $\boldsymbol{C}$, i.e.,

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_M \end{bmatrix} = \begin{bmatrix} c_{1,1} & \dots & c_{1,M} \\ \vdots & & \\ c_{M,1} & \dots & c_{M,M} \end{bmatrix} \tag{2.29}$$

where the codeword $\boldsymbol{c}_i \in \mathbb{R}^{1 \times M}$.

Assuming the same value of $\mathcal{E}$ in all three cases, note that each constraint is succes-

sively less restrictive. In other words, satisfying the per-element energy constraint automatically satisfies the per-codeword and per-codebook constraints. Satisfying the per-codeword energy constraint automatically satisfies the per-codebook energy constraint, but may not satisfy the per-element energy constraint. This is also depicted in the Venn diagram in Figure 2.6.

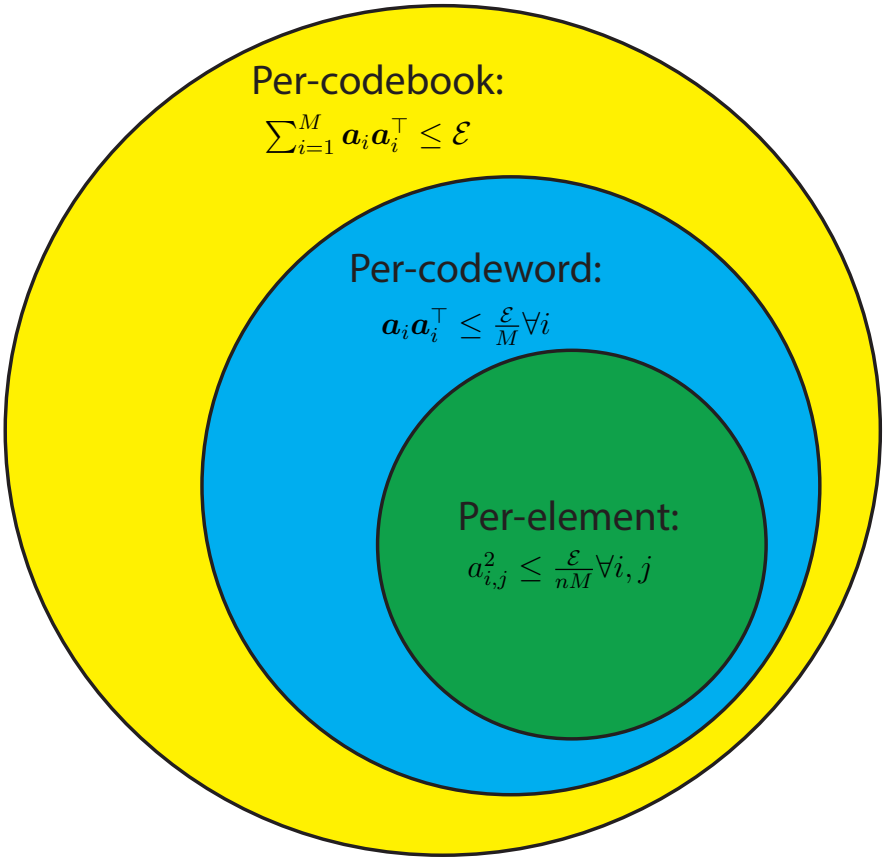

Figure 2.6: Energy constraints on a codebook for linear block codes.

Per-element energy constraint

We can impose a constraint on each individual element as shown in equation

$$c_{i,j}^2 \leq \frac{\mathcal{E}}{M^2} \tag{2.30}$$

for all i, j. This restricts every element in the codebook to have the same energy. This is typically observed in a Quadrature Phase Shift Keying (QPSK) modulation where all

elements are required to have energy 1. This is shown in Figure 2.7.

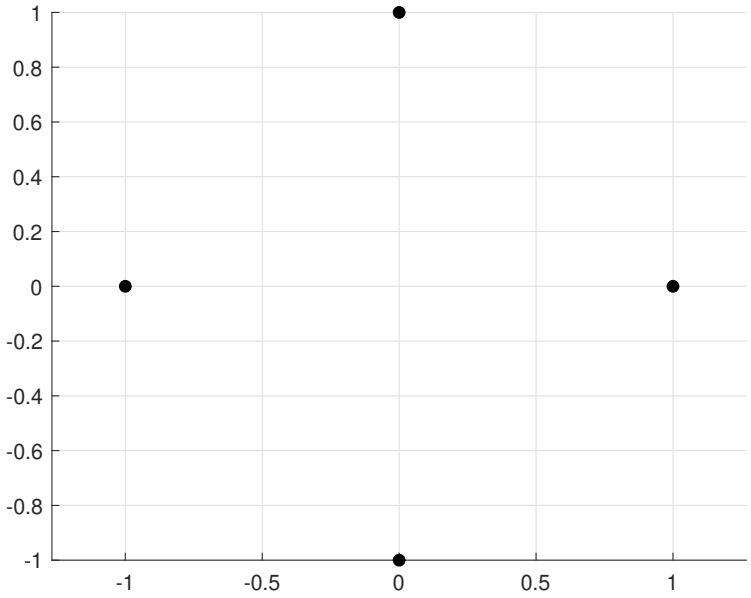

Figure 2.7: Example of per-element energy constraint.

Per-codeword energy constraint

Alternatively, we can impose a constraint on each codeword as shown in equation

$$\boldsymbol{c}_i \boldsymbol{c}_i^{\top} \leq \frac{\mathcal{E}}{M} \tag{2.31}$$

for all i.

Here, each codeword is free to choose a point on the circle, but it is restricted to lie on that circle. 8-PSK modulation is an example of per-codeword energy constraint. This is shown in Figure 2.8.

Per-codebook energy constraint

With this kind of constraint, we have the freedom to map the channel symbol with smaller energy cost to 0 symbol. The per-codebook energy constraint is equivalent to an

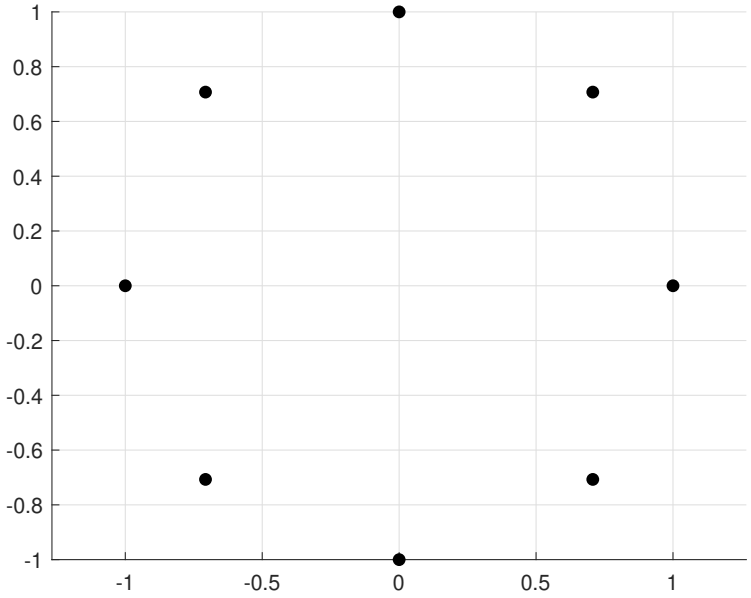

Figure 2.8: Example for a per-codeword energy constraint.

average-codeword energy constraint since

$$\sum_{i=1}^{M} \boldsymbol{c}_i \boldsymbol{c}_i^{\top} \leq \mathcal{E} \quad \Leftrightarrow \quad \underbrace{\frac{1}{M} \sum_{i=1}^{M} \boldsymbol{c}_i \boldsymbol{c}_i^{\top}}_{\text{average codeword energy}} \leq \frac{\mathcal{E}}{M} \tag{2.32}$$

$$\sum_{i=1}^{M} \boldsymbol{c}_i \boldsymbol{c}_i^{\top} \leq \mathcal{E} \tag{2.33}$$

A standard QAM constellation is a good example for per-codebook energy constraint, as shown in Figure 2.9.

The per-codebook energy is equal to the squared Frobenius norm, i.e.,

$$\sum_{i=1}^{M} \boldsymbol{c}_i \boldsymbol{c}_i^{\top} = \|\boldsymbol{C}\|_{\mathrm{F}}^2 \leq \mathcal{E} \tag{2.34}$$

hence, given any non-zero matrix $\boldsymbol{C}$, we can scale it to satisfy the per-codebook energy constraint by computing

$$\bar{\boldsymbol{C}} = \frac{\sqrt{\mathcal{E}}}{\|\boldsymbol{C}\|_{\mathrm{F}}} \boldsymbol{C}. \tag{2.35}$$

We finish the discussion on the ideas pertaining to Part I. These ideas will be used in

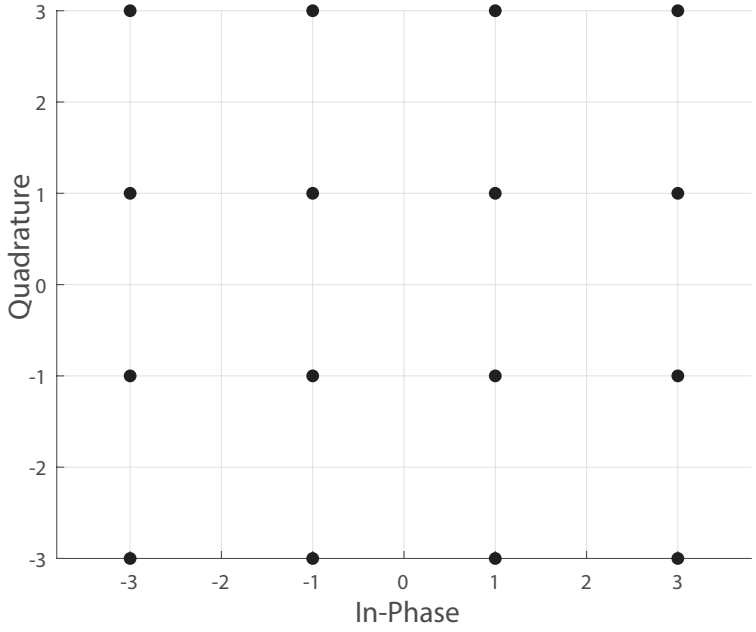

Figure 2.9: Example for a per-codebook energy constraint.

Chapters 3 and 4. We now move on to ideas pertaining to Part II of the dissertation.

2.2 Dynamical Systems

In this section, we start with introducing the notion of state, observations and tracking. In Section 2.2.1, we discuss the steps involved in determining the optimal estimates and predictions using a Kalman Filter. Then, we discuss other techniques like least squares fit in Section 2.2.2 and discrete-time Riccati equation in Section 2.2.3. Then, we consider a situation with model switching and discuss the implementation details of an interacting multiple model (IMM) filter in Section 2.2.4.

A *state* is a set of measured and estimated target parameters. A *track* is a state trajectory estimated from a set of measurements that are assumed to be from the same target. *Tracking* involves processing of noise-corrupted *observations* obtained from a target in order to maintain an estimate of its current state.

For example, in a position tracking application, the position and velocity of the target could be the states. We aim to estimate/predict these states from a set of noisy

measurements. There are many other problems that require us to estimate dynamic or time-varying parameters, such as stock market price prediction and communication systems.

We first need to develop a general dynamic model to observe how these time-varying parameters are evolving over time as well as how the observations are generated from state parameters.

We start with a generic dynamical system model in discrete-time with time-steps of k.

$$x[k+1] = \boldsymbol{F}x[k] + \boldsymbol{G}u[k] + v[k] \tag{2.36}$$

with $v[k] \sim \mathcal{N}(0, \boldsymbol{Q})$ where $\boldsymbol{Q}$ is the discrete process noise covariance. The discrete-time observation equation is given by

$$y[k] = \boldsymbol{H}x[k] + w[k] \tag{2.37}$$

with $w[k] \sim \mathcal{N}(0, \boldsymbol{R})$.

Intuitively, the process noise represents our lack of knowledge about the system dynamics. The larger the process noise, the smaller will be our trust on the state equation. The measurement noise represents the imperfections in acquiring the data. The larger the measurement noise, the smaller will be our trust on the measurements.

In these types of dynamical systems, $x[k]$ is completely determined by the earlier state $x[l], l \leq k$ with the corresponding inputs. The state at time l gives the details on what happened prior to time l if we know $x[l]$. The estimation problem of state $x[l]$ of the system from measurements $y[0], \ldots, y[k]$ can be divided in three distinct problems:

- **Filtering:** Estimation of $x[l]$ from noisy measurements with $k = l$
- **Prediction:** Estimation of $x[l]$ from $[y_1, \ldots, y_k]$ with $l > k$
- **Smoothing:** Estimation of $x[l]$ from $[y_1, \ldots, y_k]$ with $l < k$

Traditionally, dynamical systems were handled by Kalman Filter, Extended Kalman

Variable	Meaning
$\boldsymbol{F}$	State transition matrix
$\hat{x}$	Estimated state
$\boldsymbol{G}$	Input matrix
u	Input
$\boldsymbol{H}$	Observation matrix
y	Measurements
$\boldsymbol{Q}$	Process noise covariance matrix
$\boldsymbol{K}$	Kalman gain matrix
$\boldsymbol{R}$	Measurement noise covariance matrix

Table 2.1: Notation for Kalman Filter.

Filter and Interacting Multiple Model. They are well-studied in literature. Hence, we briefly describe the main ideas. Further details can be found in [4], [53]–[56].

2.2.1 Kalman Filter

The Kalman filter (KF) [37] is a basic tool in analyzing stochastic linear systems and provides the minimum mean squared error solution to a linear system when the process under observation is completely represented by the state model. However, its success relies on the knowledge of the system model. When the models are unknown or partially known, it is hard to determine the covariances of the process and measurement noises [38]. This can typically happen during model switching which is governed by Markov chain based probability transition matrices, as demonstrated in Chapter 7

Kalman Filter starts with an initial mean and covariance equal to the true mean and covariance of the initial state distribution $\mu[0] = \hat{x}[0|-1]$ and $\Sigma[0] = \Sigma[0|-1]$, assuming a Gaussian distributed initial state $x[0] \sim \mathcal{N}(\mu[0], \Sigma[0])$. The recursive process starts with $k = 0$ and for each iteration, we have

$$\hat{x}[k|k-1] = \boldsymbol{F}[k]\hat{x}[k-1|k-1] \qquad \text{(Predicted State Estimate)} \tag{2.38}$$

$$\boldsymbol{\Sigma}[k|k-1] = \boldsymbol{F}[k]\boldsymbol{\Sigma}[k-1|k-1]\boldsymbol{F}^{\top}[k] + \boldsymbol{G}[k]\boldsymbol{Q}[k]\boldsymbol{G}^{\top}[k] \qquad \text{(Predicted Covariance)} \tag{2.39}$$

$$\boldsymbol{K}[k] = \boldsymbol{\Sigma}[k|k-1]\boldsymbol{H}^\top[k]\left(\underbrace{\boldsymbol{H}\boldsymbol{\Sigma}[k|k-1]\boldsymbol{H}^\top[k] + r[k]}_{\text{Residual Covariance}}\right)^{-1} \quad \text{(Kalman Gain)} \tag{2.40}$$

$$\hat{x}[k|k] = \hat{x}[k|k-1] + \boldsymbol{K}[k]\left(\underbrace{y[k] - \boldsymbol{H}\hat{x}[k|k-1]}_{\text{Residual/Innovation}}\right) \quad \text{(Estimated State)} \tag{2.41}$$

$$\boldsymbol{Q}[k] = \boldsymbol{I} - \boldsymbol{K}[k]\boldsymbol{H} \tag{2.42}$$

$$\boldsymbol{\Sigma}[k|k] = \boldsymbol{Q}[k]\boldsymbol{\Sigma}[k|k-1]\boldsymbol{Q}^\top[k] + \boldsymbol{K}[k]\boldsymbol{R}[k]\boldsymbol{K}^\top[k] \quad \text{(Est. Covariance)} \tag{2.43}$$

(2.38) represents the a priori state estimate (estimating the state before having seen the actual measurement) of $\hat{x}[k|k-1]$ at discrete-time k. $\boldsymbol{\Sigma}[k|k]$ and $\boldsymbol{\Sigma}[k|k-1]$ in (2.39) and (2.43) denote the one-step prediction error estimation and the next prediction error covariance matrices (ECM), also known as a priori estimate covariance. They also contain the innovation and innovation covariance terms. An innovation is simply the difference between the actual measurement and the a priori estimate, mapped into the measurement space. (2.40) is the Kalman Gain, which weights the innovation's contribution to the final state estimate. Kalman gain $\boldsymbol{K}$ is "proportional" to the covariance between the state prediction error and the innovation and is "inversely proportional" to the innovation covariance. Lastly, equations (2.41) and (2.43) are the a posteriori state estimate and a posteriori estimate covariance respectively.

Figure 2.2.1 depicts these steps sequentially. At each time step, the state vector $x[k]$ is propagated to the new state estimation $x[k+1]$ by multiplication with the constant state transition matrix $\boldsymbol{F}$. The state vector $x[k+1]$ is additionally influenced by the control input vector $u[k+1]$ multiplied by the input matrix $\boldsymbol{G}$, and the process noise vector of the system $v[k+1]$. The system state cannot be measured directly. The measurement vector $y[k]$ consists of the information contained within the state vector $y[k]$ multiplied by the measurement matrix $\boldsymbol{H}$, and the additional measurement noise $w[k]$.

Kalman Filter is versatile and can be used for for widely varying environments by

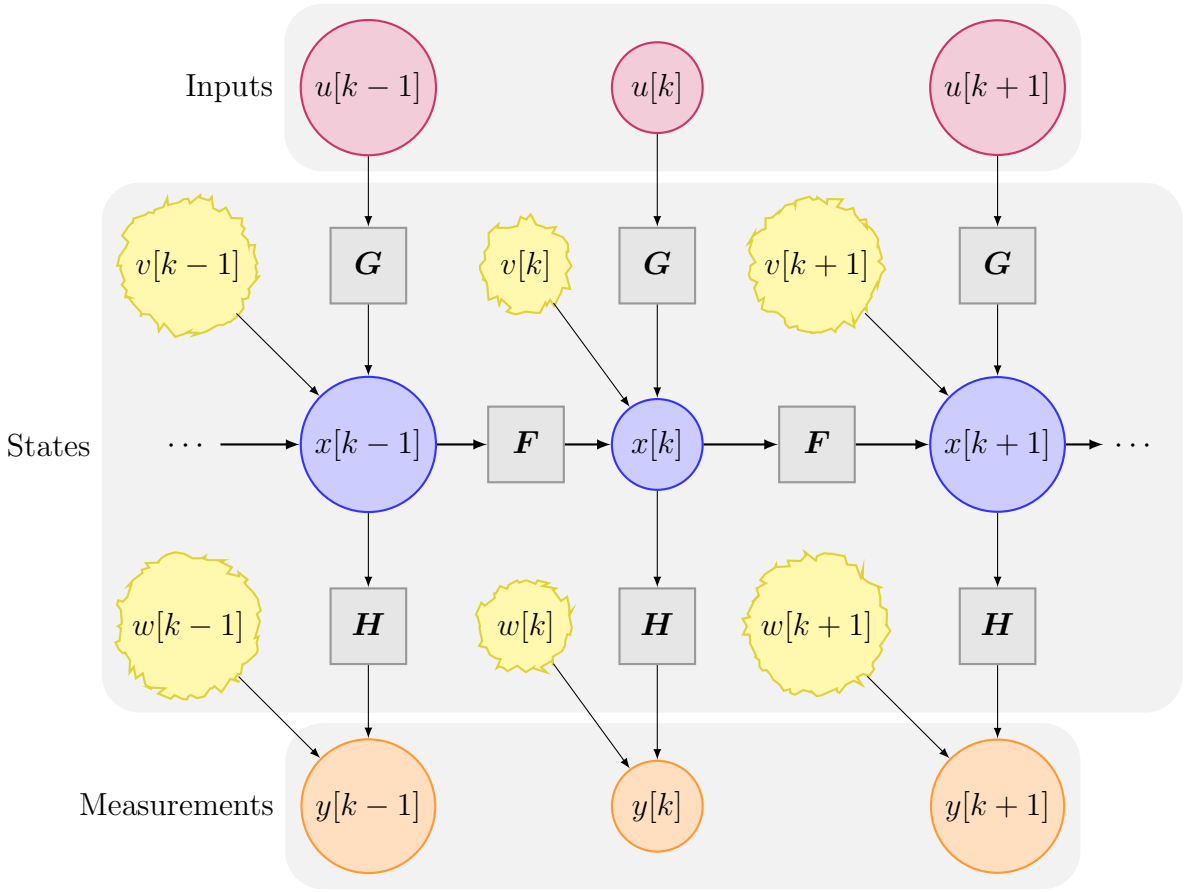

Figure 2.10: One iteration of Kalman Filter for a linear dynamical model.

changing a few parameters. It automatically handles missed detections and non-uniform sampling intervals. However, we need the state and measurement models to be linear, and the statistics for the measurement noise have to be accurately known and should follow a zero-mean Gaussian distribution. It is also computationally intensive compared to fixed-gain filters.

A major caveat for the KF is that in order to obtain an optimal state estimate, the KF requires exact knowledge of the system model as well as the process noise and measurement noise parameters. The performance of the KF degrades if there are mismatches between the true dynamical system and the assumed model; this topic has received significant attention in the literature and numerous approaches have been proposed to mitigate this degradation.

2.2.2 Least Squares

While least squares is a well-known technique, we briefly review it here because it as an example of a "data-driven" estimator that does not require knowledge of the underlying dynamical system model, though it does require training data containing a collection of known state variables and the corresponding observations. In the steady state, a Kalman filter predictor can be written in the form

$$\hat{x}[k+1|k] = a_0 y[k] + a_1 y[k-1] + a_2 y[k-2] + \dots \tag{2.44}$$

where, if the dynamics and noise parameters are all known, the $\{a_i\}$ coefficients can all be computed as functions of the steady state prediction and estimation covariances. These are all, consequently, functions of the steady state Kalman gain. This results in a stable IIR filter that resembles a Kalman filter, and it can be approximated by truncating the number of terms so that

$$\hat{x}[k+1|k] \approx a_0 y[k] + a_1 y[k-1] + \dots + a_L y[k-L] \tag{2.45}$$

where $L+1$ is the number of terms in the truncated sequence. Thus, the problem is to find $\{a_0, \ldots, a_L\}$ to minimize the mean squared prediction error without requiring knowledge of the dynamics or noise parameters. Given enough $y[k]$ observations, this can be accomplished using least squares on the linear system of equations

$$\underbrace{\begin{bmatrix} \hat{x}[k] \\ \vdots \\ \hat{x}[k-M] \end{bmatrix}}_{\triangleq \hat{X}} = \underbrace{\begin{bmatrix} y[k-1] & \ldots & y[k-L-1] \\ \vdots & \ddots & \vdots \\ y[k-M-1] & \ldots & y[k-L-M-1] \end{bmatrix}}_{\triangleq \boldsymbol{Y}} \underbrace{\begin{bmatrix} a_0 \\ \vdots \\ a_L \end{bmatrix}}_{\triangleq a} \tag{2.46}$$

which for $M \geq L$ can be solved as a standard least squares problem, i.e.,

$$a_{LS} = (\boldsymbol{Y}^\top \boldsymbol{Y})^{-1} \boldsymbol{Y}^\top \hat{X}. \tag{2.47}$$

2.2.3 Discrete-time Algebraic Riccati Equation

The Discrete-time Algebraic Riccati Equation (DARE) considers a Kalman filter at steady state and helps us better understand its sensitivities. We assume that we have a time-invariant system with $\boldsymbol{F}[n] \approx \boldsymbol{F}, \boldsymbol{G}[n] \approx \boldsymbol{G}, \boldsymbol{H}[n] \approx \boldsymbol{H}$ and the means of process and measurement noise covariance matrices $\boldsymbol{Q} = \boldsymbol{R} = \boldsymbol{0}$. Additionally, we assume that $\boldsymbol{F}, \boldsymbol{H}$ is completely observable and $\boldsymbol{F}, \boldsymbol{D}^\top$ is completely controllable, where $\boldsymbol{Q} = \boldsymbol{D}^\top \boldsymbol{D}$. Under these conditions, the covariance matrices and the Kalman gain will converge to a steady state. Now, the one-step-ahead (prediction) error covariance matrix $\boldsymbol{P}$ provides the asymptotic performance bounds using DARE. The steady-state prediction covariance matrix $\boldsymbol{P}$ is given by

$$\boldsymbol{P} = \boldsymbol{F}\left[\boldsymbol{P} - \boldsymbol{P}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}\boldsymbol{H}^T + \boldsymbol{R})^{-1}\boldsymbol{H}\boldsymbol{P}\right]\boldsymbol{F}^T + \boldsymbol{Q} \tag{2.48}$$

Using this, the steady-state estimation covariance $\boldsymbol{S}$ is calculated just after observation as

$$\boldsymbol{S} = \boldsymbol{P} - \boldsymbol{P}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}\boldsymbol{H}^T + \boldsymbol{R})^{-1}\boldsymbol{H}\boldsymbol{P} \tag{2.49}$$

The error covariance matrix can be calculated offline and updated independently of the rest of the model. DARE is typically used as a baseline in determining the optimality of tracking and prediction algorithms. We see its application in Chapters 5 and 7.

2.2.4 Interacting Multiple Model

Capturing the correct model dynamics with a single Kalman Filter can be problematic in situations where the state being estimated exhibits multiple dynamic modes. Unlike the prior estimators which do not give any special consideration to the switching nature of the model, the IMM filter [40] is a suboptimal estimator designed specifically for dynamical systems with model switching. The IMM falls into the class of estimators that use multiple filter models, typically with one matched to each of the N modes of the system. Other approaches in this class include, for example, the Generalized Pseudo-Bayesian (GPB) methods [39]. In such approaches, minimizing computation becomes very important due to the exponentially increasing number of state hypothesis. The IMM effectively combines hypotheses from multiple filter models in a computationally efficient manner, and is therefore widely used in practice for state estimation in dynamical systems with model switching.

The model inaccuracy is addressed by facilitating an interaction between them for different modes at the beginning of each filter cycle. These are weighed accordingly by the conditional probabilities of switching between model modes. We summarize the algorithm briefly here and direct the reader to [39] for a detailed treatment. Figure 2.11 presents the steps for a single iteration of the IMMKF.

The IMM estimates the blended states and covariances iteratively at each step by combining the initial conditions, states, and their associated covariances according to the

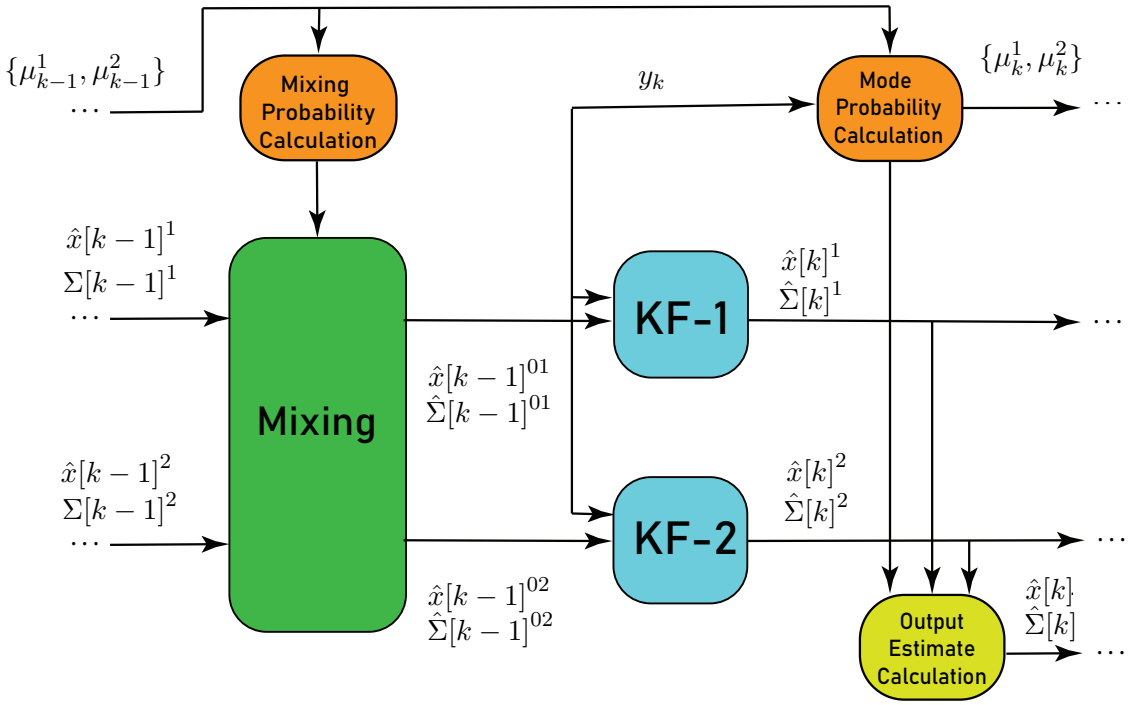

Figure 2.11: Block diagram of a single step of the IMM algorithm for two models.

mode transition probabilities. Denote the state estimate at time $k-1$ of the filter matched to the ith mode as $x^{(i)}[k-1|k-1]$ and its corresponding covariance $\Sigma^{(i)}[k-1|k-1]$ for $i \in 1, \ldots, N$, then each step of the IMM filter performs the following:

1. Calculate
   - mixing probabilities $\{\mu_{i|j}[k-1|k-1]\}_{i,j=1}^N$,
   - mixed estimates $\{\hat{x}^{(0i)}[k-1|k-1]\}_{i=1}^N$,
   - covariances $\{\Sigma^{(0i)}[k-1|k-1]\}_{i=1}^N$.
2. Using each of the N mode-matched models, calculate predicted estimates $\hat{x}^{(i)}[k|k-1]$ and covariances from mixed estimates in the previous step for i^{th} model, $i \in 1, \ldots, N$.
3. Calculate updated estimates $\hat{x}^{(i)}[k|k]$ and covariances from the predicted estimates for i^{th} model, $i \in 1, \ldots, N$ and calculate the updated mode probabilities $\mu_i[k]$.
4. Calculate the output state estimate and covariance estimates. The overall output

state estimate is computed as

$$\hat{x}[k|k] = \sum_{i=1}^{N} \hat{x}^{(i)}[k|k]\mu_i[k]. \tag{2.50}$$

While the traditional IMM as outlined above outputs state estimates, it can also be used to output state predictions. By using the assumption that the predicted mode probabilities at time $k+1$ given knowledge up through time k are equal to the estimated mode probabilities at time k, i.e., that $\mu_i[k+1|k] \approx \mu_i[k]$ as in [57], the overall predicted state can be computed via

$$\begin{aligned} \hat{x}[k+1|k] &= \sum_{i=1}^{N} \hat{x}^{(i)}[k+1|k]\mu_i[k+1|k] \\ &\approx \sum_{i=1}^{N} \hat{x}^{(i)}[k+1|k]\mu_i[k]. \end{aligned}$$

However, the IMM is a heuristic algorithm designed to yield a good performance only within the class of a fixed structure algorithms. It is not known to be generally optimal and does not guarantee robust performance. An IMM may give unsatisfactory results for particular scenarios such as nonlinear target dynamics during a turn and sudden starts and stops of maneuvers in model switching.

We finish the discussion on the ideas pertaining to Part II. These ideas will be used in Chapters 5, 6 and 7. Now, we divide this dissertation into two parts, discussing the results for communication systems in Part I and dynamical systems in Part II.

Part I

Machine Learning for Reliable Communication

Chapter 3

Joint Coding and Modulation in Additive White Gaussian Noise Channels

In this chapter, we consider the question of what can be gained by jointly designing channel coding and modulation schemes with a focus on the short blocklength regime with a small number of input bits per message. Here, we focus on linear block codes in Additive White Gaussian Noise (AWGN) channels. This restriction is motivated by their simplicity and performance. This regime is also of contemporary interest due to Internet of Things (IoT) devices often transmitting only infrequent and short messages with low latency requirements [58].

The focus in this chapter is on comparisons to BPSK-modulated Hamming codes where $(n, k) = (2^\ell - 1, 2^\ell - 1 - \ell)$ for $\ell = 3, 4, \ldots$ and $m = n$. For fair comparisons, the autoencoder uses parameters (n, k, m) identical to those of the conventional coding and modulation scheme and is subject to the same per-block total energy constraint as conventional coding and modulation.

3.1 Key Contributions

The contributions in this chapter are:

1. We provide steps to simplify the autoencoder model, improve the training speed and reduce parameters in autoencoders. This is detailed in Section 3.5.
    - The transmitter part of the autoencoder can be a simple lookup table for linear block codes. This is applicable to all channels with linear block codes.
    - For AWGN channels, the receiver part of the autoencoder can be replaced by either a matched filter or a posterior probability function when we know the model.
2. We discover that the training of autoencoders should be done at SNRs with sufficient numbers of block errors. Otherwise, the training will be slowed down. If we train at one such SNR, the autoencoder finds a code that will work at all SNRs. There is no need to have a family of codes for different SNRs. This is detailed in Section 3.3.
3. We find *improved* different sets of QAM constellations because training at different SNRs may lead to different codes and since, the objective function is multi-modal and we can converge to local maxima.
4. We find surprising results in AWGN channels since the minimum distance is not a good predictor of block error rate. The codes learned by autoencoders have worse minimum distance than classical codes with binary phase-shift-keying (BPSK) modulation, but perform better because most of the codewords have better distance properties. This is detailed in Section 3.6.5. We found codes with
    - *same* performance as Hamming+BPSK but with a different structure.
    - *better* performance than Hamming+BPSK with a structure with worse minimum distance properties.

Additionally, in Section 3.2, we discuss several aspects of an autoencoder-based communication system, followed by designing efficient parameterizations for linear block codes

in AWGN channels in Section 3.5. We provide details on training the autoencoder in Section 3.3. In Section 3.6, we look at results for Hamming $(2,4)$, $(7,4)$, $(15,11)$ and higher-order modulations for Golay codes. We finally discuss and summarize findings in Section 3.7.

3.2 System Model

We assume the point-to-point communication system model with $M = 2^k$ distinct messages and design the encoder and decoder similar to [8], [17], [59] as shown in Figure 3.2.

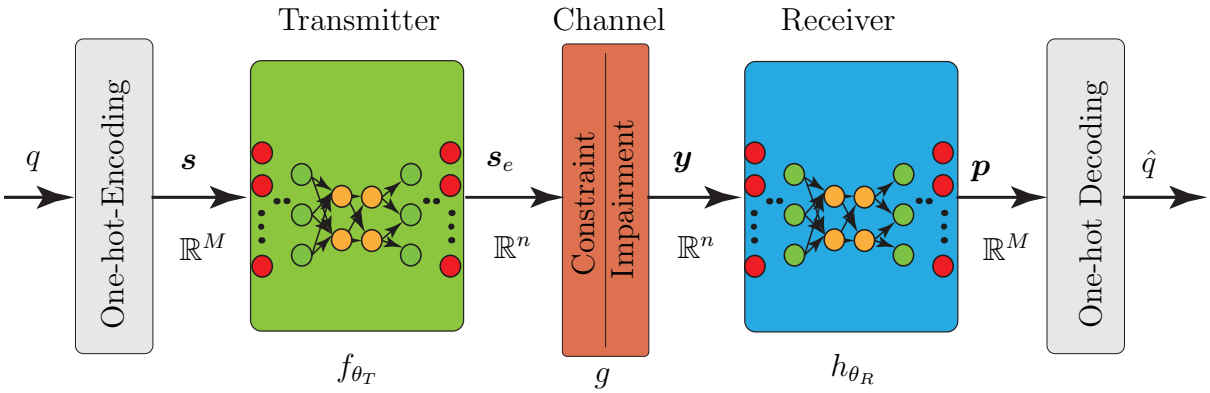

Figure 3.1: An autoencoder for an end-to-end communication system.

The classical approach at the transmitter is to provide a block of k bits at the input of the channel encoder, map these bits to an n-bit codeword, and then map this codeword to n real-valued symbols for transmission through the channel. Similarly, at the receiver, the noisy symbols are first demodulated and channel decoding is performed either on the soft demodulator outputs or on the hard decisions from the demodulator. The autoencoder considered here lumps the coding and modulation functions into $f_\theta : \{0,1\}^k \mapsto \mathbb{R}^n$, the memoryless channel function into $g : \mathbb{R}^n \mapsto \mathbb{R}^n$, and the demodulation and decoding functions into $h_\theta : \mathbb{R}^n \mapsto \{0,1\}^k$. The subscript θ indicates that these functions have parameters that we can adapt and learn to achieve a certain goal, e.g., minimizing the BLER.

We design the encoder and decoder similar to [8], [17] as shown in Figure 3.2. The transmitter seeks to communicate message $q \in \{1, \ldots, M\}$ to the receiver. The message is first mapped using a one-hot-encoding scheme to $\boldsymbol{s} = \mathbf{1}_q$ where $\mathbf{1}_q \in \mathbb{R}^M$ is a standard basis vector with q^{th} element equal to one and all other elements equal to zero. Let $\mathbb{Q} = \cup_{q=1}^{M} \mathbf{1}_q \subset \mathbb{R}^M$. The transmitter neural network then generates a channel input $\boldsymbol{s}_e = f_\theta(\boldsymbol{s})$ where $f_\theta : \mathbb{Q} \mapsto \mathbb{R}^n$ and where θ represents the weight vectors and biases. The channel input $\boldsymbol{s}_e$ is then sent through a mapping $\boldsymbol{y} = g(\boldsymbol{s}_e)$ with $g : \mathbb{R}^n \mapsto \mathbb{R}^n$ where a per-codeword energy constraint is imposed and an impairment (typically noise) is applied. The receiver then applies the transformation $h_\theta : \mathbb{R}^n \mapsto \mathbb{R}^M$ to compute a posterior probability vector $\boldsymbol{p} \in \mathbb{R}^M$ of all possible messages $q \in \{1, \ldots, M\}$ given $\boldsymbol{y}$. The decoded message $\hat{q}$ is simply the index of the maximum element of $\boldsymbol{p}$. The autoencoder is trained end-to-end to minimize the categorical cross-entropy loss function $\mathcal{L}$ between $\boldsymbol{s}$ and $\boldsymbol{p}$ with respect to θ.

We now describe each component in the block diagram.

3.2.1 One-hot Encoding

One hot encoding is a process by which categorical variables are converted into a form that could help a machine learning algorithm to correctly classify into categories. Table 3.1 describes an example of one-hot encoding for a given set of colors *red, yellow, green*. We can map the set in a way that there exists only one color in each row. Now, this row uniquely represents the color and can easily be mapped back to it. We apply the same

Table 3.1: An example demonstrating one-hot encoding.

Color	Red	Blue	Green
Red	1	0	0
Red	1	0	0
Blue	0	1	0
Green	0	0	1
Blue	0	1	0

concept on $M = 2^k$ messages.

While one-hot encoding works well with nominal data and eliminates any issue of higher categorical values influencing data, it can create very high dimensional encodings depending on the number of categorical features you have and the number of categories per feature. This can become problematic not only in smaller datasets but also potentially in larger datasets as well.

3.2.2 Transmitter

The transmitter lumps the encoder and modulator together and seeks to communicate a message $q \in \{1, \ldots, M\}$ to the receiver. The message is first mapped using a one-hot-encoding scheme to $\boldsymbol{s} = \mathbf{1}_q$ where $\mathbf{1}_q \in \mathbb{R}^M$ is a standard basis vector with q^{th} element equal to one and all other elements equal to zero. Let $\mathbb{Q} = \cup_{q=1}^{M} \mathbf{1}_q \subset \mathbb{R}^M$. The coding and modulation functions are into $f_{\theta_T} : \{0,1\}^k \mapsto \mathbb{R}^n$. The transmitter neural network then generates a channel input $\boldsymbol{s}_e = f_{\theta_T}(\boldsymbol{s})$ where $f_{\theta_T} : \mathbb{Q} \mapsto \mathbb{R}^n$ and where θ_T represents the weight vectors and biases. A constellation of m complex dimensions is learned by choosing the output of the encoder network and input of the decoder network to hold $n = 2m$ real dimensions.

3.2.3 Power Constraints

The channel input $\boldsymbol{s}_e$ is sent through a mapping $\boldsymbol{y} = g(\boldsymbol{s}_e)$ with $g : \mathbb{R}^n \mapsto \mathbb{R}^n$ where a constraint is imposed on the signal constellation. This is to ensure that the signals are not being mapped too far from each other. Without any constraints on $\boldsymbol{W}$, an autoencoder will learn to make the codewords very large and drive the BLER to zero for any typical additive noise distribution.

Hence, any reasonable formulation of an autoencoder for an additive noise channel needs to apply a constraint to the embedding matrix $\boldsymbol{W}$.

- Per-element energy constraint: $w_{i,j}^2 \leq \frac{\mathcal{E}}{nM}$ for all i, j.
- Per-codeword energy constraint: $\boldsymbol{w}_i \boldsymbol{w}_i^\top \leq \frac{\mathcal{E}}{M}$ for all i.

- Per-codebook energy constraint: $\sum_{i=1}^{M} \boldsymbol{w}_i \boldsymbol{w}_i^\top \leq \mathcal{E}$.

Assuming the same value of $\mathcal{E}$ in all three cases, note that each constraint is successively less restrictive. In other words, satisfying the per-element energy constraint automatically satisfies the per-codeword and per-codebook constraints. Satisfying the per-codeword energy constraint automatically satisfies the per-codebook energy constraint, but may not satisfy the per-element energy constraint.

Also note that the per-codebook energy constraint is equivalent to an average codeword energy constraint since

$$\sum_{i=1}^{M} \boldsymbol{w}_i \boldsymbol{w}_i^\top \leq \mathcal{E} \quad \Leftrightarrow \quad \underbrace{\frac{1}{M} \sum_{i=1}^{M} \boldsymbol{w}_i \boldsymbol{w}_i^\top}_{\text{average codeword energy}} \leq \frac{\mathcal{E}}{M} \tag{3.1}$$

Finally, note that the per-codebook energy is equal to the squared Frobenius norm, i.e.,

$$\sum_{i=1}^{M} \boldsymbol{w}_i \boldsymbol{w}_i^\top = \|\boldsymbol{W}\|_{\mathrm{F}}^2 \leq \mathcal{E} \tag{3.2}$$

hence, given any non-zero matrix $\boldsymbol{W}$, we can scale it to satisfy the per-codebook energy constraint by computing

$$\bar{\boldsymbol{W}} = \frac{\sqrt{\mathcal{E}}}{\|\boldsymbol{W}\|_{\mathrm{F}}} \boldsymbol{W}. \tag{3.3}$$

3.2.4 Block AWGN Channel

We define a memoryless channel function into $g : \mathbb{R}^n \mapsto \mathbb{R}^n$. Note that g subsumes both channel and constraint. The channel does not have any trainable parameters and just acts like a stochastic transformation of the input.

In this chapter, we consider a block AWGN channel which defines a mapping from $\mathbb{R}^n \to \mathbb{R}^n$

$$\boldsymbol{Y} = g(\boldsymbol{X}) = \boldsymbol{X} + \boldsymbol{Z} \tag{3.4}$$

where $\boldsymbol{X} \in \mathcal{X}^n$ and $\boldsymbol{Z} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I}_n)$ and n is the number of real symbols. $\boldsymbol{I}_n$ is an

identity matrix of dimension $n \times n$.

For binary phase-shift-keying (BPSK) modulated symbols, $\mathcal{X} = \{-\sqrt{\mathcal{E}_c}, +\sqrt{\mathcal{E}_c}\}$ where $\mathcal{E}_c = k\mathcal{E}_b/n$ is the energy per bit of the modulated coded signal and $\mathcal{E}_b$ is the energy per information bit. Such a channel can be compactly represented as AWGN($\mathcal{E}_b/N_0$) where $\mathcal{E}_b/N_0$ is the SNR of the AWGN channel.

The model in (3.4) has been studied in detail for decades and used as a benchmark to understand and model various practical communication scenarios. The channel capacity for AWGN channels is achieved by a codebook with Gaussian signaling. However, for practical implementations, such a codebook is not feasible because of its complexity and storage constraints.

The AWGN channel with BPSK inputs is used as a model for studying digital communication schemes as noise sources are additive and independent of each other and when added together, it can be approximated by a zero-mean Gaussian random variable according to Central Limit Theorem. More details on phase-shift-keying and other modulation schemes are given in Section 2.1.6.

3.2.5 Receiver

The receiver applies the transformation $h_{\theta_R} : \mathbb{R}^n \mapsto \mathbb{R}^M$ to compute a posterior probability vector $\boldsymbol{p} \in \mathbb{R}^M$ of all possible messages $q \in \{1, \ldots, M\}$ given $\boldsymbol{y}$. The demodulation and decoding functions into $h_{\theta_R} : \mathbb{R}^n \mapsto \{0, 1\}^k$. The decoded message $\hat{q}$ is simply the index of the maximum element of $\boldsymbol{p}$.

3.2.6 Intuition

We use the term *autoencoders* throughout this dissertation. An autoencoder refers to a neural network architecture which is trained to replicate its input to its output [60], [61]. This is done in two stages as shown in Figure 3.2.6: an encoder, which compresses its input to a lower dimension vector and a decoder that seeks to replicate the original input

from this lower dimensional *non-linear* manifold. Autoencoders are symmetrical in that the encoding layer is mimicked in the decoding layer as an inverted version of the encoding layer. They are typically used in non-linear dimensionality reduction, data-denoising and compression.

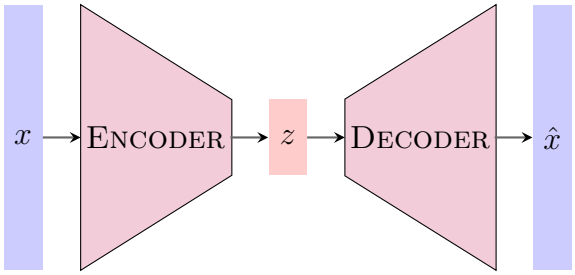

Figure 3.2: A simple schematic for an autoencoder.

An autoencoder is a composition of two parametric functions, an encoder f_{θ_T} and a decoder h_{θ_R}, with the aim of reproducing the input vector at the output. The parameter vector $\boldsymbol{\theta} = \{\theta_T, \theta_R\}$ holds all trainable variables. The subscript $\boldsymbol{\theta}$ indicates that these functions have parameters that we can adapt and learn to achieve a certain goal, e.g., minimizing the block error rate.

- The transmitter has to learn a meaningful representation of the input vector, which when provided to the decoder, holds enough information for replication of the input vector. To this end, the transmitter learns a higher order constellation where the symbols align themselves in an optimal constellation geometry.
- The receiver learns decision boundaries in between the impaired symbols performing like a maximum likelihood detector for the received signals.

The expectation is taken over each training batch. Since each message only holds a single non-zero value, the inner summation over M requires only one evaluation. The average of the cross-entropy over all samples is computed and through them an estimate of the gradient with respect to the parameters of the model.

3.3 Training the Autoencoder

We take the following factors into account when we train the autoencoder.

- **Batch Size**: Large training batch size leads to slower convergence but better final performance, and a small training batch size leads to faster convergence but worse final performance.
- **Training SNR**: In our tests, we observed that the adaptation rate of the autoencoder is highly dependent on the training SNR. It is important to set $\mathcal{E}_b/N_0$ such that the autoencoder observes frequent examples of correctly and incorrectly decoded blocks. This facilitates the training process. Training at high SNRs, e.g., $\mathcal{E}_b/N_0 = 8.5$ dB, does not provide enough examples of block errors and adaptation is slow in this setting. Similarly, training at a very low SNR like $\mathcal{E}_b/N_0 = -3$ dB limits the number of examples of correctly decoded blocks and results in slow adaptation.
- **Activation Function**: The network is trained using the Adam optimizer [62]

The best trade-off between convergence speed, computation time and performance is achieved by starting the training process with smaller training batch size and increasing it after initial convergence.

Following up from Section 3.2.3, there are two ways to normalize the codebook:

1. **Fixing the Noise Variance**: If we assume a setting where the *noise variance is fixed* and we want to vary the per-codebook norm constraint so that the system has the correct $\mathcal{E}_b/N_0$, this would be how to do it:
   (a) Given $\mathcal{E}_b/N_0$ in dB, first convert it to the actual ratio *not* in dB. We call this γ.
   (b) We can then assume $N_0 = 2$ so that the noise has unit variance in each dimension, which gives us $\mathcal{E}_b = 2\gamma$.
   (c) Compute $\mathcal{E}_s = k\mathcal{E}_b$. This is what we want the *average* symbol energy to be.
   (d) The total codebook energy is then just $\mathcal{E} = M\mathcal{E}_s$.

In other words, given γ and assuming $N_0 = 2$ so that we have unit variance AWGN in each dimension, the total codebook energy is then

$$\mathcal{E} = 2Mk\gamma \tag{3.5}$$

and hence the per-codebook energy constraint on the embedding matrix can be expressed as

$$\|\boldsymbol{W}\|_{\mathrm{F}}^2 \leq 2Mk\gamma. \tag{3.6}$$

This norm constraint should be applied after the embedding and the training should be done with unit variance noise in each dimension.

2. **Fixing the Norm Constraint**: If we assume a setting where the *norm constraint is fixed* (which implies a fixed value for $\mathcal{E}_b$) and we want to vary N_0 so that the system has the correct $\mathcal{E}_b/N_0$, this would be how to do it:
   (a) Given $\mathcal{E}_b/N_0$ in dB, first convert it to the actual ratio *not* in dB. We call this γ.
   (b) We are also given $\mathcal{E}$, the fixed per-codebook energy constraint. This is arbitrary, but we probably don't want to make it too small to avoid losing precision.
   (c) Compute $\mathcal{E}_s = \mathcal{E}/M$ (average energy per symbol).
   (d) Compute $\mathcal{E}_b = \mathcal{E}_s/k = \frac{\mathcal{E}}{Mk}$ (average energy per bit).
   (e) Solve for $N_0 = \mathcal{E}_b/\gamma = \frac{\mathcal{E}}{Mk\gamma}$.

   This *fixed* per-codebook energy constraint of $\mathcal{E}$ should be applied after the embedding by scaling the embedding matrix to have a Frobenius norm equal to $\mathcal{E}$. This scaling is always the same. The training should be done with noise drawn from $\mathcal{N}(0, \frac{N_0}{2}\boldsymbol{I})$, where N_0 is calculated as discussed above, based on the SNR in γ and the per-codebook norm constraint $\mathcal{E}$.

3.4 Higher-Order Modulations

Different order modulations allow us to send more bits per symbol and thus achieve higher throughputs and better spectral efficiencies. However, better SNRs are needed to overcome any interference and maintain a certain BLER. Table 3.2 lists several standard higher-order modulation schemes for which each 24-bit codeword can be mapped to an integer number of symbols.

Table 3.2: Functional mapping f_θ for different linear block codes.

Modulation	**Hamming(7,4)**	**Extended Golay(24,12)**
BPSK	$f : \{0,1\}^4 \mapsto \mathbb{R}^7$	$f : \{0,1\}^{12} \mapsto \mathbb{R}^{24}$
QPSK	—	$f : \{0,1\}^{12} \mapsto \mathbb{C}^{12}$
8-PSK	—	$f : \{0,1\}^{12} \mapsto \mathbb{C}^{8}$
16-QAM	—	$f : \{0,1\}^{12} \mapsto \mathbb{C}^{6}$
64-QAM	—	$f : \{0,1\}^{12} \mapsto \mathbb{C}^{4}$
256-QAM	—	$f : \{0,1\}^{12} \mapsto \mathbb{C}^{3}$

3.5 Efficient Parameterizations for Block Codes in AWGN channel

In this section, we look at three different ways of designing effective parameterizations for linear block codes in AWGN channels.

1. Embedding to replace Linear Block Coding
2. Matched Filter to replace the receiver.
3. Optimum receiver solvable by closed-form gradient descent.

If we assume a one-hot encoded message vector $\boldsymbol{x} \in \mathbb{R}^M$ and a "codebook" or constellation matrix $\boldsymbol{W} \in \mathbb{R}^{n \times M}$, the output of an AWGN channel can be written as

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{n} \tag{3.7}$$

where $\boldsymbol{n} \in \mathbb{R}^n$ is the additive white Gaussian noise.

3.5.1 Linear Block Coding and Modulation is Simply an Embedding

We can envision linear block coding as a lower dimensional embedding, mapping integers to vectors , i.e., essentially a lookup table that returns columns. The input symbols go through this layer followed by dense layers that turns positive integers to dense vectors of fixed size.

An embedding $\boldsymbol{W} \in \mathbb{R}^{M \times n}$ is a map from the set $\mathcal{S}$ containing $M = 2^k$ one-hot encoded vectors to a lower dimensional representation in $\mathbb{R}^n$ typically with $n \ll M$. The codewords of the block code are represented by columns of the embedding matrix $\boldsymbol{W}$:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1 & \dots & \boldsymbol{w}_M \end{bmatrix} = \begin{bmatrix} w_{1,1} & \dots & w_{1,M} \\ \vdots & & \\ w_{n,1} & \dots & w_{n,M} \end{bmatrix} \tag{3.8}$$

where the codeword $\boldsymbol{w}_i \in \mathbb{R}^{1 \times n}$. Now, $\boldsymbol{W}$ is trainable like the weight matrix of a dense neural network layer. This is a more efficient implementation of a dense layer with one hot encoded inputs. This is depicted in Figure 3.3.

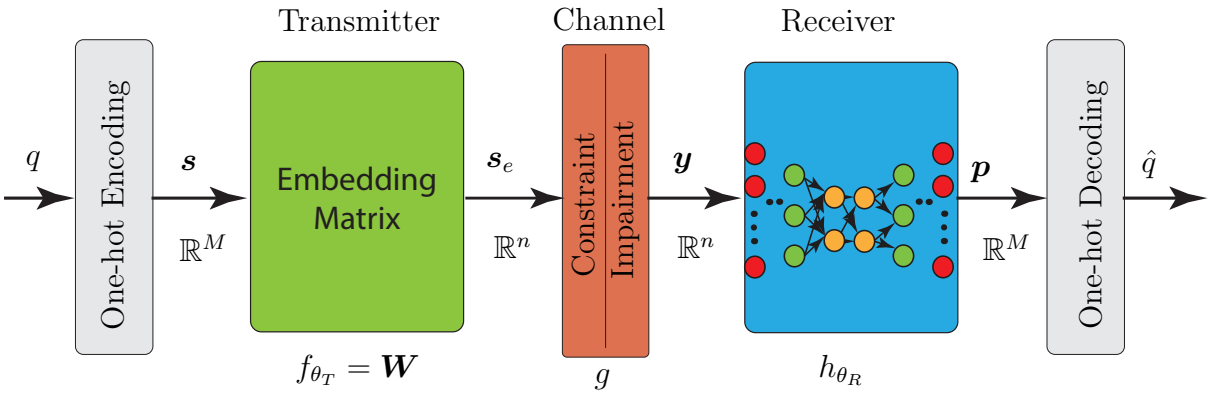

Figure 3.3: An autoencoder for an end-to-end communication system with transmitter replaced by an embedding.

This implementation reduces the number of parameters by 2x compared to few standard results [8], [63], [64]. The details are shown in Table 3.3. By making the transmitter

an embedding (or a linear layer with weights and no bias terms), we can have a simpler model with lesser number of parameters.

Table 3.3: All channels: Number of parameters with Embedding.

Layer(Output Dim.)	**(M,n)**	**(16,2)**	**(16,7)**	**(2048,15)**
Input (M)	0	0	0	0
Embedding (n)	Mn	32	112	30720
Constraint (n)	0	0	0	0
Channel (n)	0	0	0	0
Dense-ReLU (M)	$Mn + M$	48	128	32768
Dense-softmax (M)	$M^2 + M$	272	272	4196352
Total parameters	$M^2 + 2M(n+1)$	352	512	4259840
Total parameters [8]		626	791	8456207
Total parameters [63]		1170	1335	16848911
Total parameters [64]		33056	33146	$\approx 2^{22}$

3.5.2 Receiver as a Matched Filter

For an AWGN channel, we do not necessarily need to have an adaptive receiver. The receiver can copy the embedding matrix and perform matched filtering and pick the largest product. Now, the adaptation is only on the embedding matrix. Then, the receiver uses a transposed version of the embedding matrix and picks the maximum value. Figure 3.5.2 shows this modification with an connection from embedding layer to matched filter.

Table 3.4: AWGN channel: Number of parameters with an embedding and matched filter.

Layer(Output Dim.)	**(M,n)**	**(16,2)**	**(16,7)**	**(2048,15)**
Input (M)	0	0	0	0
Embedding (n)	Mn	32	112	30720
Constraint (n)	0	0	0	0
Channel (n)	0	0	0	0
Matched Filter (MF) + $\arg\max$ (M)	0	0	0	0
Total parameters	Mn	32	112	30720
Total parameters [8]		626	791	8456207
Total parameters [63]		1170	1335	16848911
Total parameters [64]		33056	33146	$\approx 2^{22}$

This implementation reduces the number of parameters many-fold compared to few

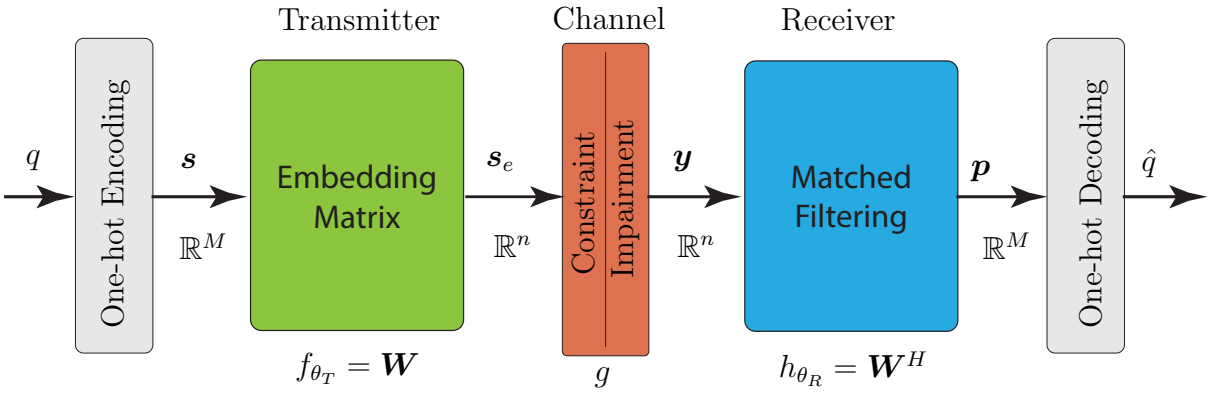

Figure 3.4: An autoencoder for an end-to-end communication system with transmitter replaced by an embedding, receiver replaced by a matched filter.

standard results [8], [63], [64]. The details are shown in Table 3.5.2. We now design an optimum receiver for AWGN channels.

3.6 Results for AWGN Channels

While the proposed autoencoder structure in Figure 3.5.2 is general and can be applied to any (n, k, m) linear block code for AWGN channels, we pick the values of n, k, m such that they can be compared to a few classical linear block codes such as Hamming or Golay codes.

3.6.1 (2,4,2) Autoencoder

Quadrature amplitude modulation (QAM) is widely employed to transmit more than one bit per modulation symbol as discussed in Section 2.1.6. The most common QAM signal constellations are QPSK and 16, 64 and 256-QAM, which carry $2, 4, 6$ and 8 bits per symbol, respectively. In this section, we design a 16-QAM constellation using autoencoders.

With conventional QAM, the signal space is partitioned into rectangular decision regions mainly because of the advantage that we can use binary data more effectively. However, it is well known that a two-dimensional regular tiling with hexagons is the most

efficient packing in terms of compactness [65]. H-QAM maximizes the minimum distance between signals in the constellation and thus minimizes the symbol error probability for a given average signal energy as well as the peak-to-average power ratio, which is important for OFDM systems. Similarly, there are other optimal constellations such as triangular, pentagonal and hexagonal QAM.

However, these techniques cannot fully utilize the gains possible with H-QAM only when the size of the signal constellation is small as it sometimes requires using a binary-input and ternary-output (BITO) code at the transmitter and reverse the procedure at the receiver [66]. Thus, current H-QAM solutions alone cannot provide sufficient performance improvement for wireless communication systems, although it has been adopted in optical systems. Similarly, there are limitations with other approaches as well.

In this section, we plan to impose a per-codebook energy constraint on the codewords and try to understand the solutions learned by the autoencoder. Since a standard 16-QAM follows a per-codebook energy constraint, we impose the same on the autoencoder.

Figure 3.5 illustrates the learned 16-point constellations with random intializations. Each random initialization gives a different constellation, each one non-rectangular near-optimally packed 16-QAM is achieved for (2,4) as shown in Figure 3.7.

We can observe in Figure 3.6.1 that there are atleast 3 optimal non-standard constellations which could be more optimal than a standard constellation. It takes a message index in $1, ..., 16$ and maps it into interesting constellations that look like H-QAM, θ-QAM [65].

This gives us hope that the autoencoder can find mappings that result in low BLER. The autoencoder is performing geometric shaping (GS) of the constellations here. While probabilistic shaping imposes a non-uniform distribution on a set of equidistant constellation points, GS employs a uniform distribution on non-equidistant constellation points [20].

It is also to be noted that the autoencoder converges to different constellations corresponding to different maxima of the multimodal objective function as shown in Table 3.5. This is interesting since only one of these constellations is actually optimal for AWGN.

The optimal *learned* constellation has better BLER than a standard QAM as shown in Figure 3.7.

Table 3.5: Pairwise Euclidean distance statistics for $(M, n) = (16, 2)$ from $\mathcal{E}_b/N_0 = 0$ to $\mathcal{E}_b/N_0 = 5.5$.

$\mathcal{E}_b/N_0$	**Min**	**Mean**	**Max**
0.0	0.497	1.905	3.583
0.5	0.481	1.902	3.733
1.0	0.510	1.907	3.554
1.5	0.648	1.905	3.718
2.0	0.683	1.910	3.515
2.5	0.755	1.911	3.525
3.0	0.776	1.912	3.525
3.5	0.806	1.913	3.488
4.0	0.803	1.914	3.514
4.5	0.833	1.914	3.494
5.0	0.846	1.915	3.480

Hence, our autoencoder can design non-standard QAMs. Following this, we can approach constellation design from a new angle with the following steps:

- Get a family of autoencoders for different random initialization.
- *Learn from* autoencoders about the optimal constellations and pick the best constellation.
- Design geometrical objects mathematically

3.6.2 (7,4,7) Autoencoder

Now, we take an example where we set $n = 7, k = 4$ and choose a BPSK modulation. We trained a fully connected autoencoder with following training parameters:

- Number of Examples: 10^6
- Number of Epochs: 100
- Training SNR: 3.0 dB
- Batch Size: 1000

and Figure 3.8 shows the corresponding training and validation accuracy curve. We can notice that the accuracy reaches a steady-state value around 0.97. It means that out of 100 blocks we send, 97 of them get correctly classified.

With the model defined as in Figure 3.5.2, we only need $16 \times 7 = 112$ parameters and the training takes about three seconds per epoch. It makes it possible to generate a linear block code for Hamming $(7, 4)$ within five minutes. However, due to the mathematical formulation of the elements in the autoencoder as described in Section 3.2, the codewords generated have different geometrical properties than a conventional Hamming $(7, 4)$ code.

Here, we apply a per-codeword energy constraint as discussed in Section 2.1.2. Table 3.6 compares the learned autoencoder codebook (in yellow) to a $(7, 4)$ Hamming code with BPSK modulation [67] (in blue), both with a total energy per encoded block (or per codeword) set to $\mathcal{E} = 7$. We can observe that the learned codewords have the same total per-codeword or per-block energy as conventional BPSK-modulated $(7, 4)$ Hamming codes, but the elements of each learned codeword are not constant modulus.

Table 3.6: BPSK-modulated Hamming $(7, 4)$ codewords and the learned $(7, 4, 7)$ autoencoder codewords, both with $\mathcal{E} = 7$.

BPSK+Hamming$(7, 4)$							Learned Codebook - Autoencoder$(7, 4)$						
-1	-1	-1	-1	-1	-1	-1	0.60	0.37	1.30	-0.68	-1.15	0.40	1.69
-1	-1	-1	1	-1	1	1	0.89	-0.32	-0.06	-1.66	-1.57	-0.82	-0.43
-1	-1	1	-1	1	1	1	-1.40	-0.58	-0.88	0.58	0.65	-0.54	-1.70
-1	-1	1	1	1	-1	-1	0.71	-0.93	-1.92	0.08	0.13	1.22	-0.64
-1	1	-1	-1	1	1	-1	-1.50	-0.44	1.44	-1.41	0.14	-0.61	-0.29
-1	1	-1	1	1	-1	1	-0.17	-0.99	0.01	-0.29	1.90	-1.16	0.98
-1	1	1	-1	-1	-1	1	-1.59	0.21	-0.99	-0.47	-0.68	-1.48	0.76
-1	1	1	1	-1	1	-1	-0.29	0.68	1.71	0.46	0.19	-1.00	-1.51
1	-1	-1	-1	1	-1	1	-0.66	0.40	-1.19	1.68	0.68	0.37	1.25
1	-1	-1	1	1	1	-1	0.10	1.41	-1.46	-0.67	-1.15	0.92	0.48
1	-1	1	-1	-1	1	-1	1.79	-0.60	0.92	1.10	0.72	0.37	0.84
1	-1	1	1	-1	-1	1	0.88	1.83	-0.05	0.33	0.57	-1.29	0.88
1	1	-1	-1	-1	1	1	0.68	-0.78	1.08	0.04	-0.91	1.39	-1.42
1	1	-1	1	-1	-1	-1	-0.64	-1.97	-0.54	-0.30	-0.81	0.68	1.10
1	1	1	-1	1	-1	-1	-0.97	-0.79	0.72	0.28	1.27	1.79	0.11
1	1	1	1	1	1	1	-1.79	0.23	0.41	0.87	-1.53	0.69	-0.03

Table 3.7 shows the pairwise Euclidean distance statistics of conventional Hamming BPSK-modulated codewords and of the autoencoder learned codewords in Table 3.6. We

can observe that, although the codewords learned by the autoencoder do not exhibit the structure of the BPSK-modulated Hamming $(7, 4)$ code, the distance statistics of the autoencoder are essentially identical to those of the Hamming code. Also note that the learned codewords are not unique, i.e., retraining will result in different learned codewords. Nevertheless, in each test, the distance statistics after training were always consistent with those in Table 3.7.

Table 3.7: Pairwise Euclidean distance statistics for BPSK-modulated Hamming $(7, 4)$ and $(7, 4, 7)$ autoencoders with $\mathcal{E} = 7$.

Scheme	**Min**	**Mean**	**Max**
Hamming $(7, 4)$ with BPSK	3.464	3.836	5.292
$(7, 4, 7)$ Autoencoder	3.429	3.836	5.289

We can see from Figure 3.9 that the codebook learned by the autoencoder has similar distance statistics to the BPSK-modulated Hamming $(7, 4)$ code and it outperforms a BPSK-modulated Hamming $(7, 4)$ code with hard decision decoding (HDD) and is similar to a BPSK-modulated Hamming $(7, 4)$ code with soft decision decoding (SDD).

Figure 3.10 plots the achieved BLER of the $(7, 4, 7)$ autoencoder in AWGN channels. The BLER of BPSK-modulated Hamming $(7, 4)$ codes and several finite-blocklength bounds (theoretical RCU [47] and metaconverse [48]) along with the normal approximation [68] are also plotted for comparison. More details about these are presented in Section 2.1.5. In this case, the BLER performance of the autoencoder is essentially identical to a BPSK-modulated Hamming $(7, 4)$ code with soft decision or maximum likelihood decoding.

3.6.3 Conditional BLER for a (7,4,7) Autoencoder

In this section, we delve deeper to understand the geometry of the codewords generated by autoencoder. We take each message independently and send each of the 16 codewords 10^7 times to the autoencoder and the soft decision decoder at $\mathcal{E}_b/N_0 = 3.0$ dB to understand how each noisy symbol gets classified as the right codeword. Thereby, we populate

the confusion matrix between the sent and the received codewords to understand the occurrence of misclassifications for a given codeword. Figure 3.11 shows the pairwise distances between the sent and received codewords for the autoencoder and the Hamming code.

Figure 3.12 shows the *sorted* pairwise distances between the sent and received codewords. We can see that the distances for SDD are closer to its overall BLER as compared to autoencoder which has more variation. Since we tested only with 10^7 samples, the red dots are slightly shifted away from the HBLER line. They would reach HBLER asymptotically.

To understand the progression of block errors, we take a codeword, 9 here, and consider the cumulative sum of the elements in the row that reaches the minimum pairwise Euclidean distance with $(7, 4, 7)$ autoencoder. In this case, code word 9 achieves this minimum. We can see from figure 3.13 that the distance statistics for this word are similar for the autoencoder and the soft-decision-decoder.

This way of analyzing conditional BLER gives us insights into why an autoencoder performs as well as a classical coding and modulation scheme.

3.6.4 (15,11,15) Autoencoder

In this subsection, we consider a higher order constellation with 2048 messages getting mapped into 15 symbols. We continue to consider BPSK modulation here, so $n = m = 15$.

Similar to Section 3.6.2, we trained a fully connected autoencoder with following training parameters:

- Number of Examples: 10^5
- Number of Epochs: 100
- Training SNR: 3.0 dB
- Batch Size: 1000

and Figure 3.8 shows the corresponding training and validation accuracy curve. We can

notice that the accuracy reaches a steady-state value around 0.57. It means that out of 100 blocks we send, 57 of them get correctly classified.

We use the model defined as in Figure 3.5.2 and apply a per-codeword energy constraint as discussed in Section 2.1.2. Then, we would only need $15 \times 11 = 165$ parameters and the training takes about five seconds per epoch. As we have seen in the $(7, 4, 7)$ Autoencoder case, the codewords generated by $(15, 11, 15)$ Autoencoder has different geometrical properties than a conventional Hamming $(15, 11)$ code.

Since it is hard to reproduce all 2048 codewords as in Table 3.6, we consider pairwise Euclidean distance statistics. It is to be noted that the learned codewords have the same total per-codeword or per-block energy as conventional BPSK-modulated $(15, 11)$ Hamming codes, but the elements of each learned codeword are not constant modulus. Nevertheless, in each test, the distance statistics after training were always consistent with those in Table 3.8.

Table 3.8: Pairwise Euclidean distance statistics for BPSK-modulated Hamming $(15, 11)$ and $(15, 11, 15)$ autoencoders with $\mathcal{E} = 15$.

Scheme	**Min**	**Mean**	**Max**
Hamming $(15, 11)$ with BPSK	3.464	5.430	7.746
$(15, 11, 15)$ Autoencoder	3.277	5.431	7.609

We observed that the autoencoder quickly converged to codewords with mean distance identical to BPSK-modulated $(15, 11)$ Hamming codes, whereas the minimum distance was much slower to converge. The results shown in Table 3.8 were achieved after training on 2×10^7 examples for 150 epochs (other training parameters same as previous example).

We can see from Figure 3.15 that the codebook learned by the autoencoder has similar distance statistics to the BPSK-modulated Hamming $(7, 4)$ code and it outperforms a BPSK-modulated Hamming $(7, 4)$ code with hard decision decoding (HDD) and is similar to a BPSK-modulated Hamming $(7, 4)$ code with soft decision decoding (SDD).

Figure 3.16 plots the achieved BLER of the $(15, 11, 15)$ autoencoder in AWGN channels along with the BLER of BPSK-modulated Hamming $(15, 11)$ code and the finite-

blocklength bounds and approximations. Somewhat surprisingly, in light of the autoencoder's worse minimum distance statistic in Table 3.8, the achieved BLER of the $(15, 11, 15)$ autoencoder is approximately 0.5 dB better than that of the conventional BPSK-modulated Hamming $(15, 11)$ code with soft decision (maximum likelihood) decoding.

3.6.5 Conditional BLER for a (15,11,15) Autoencoder

Following a similar approach described in Section 3.6.3, we send each of the 2048 codewords 5×10^5 times to the autoencoder and the soft decision decoder at $\mathcal{E}_b/N_0 = 3$ dB to understand how each noisy symbol gets classified as the right codeword. Then, we populate the confusion matrix between the sent and the received codewords to understand the occurrence of misclassifications for a given codeword.

Figure 3.17 shows the pairwise distances between the sent and received codewords for the autoencoder and the Hamming code and Figure 3.18 shows the sorted pairwise distances between the sent and received codewords. We can see that, similar to the $(7, 4)$ case, the pairwise distances for SDD are closer to its overall BLER as compared to autoencoder which has more variation.

We consider the cumulative sum of the elements in the row that reaches the minimum pairwise Euclidean distance with $(15, 11, 15)$ autoencoder. In this case, code word 849 achieves this minimum 3.277. We can see from Figure 3.19 that the individual distance statistics for this word for the autoencoder are much lower than that of the soft-decision-decoder.

We observe that the distance mapping for Hamming $(15, 11)$ is sparser than that of the autoencoder. The most interesting part occurs with the first few distances as seen in Figure 3.6.5, the first four distances for the SDD are smaller than the autoencoder, which resulted in the autoencoder having a lower minimum pairwise distance statistic. This is shown in Table 3.9. Additional inspection of the *conditional* BLER for each

BPSK+Hamming(15,11)	**Autoencoder(15,11,15)**
588	788
586	705
573	645
571	626
565	566
564	562
563	556
560	526
555	465
553	458
545	432
545	431
544	406
544	403
543	380

Table 3.9: First 15 elements of row 849 of confusion matrix (sorted) for a $(15, 11, 15)$ autoencoder.

learned codeword shows that the autoencoder learned an *asymmetric* code in the sense that certain codewords had worse conditional BLER and other codewords had better conditional BLER than the unconditional BLER. This is in contrast to Hamming codes with BPSK modulation where each codeword has a conditional BLER matching the unconditional BLER due to symmetry. By finding more codewords with good conditional BLER , the autoencoder can outperform Hamming codes with BPSK modulation in terms of unconditional BLER. This could possibly be the reason why the autoencoder approach beats the soft decision decoder method by 0.5 dB.

This comes at the cost of a small number of codewords with worse conditional BLER. As we can see cumulative distributive function from Figure 3.6.5, the autoencoder has worse codeword distances for 10 codewords and then the soft-decision-decoding curve takes over.

Additionally, we can observe that for Hamming $(7, 4)$, the bounds are loose and the curves for HDD and SDD and the autoencoder fall between the bounds. However, for a longer blocklength code like Hamming $(15, 11)$, the bounds become tighter. Specifically, we can observe that the HDD curve falls beyond the RCU bound and normal approxi-

mation, and the SDD and the autoencoder curves align closely to the RCU bound and normal approximation.

3.6.6 Higher-order Modulations with Extended Golay codes

To consider joint modulation and coding with higher-order modulation schemes, this section considers the extended Golay $(24, 12)$ code [69]. The basic approach for the autoencoder in this case is essentially identical to the approach discussed in Section 3.2, except the autoencoder is designed to produce pairs of real-valued channel inputs corresponding to the complex symbols of a modulated signal.

For example, when designing an autoencoder to compare against 16-QAM-modulated extended Golay$(24, 12)$, the autoencoder f_θ function is designed to produce an output, i.e., a channel input, of dimension $\mathbb{R}^{12}$, which contains I and Q symbols of dimension $\mathbb{R}^6$ each.

As discussed in Section 2.1.2, the autoencoder was trained to meet a per-codebook energy constraint, consistent with conventionally coded and modulated system at each modulation order. It is trained separately for each modulation order at 3.0 dB $\mathcal{E}_b/N_0$ and is tested for $\mathcal{E}_b/N_0$ varying from 0 dB to 10 dB.

Figure 3.21 presents BLER results for the trained autoencoder-based joint modulation and coding scheme against the Golay $(24, 12)$ code with several different modulation orders with both hard decision and soft decision decoding. As seen in the BPSK-modulated Hamming $(7, 4)$ results, the autoencoder outperforms hard decision decoding and achieves performance close to soft decision decoding. In fact, the autoencoder appears to slightly outperform soft decision decoding with 8-PSK, 16-QAM, and 64-QAM modulation with an extended Golay $(24, 12)$ code. These results show strong generalization as the training was performed at a single $\mathcal{E}_b/N_0$ while the testing was performed over a wide range of $\mathcal{E}_b/N_0$s. These simulations were conducted with 12×10^5 examples in the training and test datasets over 100 epochs. With more data and longer training, it is possible to see

further improvement in the autoencoder performance.

Table 3.10 provides some statistics on the pairwise Euclidean distances between the message vectors for BPSK, QPSK, 8-PSK modulated Golay $(24, 12)$ codes and the corresponding autoencoders. As observed in Table 3.10, the autoencoder statistics are similar to those of the Golay $(24, 12)$ codes. Similar to Hamming codes, the Golay codes produce constant-modulus codewords upto 8-PSK modulation whereas the autoencoder only satisfies a per-codeword energy constraint and is not constrained to learn constant modulus codewords.

Table 3.10: Pairwise Euclidean distance statistics for Golay $(24, 12)$ codes and the corresponding autoencoders.

Modulation	**Scheme**	**Min**	**Mean**	**Max**
BPSK	Golay $(24, 12)$	5.6569	6.8919	9.7980
	Autoencoder $(24\text{R}, 12)$	4.0177	6.5488	9.1863
QPSK	Golay $(24, 12)$	4.0000	4.8733	6.9282
	Autoencoder $(12\text{C}, 12)$	2.8962	4.6265	6.4538
8-PSK	Golay $(24, 12)$	2.1648	3.9675	5.4458
	Autoencoder $(8\text{C}, 12)$	2.0918	3.8160	5.5537
16-QAM	Golay $(24, 12)$	1.7889	3.4151	5.6569
	Autoencoder $(6\text{C}, 12)$	1.4595	3.2637	5.0779
64-QAM	Golay $(24, 12)$	0.8729	2.7647	5.4336
	Autoencoder $(4\text{C}, 12)$	0.7960	2.7362	4.4232

3.7 Conclusion

We propose a new data-driven approach to wireless system design using machine learning. Within this work, we demonstrate that end-to-end deep neural network based learning can be adapted effectively for physical layer radio systems to achieve state of the art levels of performance in numerous scenarios.

First, we discussed the background and fundamental tools used. Then, we considered AWGN channels and formulated efficient parameterizations for linear block codes. Then,

we consider three cases in Hamming codes: $(2, 4), (7, 4), (15, 11)$ and show the nuances in how they work and present an in-depth analysis of why they work well.

There is a lot of scope for improvement especially in the following directions:

1. We can consider channels with memory, e.g., Markov-Gaussian models
2. We can reduce training workload with domain adaptation and transfer learning
3. There are a lot of interesting directions in leveraging autoencoders to perform on
    - structural interference/jamming channels
    - Rayleigh block fading channels
    - deletion channels
    - real channel data obtained from USRPs

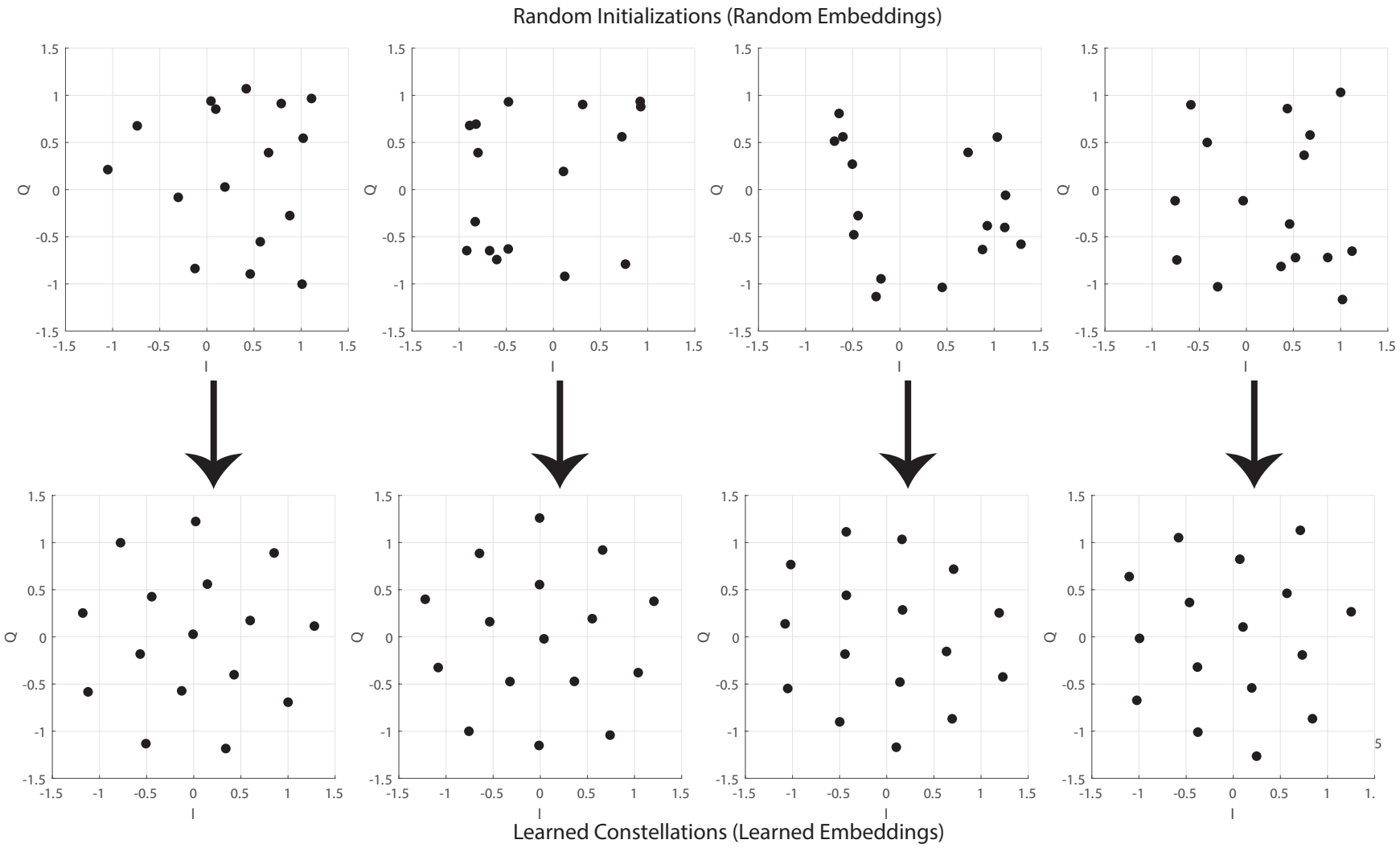

Figure 3.5: Constellations for $(M, n) = (16, 2)$ generated by an autoencoder. Every random initialization in the top row gives correspondingly a different constellation in the bottom row.

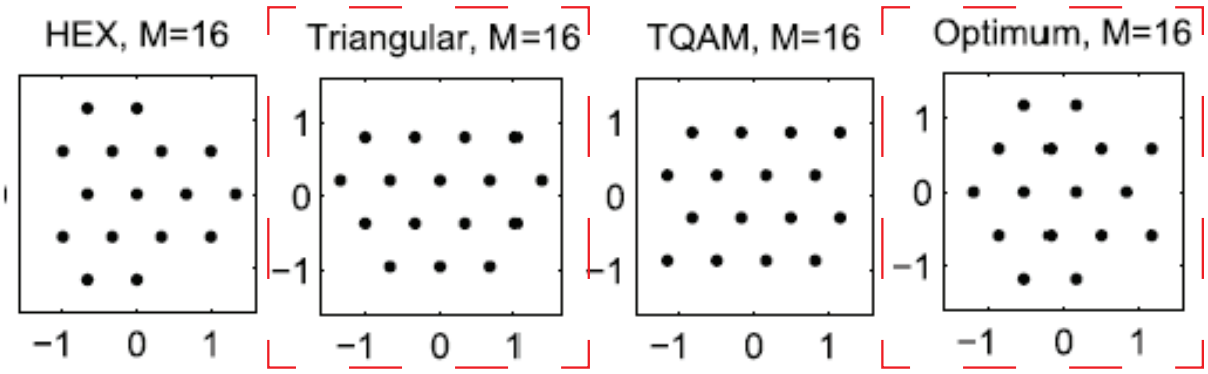

Figure 3.6: Comparison with non-standard QAM constellations [65]. We see that the constellations in 3.5 resemble the triangular and optimum constellations here.

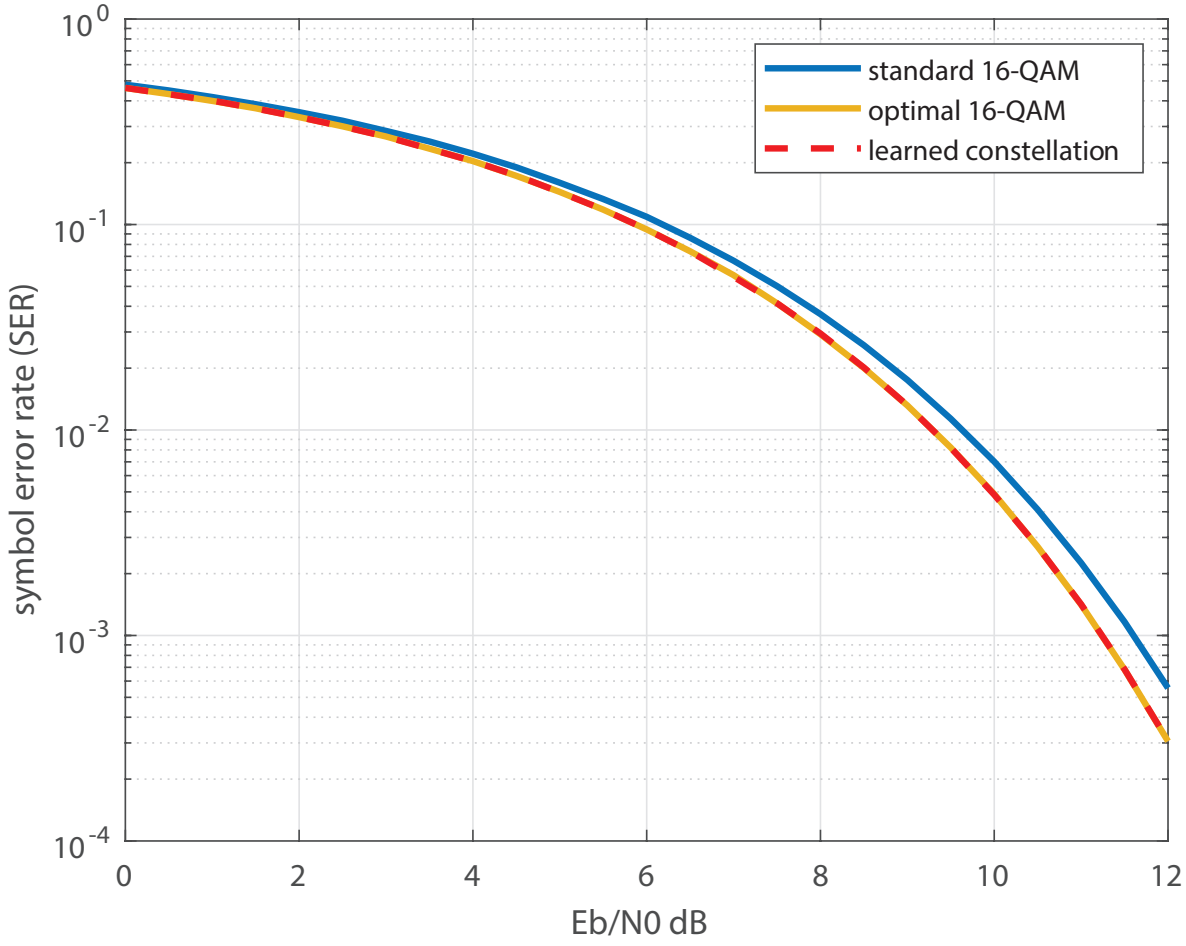

Figure 3.7: Symbol error rate for $(M, n) = (16, 2)$ from $\mathcal{E}_b/N_0 = 0$ to $\mathcal{E}_b/N_0 = 12$.

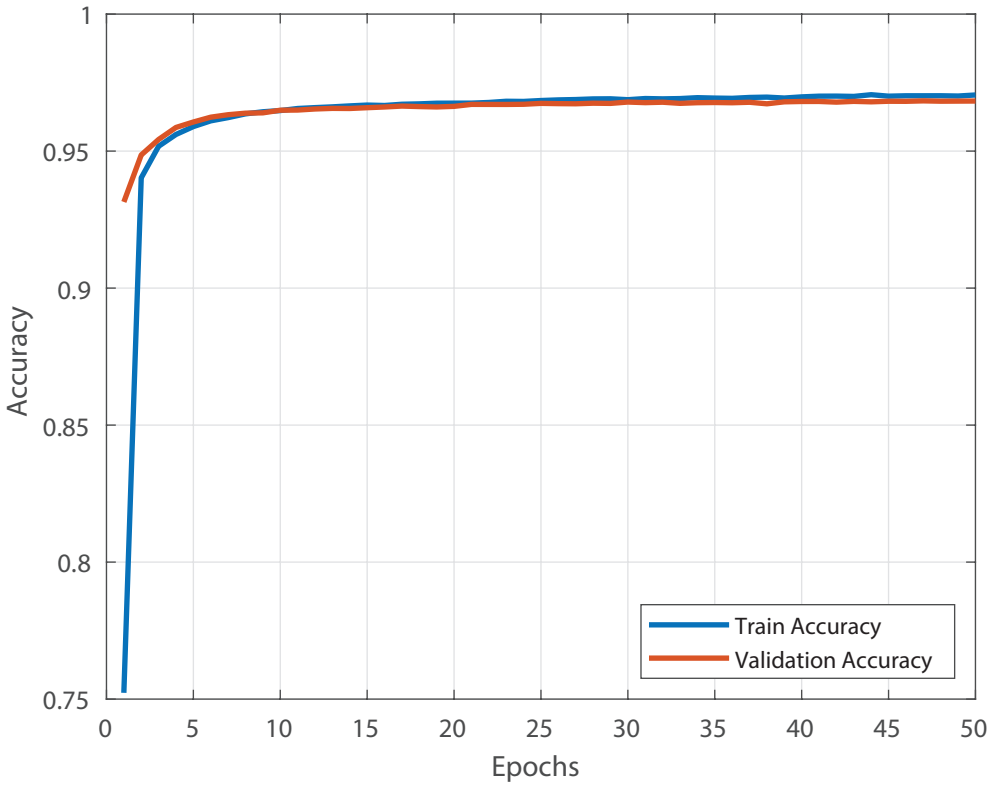

Figure 3.8: Training and validation accuracy curve for $(7, 4, 7)$ autoencoder.

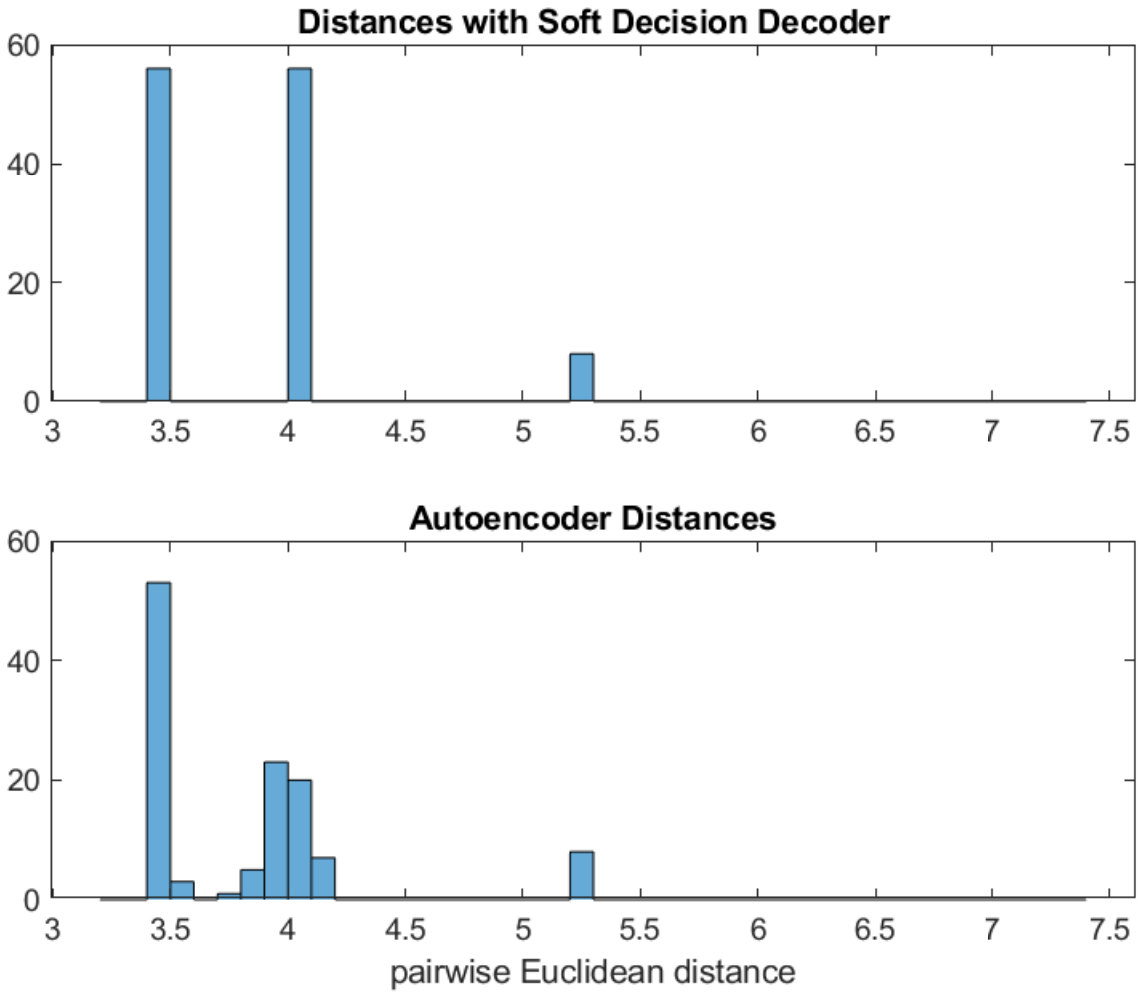

Figure 3.9: Histogram of pairwise Euclidean distances for Hamming $(7, 4)$ BPSK-modulated codewords and autoencoder $(7, 4, 7)$ learned codewords.

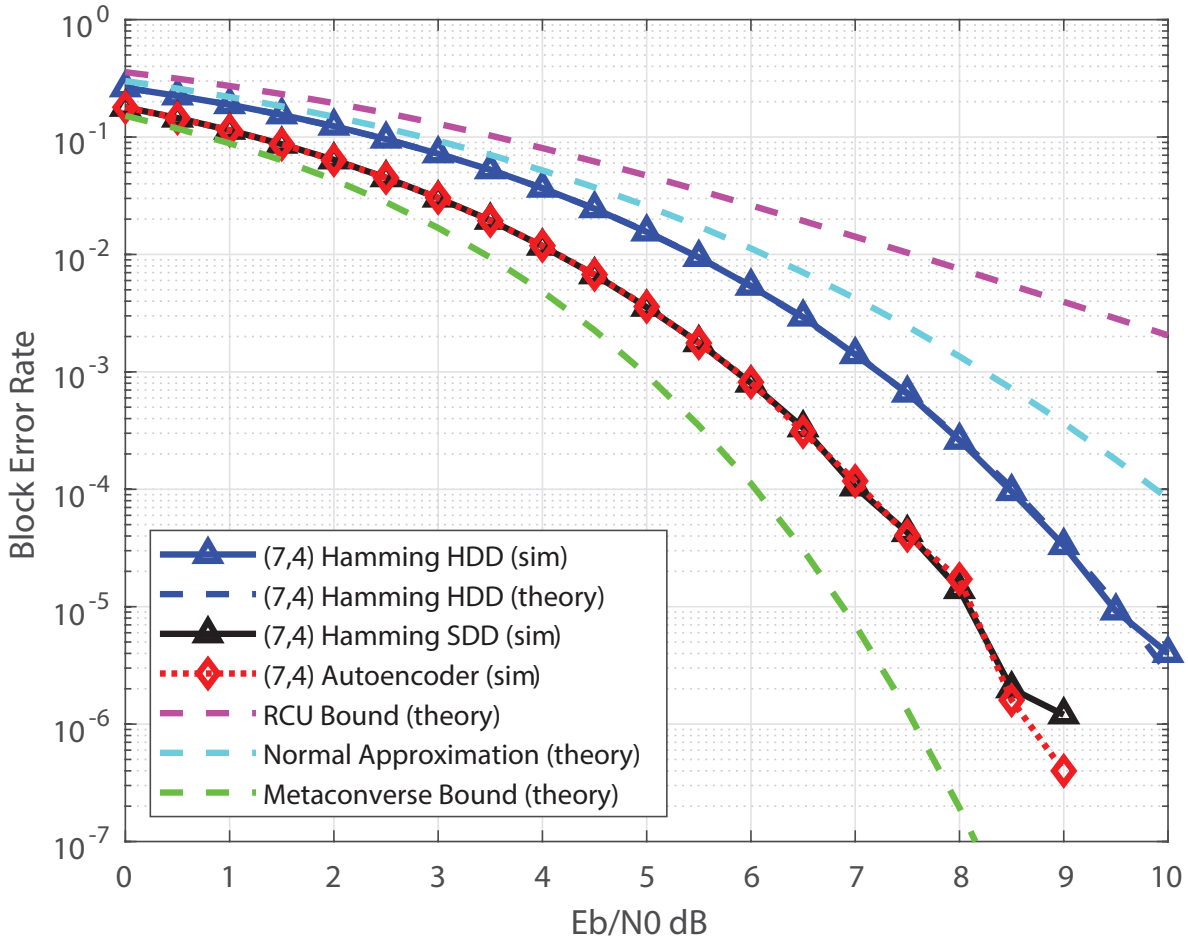

Figure 3.10: BLER of the trained $(7, 4, 7)$ autoencoder and BPSK-modulated Hamming $(7, 4)$ in an AWGN$(\mathcal{E}_b/N_0)$ channel. Random coding union (RCU), metaconverse, and normal approximation bounds are also plotted.

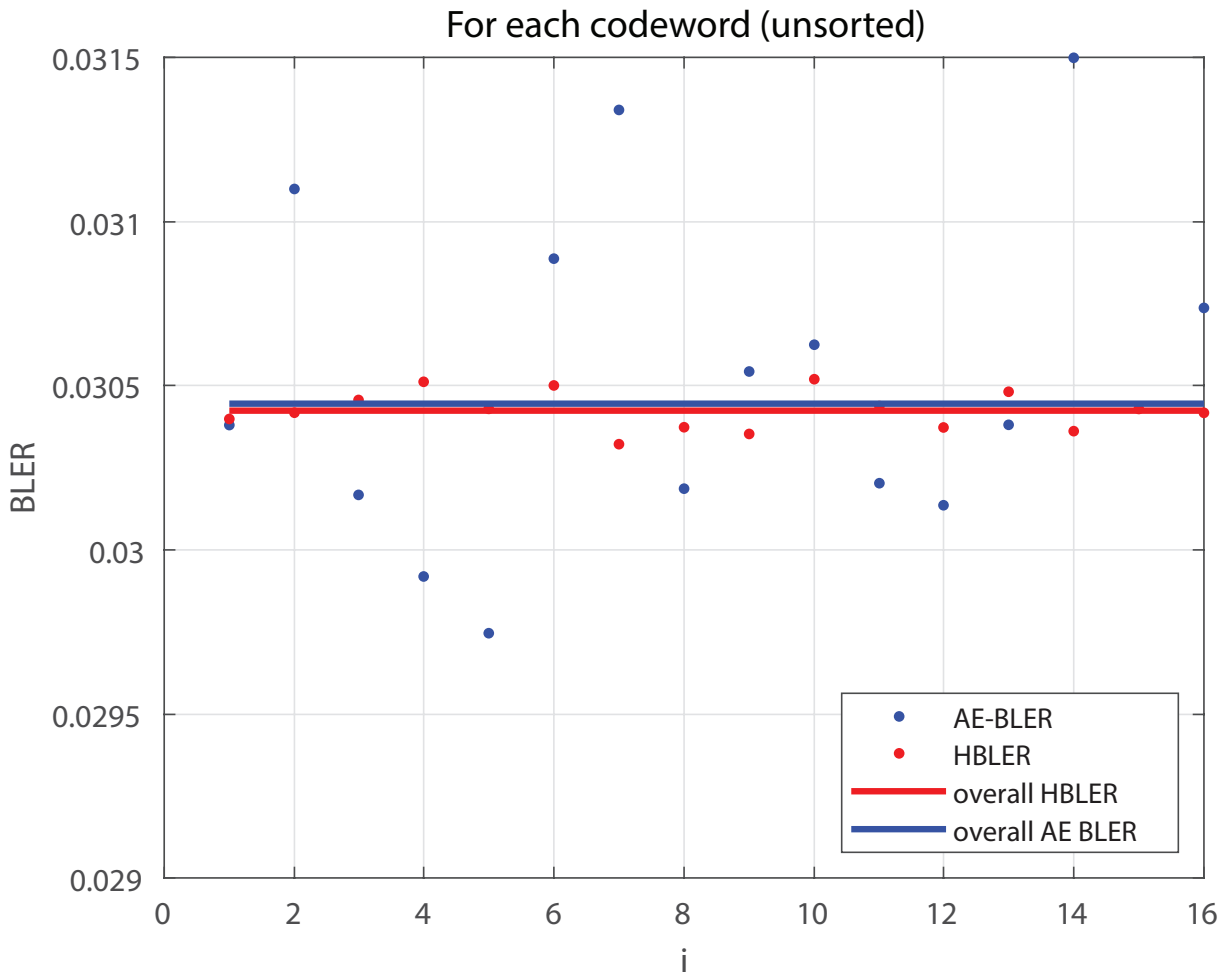

Figure 3.11: BLER for each codeword of the trained $(7, 4, 7)$ autoencoder and BPSK-modulated Hamming $(7, 4)$ in an AWGN$(\mathcal{E}_b/N_0)$ channel.

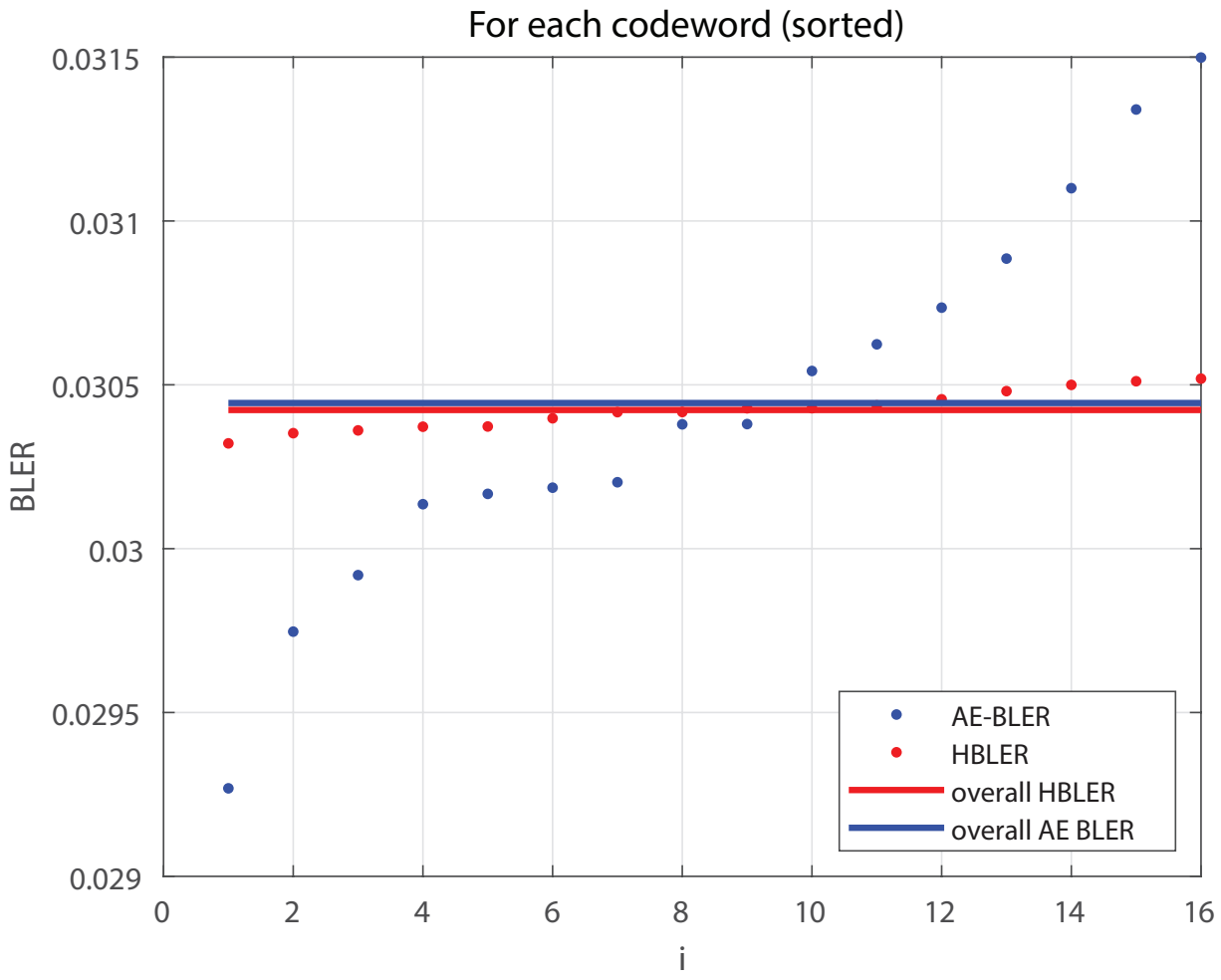

Figure 3.12: BLER for each codeword (sorted) of the trained $(7, 4, 7)$ autoencoder and BPSK-modulated Hamming $(7, 4)$ in an AWGN($\mathcal{E}_b/N_0$) channel.

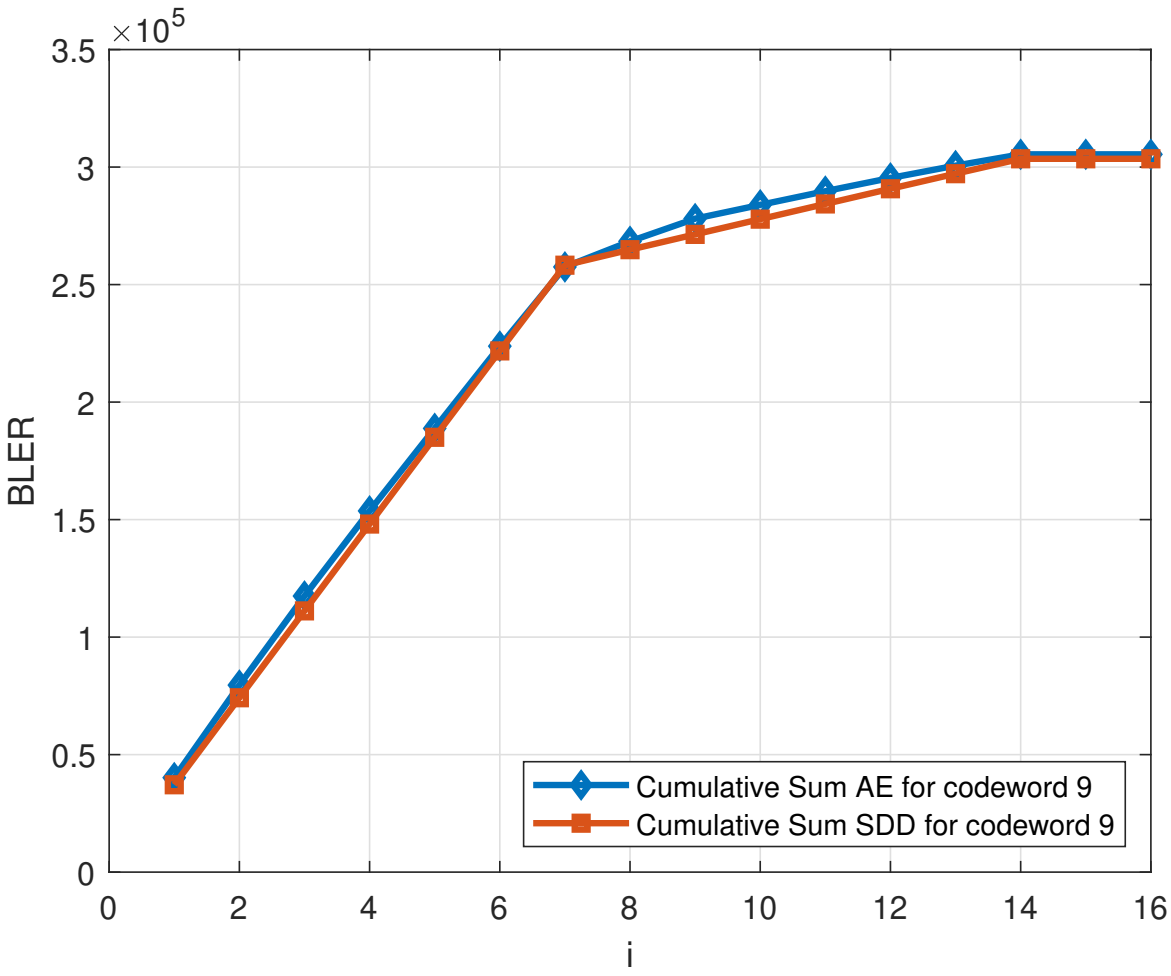

Figure 3.13: Cumulative sum for codeword 9 with the trained $(7, 4, 7)$ autoencoder and BPSK-modulated Hamming $(7, 4)$ in an AWGN($\mathcal{E}_b/N_0$) channel.

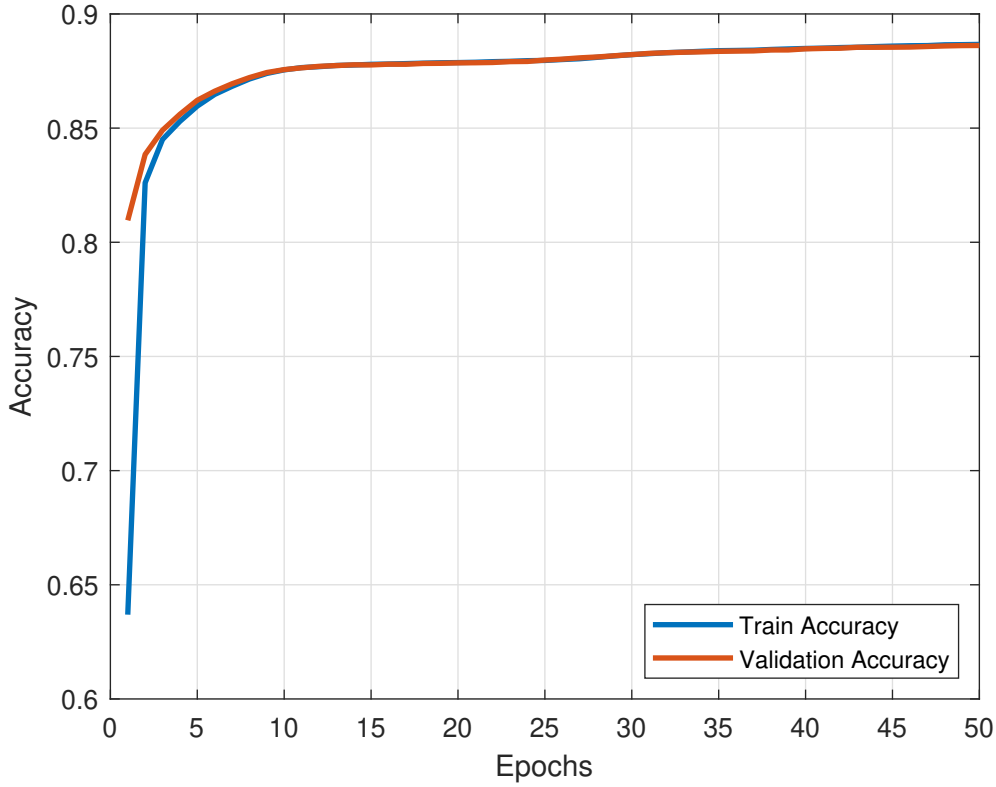

Figure 3.14: Training and validation accuracy curve for $(15, 11, 15)$ autoencoder.

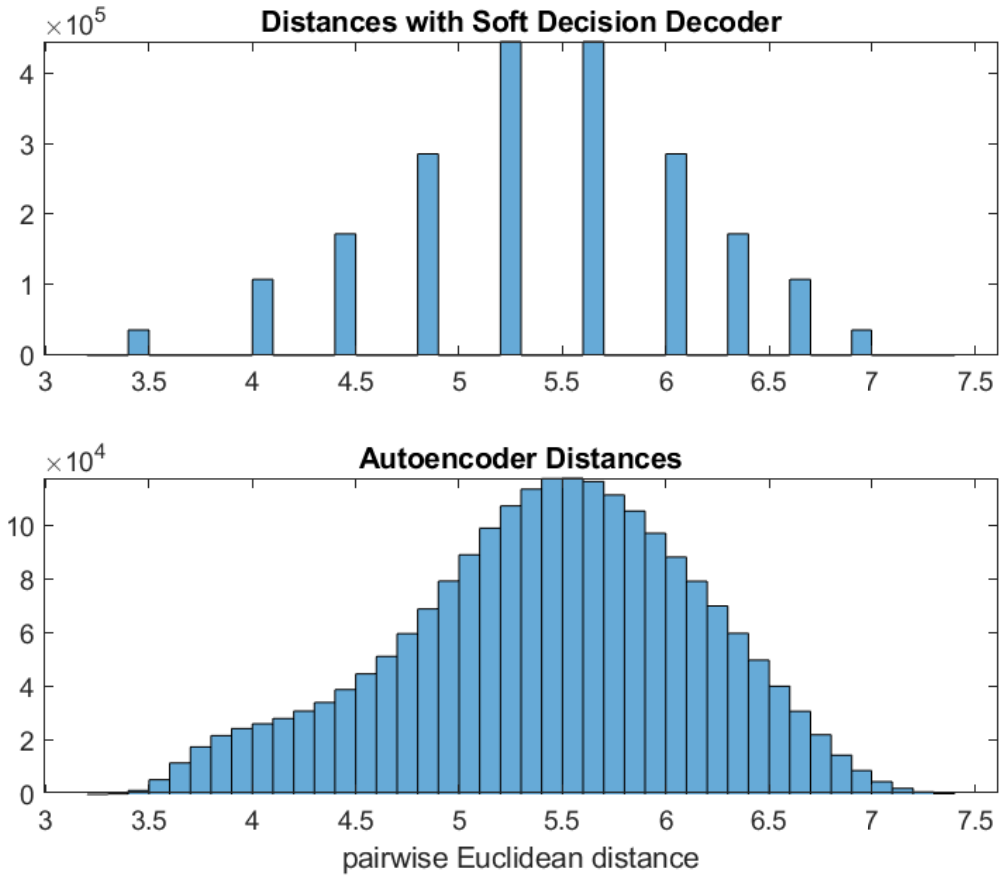

Figure 3.15: Histogram of pairwise Euclidean distances for Hamming $(15, 11)$ BPSK-modulated codewords and autoencoder $(15, 11, 15)$ learned codewords.

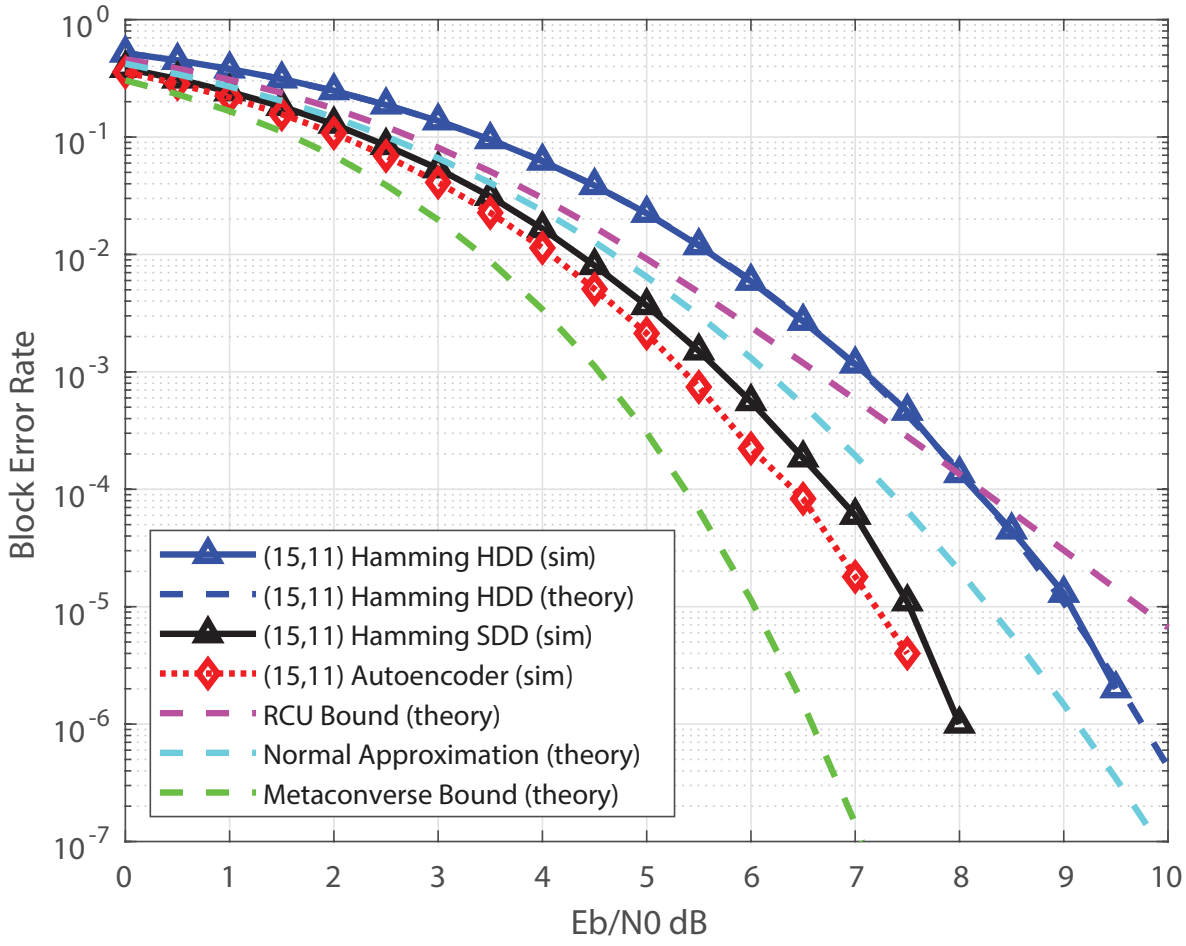

Figure 3.16: BLER of the trained $(15, 11, 15)$ autoencoder and BPSK-modulated Hamming $(15, 11)$ in an AWGN$(\mathcal{E}_b/N_0)$ channel. Random coding union (RCU), metaconverse, and normal approximation bounds are also plotted.

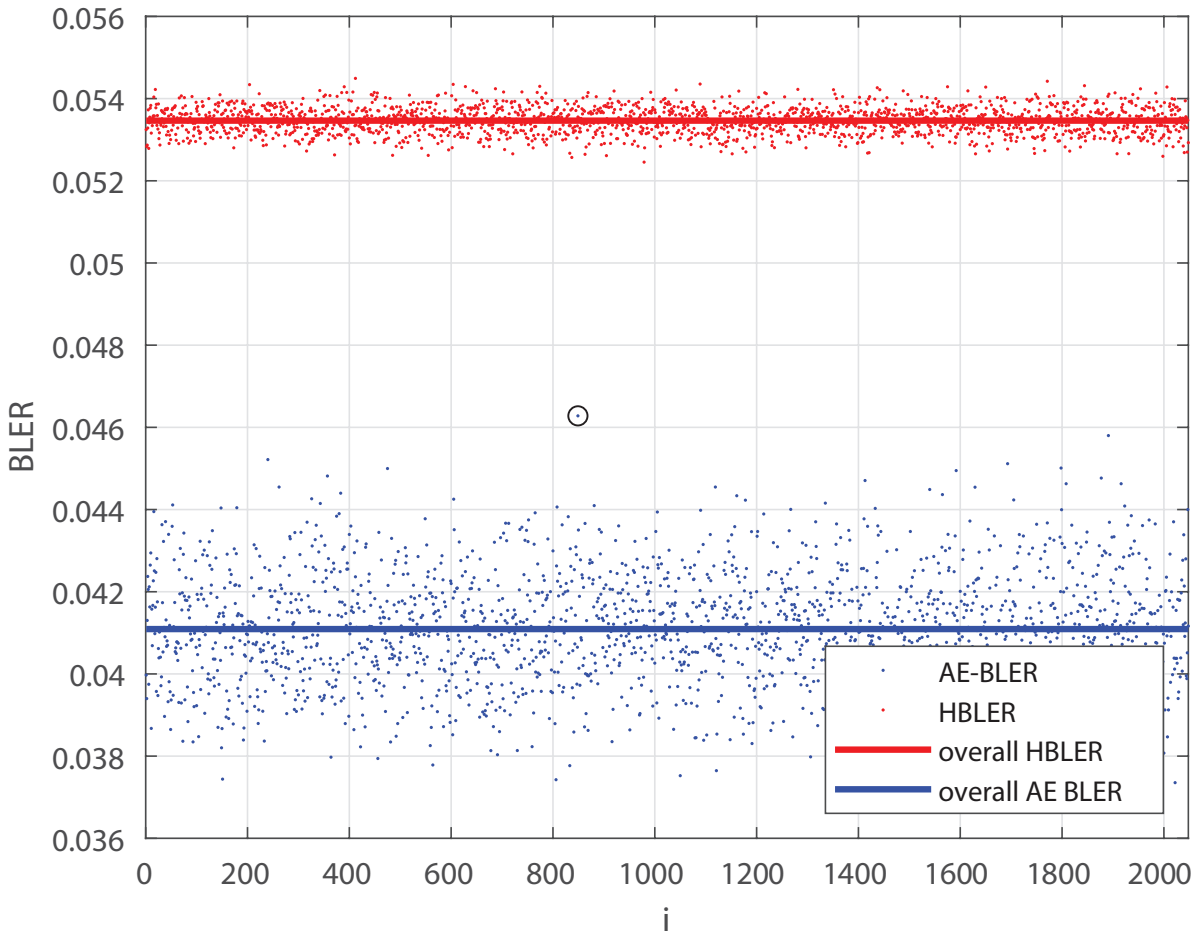

Figure 3.17: BLER for each codeword of the trained $(15, 11, 15)$ autoencoder and BPSK-modulated Hamming $(15, 11)$ in an AWGN($\mathcal{E}_b/N_0$) channel.

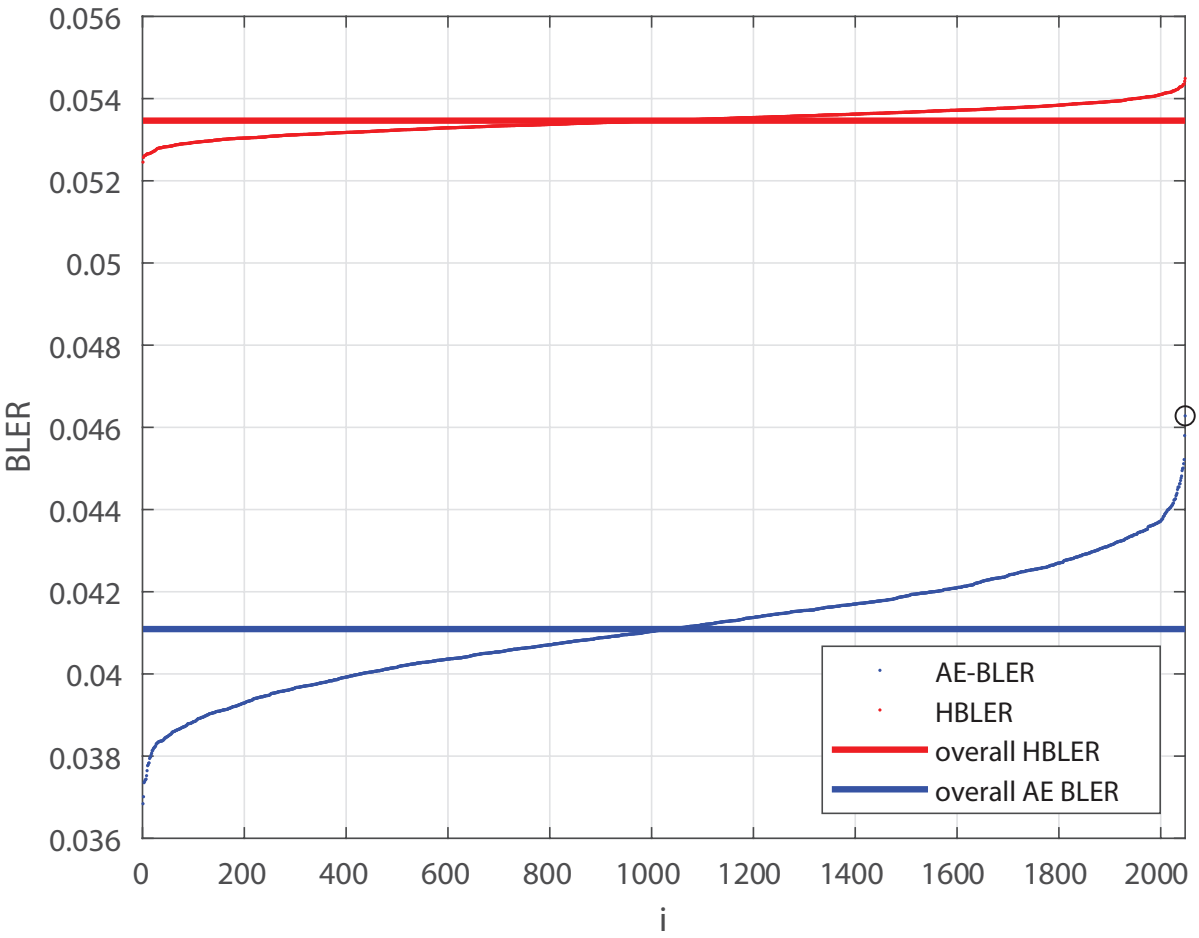

Figure 3.18: BLER for each codeword (sorted) of the trained $(15, 11, 15)$ autoencoder and BPSK-modulated Hamming $(15, 11)$ in an AWGN($\mathcal{E}_b/N_0$) channel.

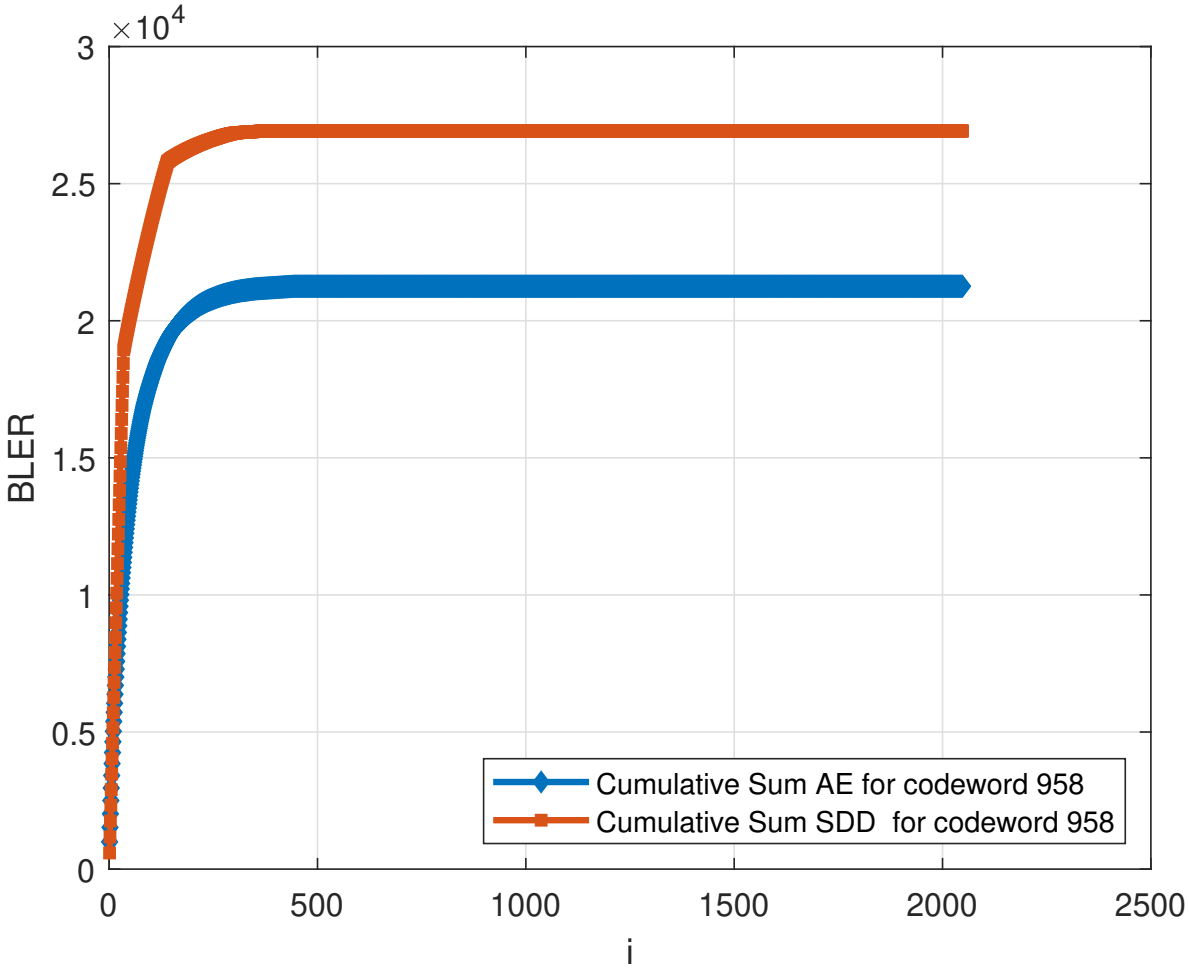

Figure 3.19: Cumulative Sum for codeword 849 with the trained $(15, 11, 15)$ autoencoder and BPSK-modulated Hamming $(15, 11)$ in an AWGN($\mathcal{E}_b/N_0$).

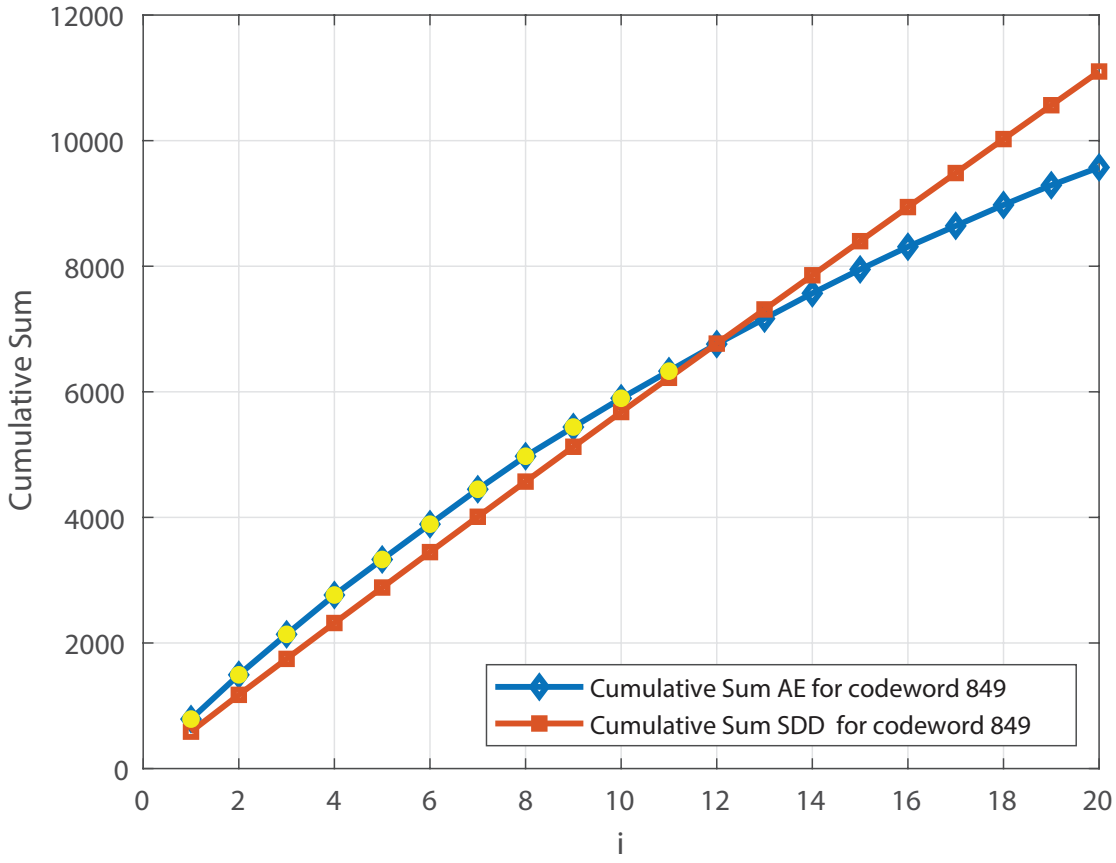

Figure 3.20: Zoomed-out version of the cumulative sum for the first 20 distances $(15, 11, 15)$ autoencoder.

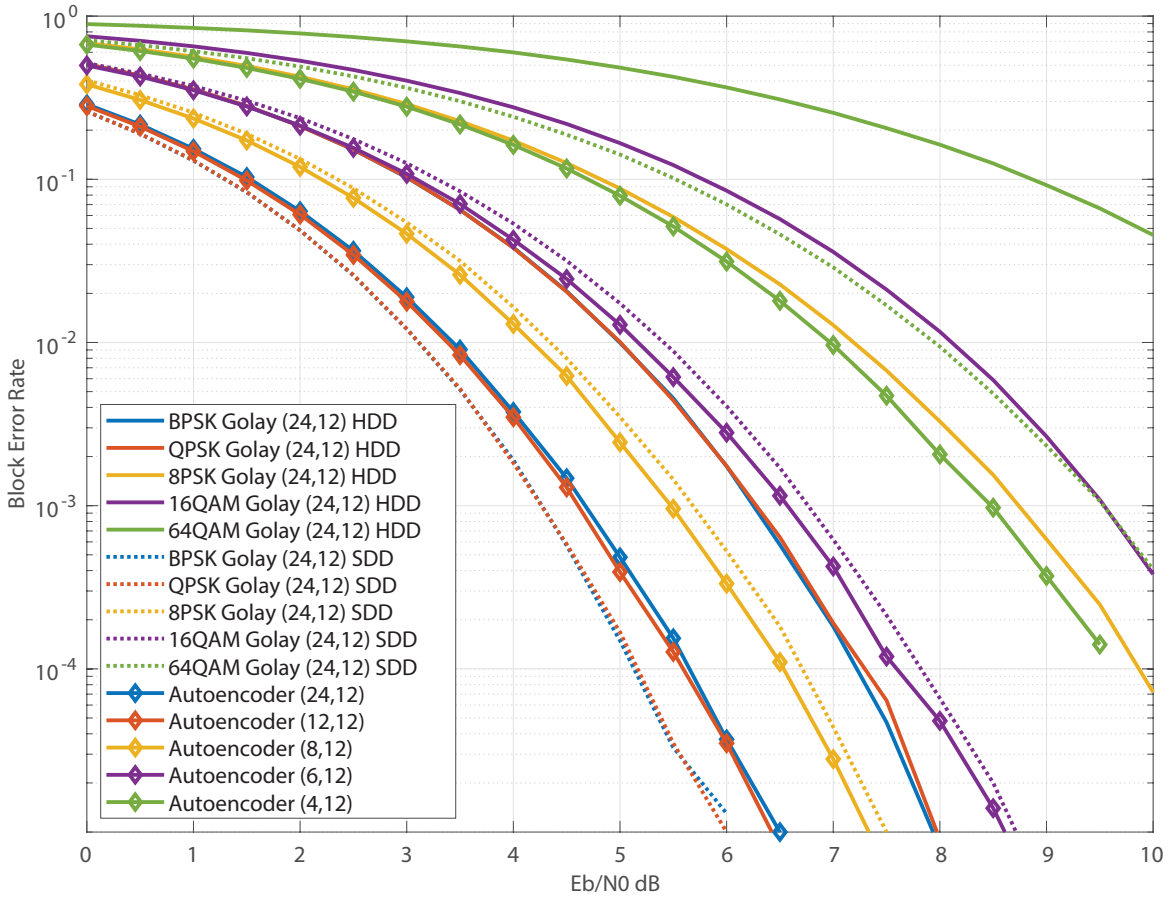

Figure 3.21: AWGN channel block error rate comparison of higher-order-modulated extended Golay $(24, 12)$ with hard decision and soft decision decoding and the corresponding trained autoencoder.

Chapter 4

Joint Coding and Modulation in Non-AWGN Channels

In this chapter, we tackle a non-standard communication model, a BGIN channel which has no-known good codewords. We apply ideas from [8], [17], [19]–[22] toward the development of new codes for the BGIN channel which, to the best of our knowledge, has not been studied in this context.

Impulsive noise is characterized by non-stationary large noise samples that significantly degrade the performance and reliability of communication systems [27], [28]. Since a single Gaussian noise model is not correct and many distributions such as BGIN channel model, Middleton Class A, B and Symmetric Alpha Stable (SαS) distributions are developed.

We consider a BGIN channel that is widely applied on impulsive noises. In fact, soft decision decoding, while optimal in the AWGN channel, can perform worse than hard decision decoding in impulsive noise channels. This has led to the development of impulsive noise mitigation techniques such as blanking and clipping [30]–[32].

4.1 Key Contributions

In this context, the contributions in this chapter are:

1. Training and developing a family of autoencoders for a given probability distribution in impulsive noise channels to minimize block error rate.
2. Demonstration using numerical examples with Hamming $(7, 4)$ and $(15, 11)$ codes with BPSK modulation that the family of autoencoders approach beats both traditional and heuristic approaches.

4.2 Block BGIN Channel

We consider a Bernoulli-Gaussian version of Middleton Class-A noise model with two channels: good and bad, both Gaussian. We assume that the true channel is good, and it is being corrupted by a bad channel with a probability p_b. However, the state with impulse noise does not necessarily need to have a Gaussian distribution. For the states with impulse noise, the impulse noise variance is decided by the probability of entering that state.

The block BGIN channel can be represented by

$$\boldsymbol{Y} = g(\boldsymbol{X}) = \boldsymbol{X} + (\boldsymbol{I}_m - \boldsymbol{B})\boldsymbol{Z}_0 + \boldsymbol{B}\boldsymbol{Z}_1 \tag{4.1}$$

where $\boldsymbol{X} \in \mathcal{X}^m$, $\boldsymbol{Z}_0 \sim \mathcal{N}(0, \sigma_0^2 \boldsymbol{I}_m)$, $\boldsymbol{Z}_1 \sim \mathcal{N}(0, \sigma_1^2 \boldsymbol{I}_m)$ and $\boldsymbol{B} = \text{diag}(b_1, \ldots, b_m)$ where b_i are i.i.d. Bernoulli random variables with $b_i = 1$ with probability p_b and $b_i = 0$ otherwise. In typical impulsive noise channels we assume $\sigma_1^2 \gg \sigma_0^2$ such that p_b represents the probability of the occurrence of high variance impulsive noise for a given channel input.

Such a channel can be compactly represented as $\text{BGIN}(\mathcal{E}_b/N_0, \mathcal{E}_b/N_1, p_b)$ where $\mathcal{E}_b/N_0$ is the SNR when the noise has low variance, $\mathcal{E}_b/N_1$ is the SNR when the noise has high variance, and p_b is the probability the high noise variance channel. This is shown in Figure 4.1.

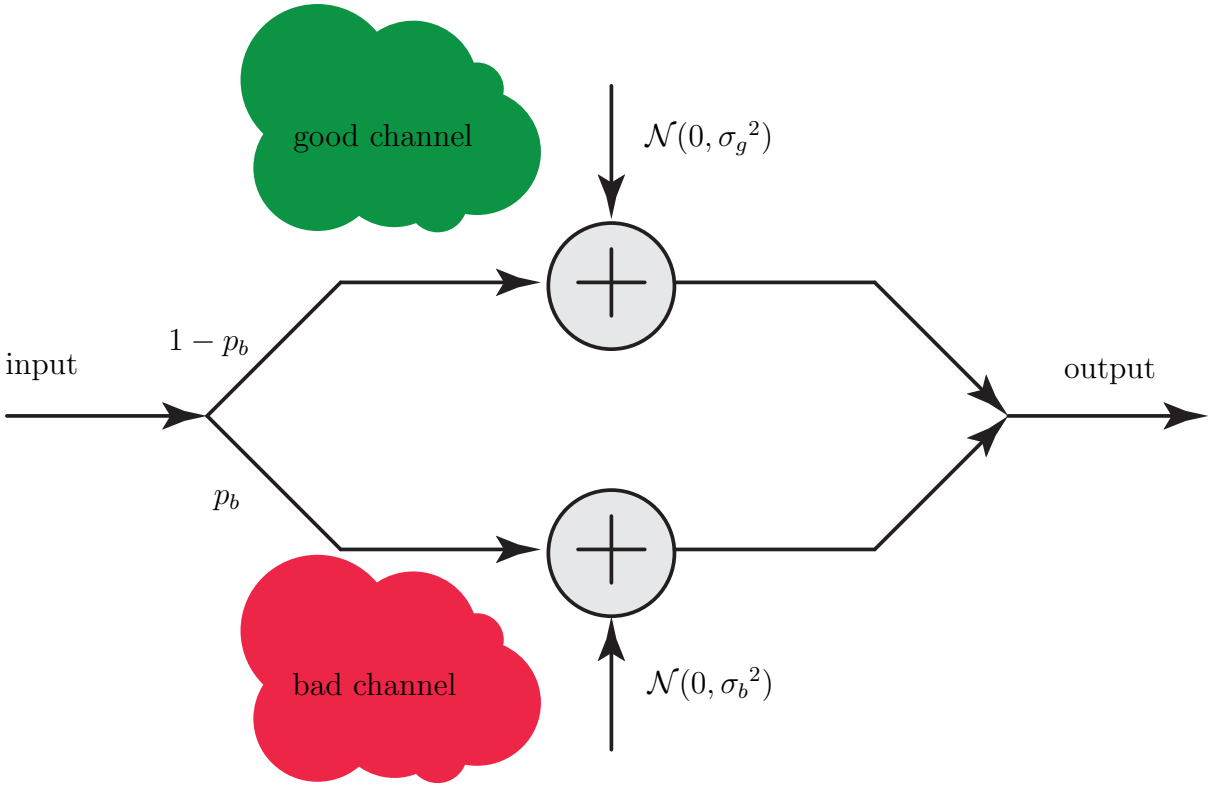

Figure 4.1: Bernoulli-Gaussian Impulsive Noise (BGIN) channel.

In a BGIN channel, if we have a codeword 0110101 for example, it can either propagate through a good or a bad channel with a pre-determined p_b. We have a few possibilities for this as shown in Figure 4.2.

1. If $p_b = 1/7$, one out of seven codewords is randomly sent to the bad channel and others are sent to the good channel i.e. we do not know what bit is being sent.
2. When the same codeword is sent with $p_b = 3/7$ in two instances, we do not get the same output since the choice of what bits get sent to the good/bad channel is made randomly.
3. As the probability p_b increases, there are more chances of entering the bad channel.

This channel is not a straightforward additive noise variation as it picks elements from codewords randomly and operates on them. So, traditional decoding schemes for AWGN channels such as hard decision and soft decision decoding will not work in this case.

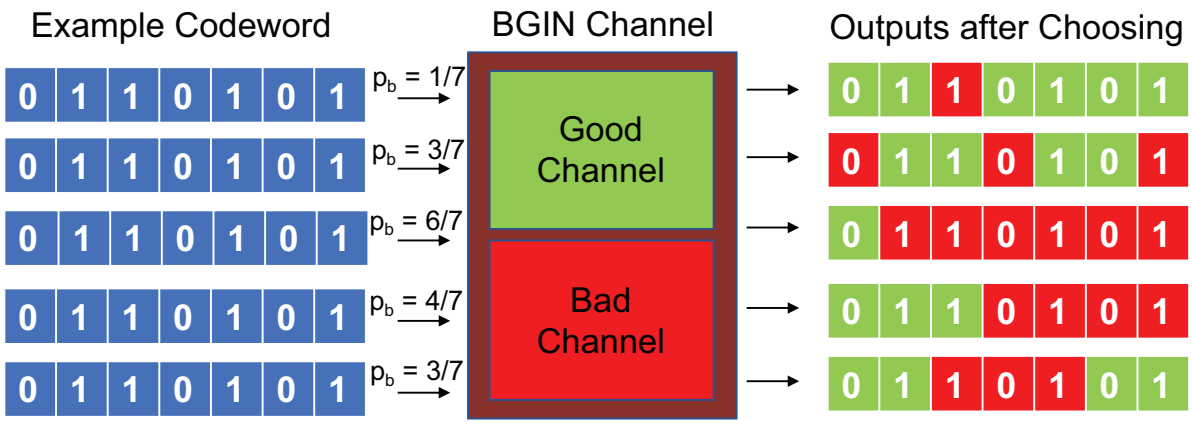

Figure 4.2: Example: A codeword passing through a BGIN Channel.

4.3 Traditional Methods to Mitigate Noise in BGIN channel

A common approach for mitigating the effects of impulsive noise is to use clipping or blanking before demodulation [70]. With clipping, the received signal is limited to a clipping threshold, i.e.,

$$y_{\text{clipped}} = \begin{cases} y & |y| < T_c \\ \text{sign}(y)T_c & |y| \geq T_c \end{cases}$$

where T_c is the clipping threshold. Similarly, with blanking, the received signal is set to zero if it exceeds a threshold, i.e.,

$$y_{\text{blanked}} = \begin{cases} y & |y| < T_b \\ 0 & |y| \geq T_b \end{cases}$$

where T_b is the blanking threshold. Figure 4.3 shows a demonstration of clipping and blanking methods on an impulsive noise channel. We see that the clipping approach clips the value above the threshold (red dashed line) to be equal to the threshold. Blanking sets the values that cross the threshold to 0.

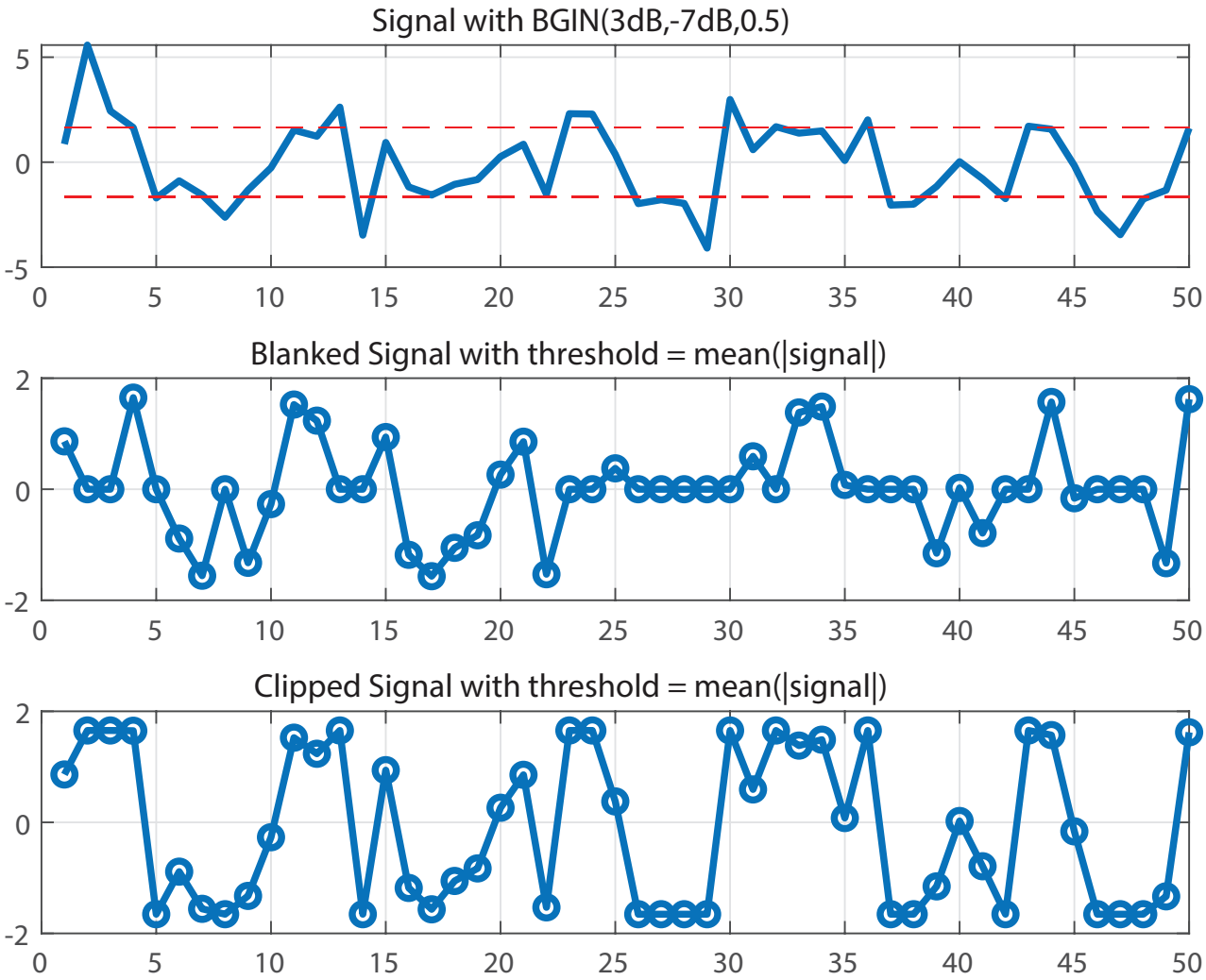

Figure 4.3: Demonstration of clipping and blanking approaches for a BGIN channel.

4.4 Autoencoder for BGIN Channels

In this section, we take a similar approach used in Section 3.6 and apply it to BGIN channels. The main difference here is that the autoencoder is trained separately on each Bernoulli-Gaussian probability $p_b \in \{0, 0.1, \ldots, 1\}$. This results in a family of learned autoencoders indexed by p_b. The training parameters are otherwise identical to those in Section 3.6.2. The training and validation accuracies are shown in Figure 4.4 for p_b= 0.1.

Here is a training example for BGIN(3dB, -7dB, p_b) for $M = 2^4, m = 7$

- *Family of Autoencoders*: Train on each p_b for $p_b \in \{0, 0.1, \ldots, 1\}$.
- *Parameters*: 50 epochs, Adam optimizer, learning rate = 10^{-3}, batch size = 256

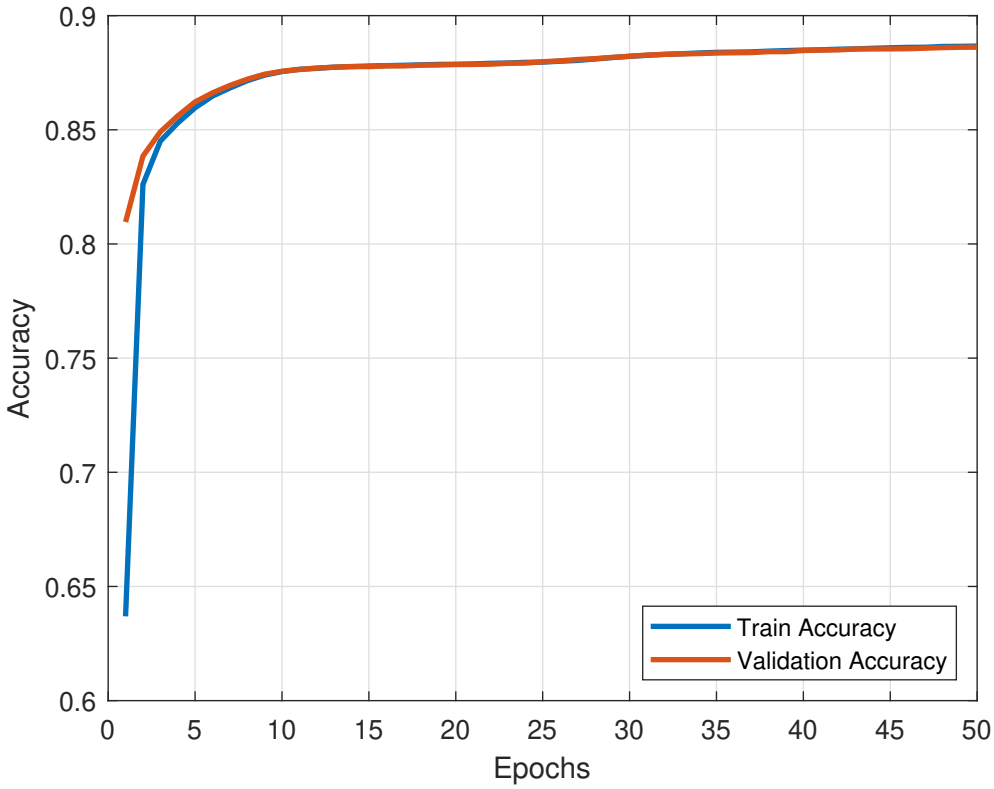

Figure 4.4: Training and validation accuracy curve for BGIN(3dB, −7dB, 0.1) autoencoder.

4.5 Results

Figure 4.5 plots the achieved BLER of the family of trained $(7, 4, 7)$ autoencoders in BGIN(3dB, −7dB, p_b) channels. The BLER of BPSK-modulated Hamming $(7, 4)$ codes with various combinations of hard decisions, soft decisions, blanking, and clipping are also plotted for comparison. The clipping and blanking thresholds were set to $T_c = T_b = \text{mean}(|y_k|)$. In this example, the autoencoder uniformly outperforms conventional coding and modulation, with or without clipping or blanking. The cyan and magenta lines represent the AWGN performance of $(7, 4)$ Hamming codes with BPSK modulation (corresponding to $p_b = 0$ and $p_b = 1$). When $p_b = 0$, channel symbols are always sent through the AWGN(7dB) channel (the less noisy channel). When $p_b = 1$, channel symbols are always sent through the AWGN(−3dB) channel (the more noisy channel). Observe

that the autoencoder is more robust than hard and soft decision decoding, even with clipping or blanking, at all values of p_b. This is because the autoencoder is trained to minimize the BLER at each value of p_b. Blanking erases both the impulsive noise and the useful signal on the selected samples. Clipping tends to perform better than blanking since it is similar to performing hard decisions.

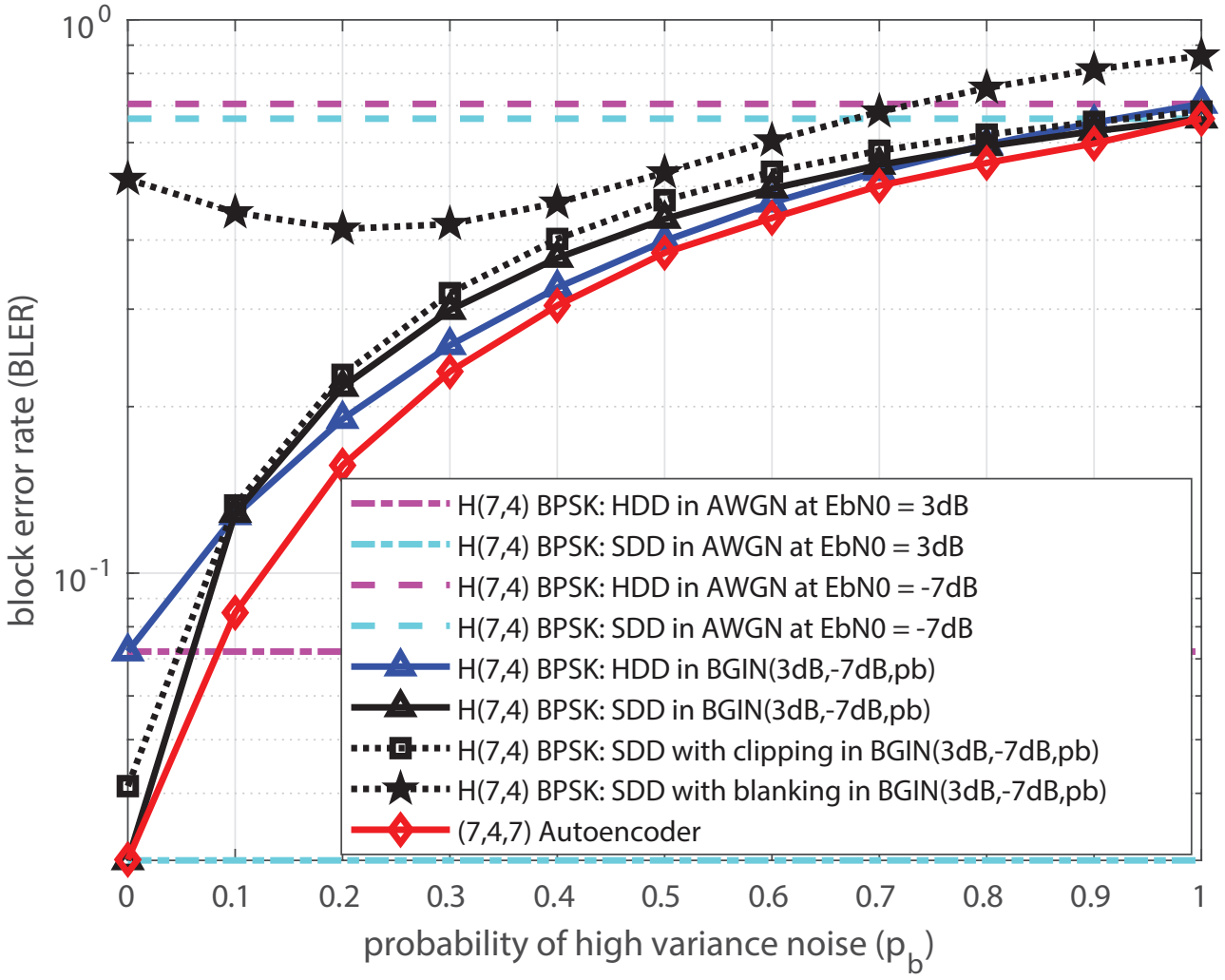

Figure 4.5: BLER comparison of the family of trained $(7, 4, 7)$ autoencoders with BPSK-modulated Hamming $(7, 4)$ in BGIN$(3\text{dB}, -7\text{dB}, p_b)$ channels.

Similarly, Figure 4.6 shows the BLER performance of the family of trained $(15, 11, 15)$ autoencoders in comparison with BPSK-modulated $(15, 11)$ Hamming codes in the impulsive channel BGIN$(3\text{dB}, -7\text{dB}, p_b)$. Again, the achieved BLER of the autoencoder uniformly outperforms conventional coding and modulation with and without clipping

and blanking with

$$T_c = T_b = \text{mean}(|y_k|).$$

This example shows that, even with longer blocklength codes, an autoencoder trained to minimize the BLER in impulsive noise is more robust than the conventional methods at all values of p_b. The training was done on 5×10^6 examples and the parameters are otherwise identical to those of the $(15, 11, 15)$ autoencoder.

As a function of probability of impulsive noise where the low-noise-variance channel has $\mathcal{E}_b/N_0 = 3$ dB and the high-noise-variance channel has $\mathcal{E}_b/N_0 = -7$ dB. The probabilities $p_b = 0$ and $p_b = 1$ indicate that the channel is AWGN with $\mathcal{E}_b/N_0 = 3$ dB and $\mathcal{E}_b/N_0 = -7$ dB respectively. We can also observe that hard decision decoding fares better than soft decision decoding as when the probability of switching is around 0.5 as it makes more sense to use hard-decisions on the received signal than use a matched filter. beats both hard decision and soft decision curves as it learns the joint coding and modulation at any given probability. In these cases, impulse noise will introduce a penalty in the performance as compared to pure Gaussian noise.

With a lower threshold T_b, the blanking algorithm is more likely to delete samples with large amplitude which are in fact not affected by impulsive noise. However, both approaches are dependent on their threshold values. To be consistent, we set both T_b and T_c to be equal to $mean(|y_k|)$,

4.6 Conclusion

Numerical results show that the trained autoencoder uniformly outperforms classical block codes with BPSK modulation in the BGIN channel even when impulsive noise mitigation techniques such as blanking and clipping are employed. The proposed architecture is general and can be modified for comparison against other block coding schemes

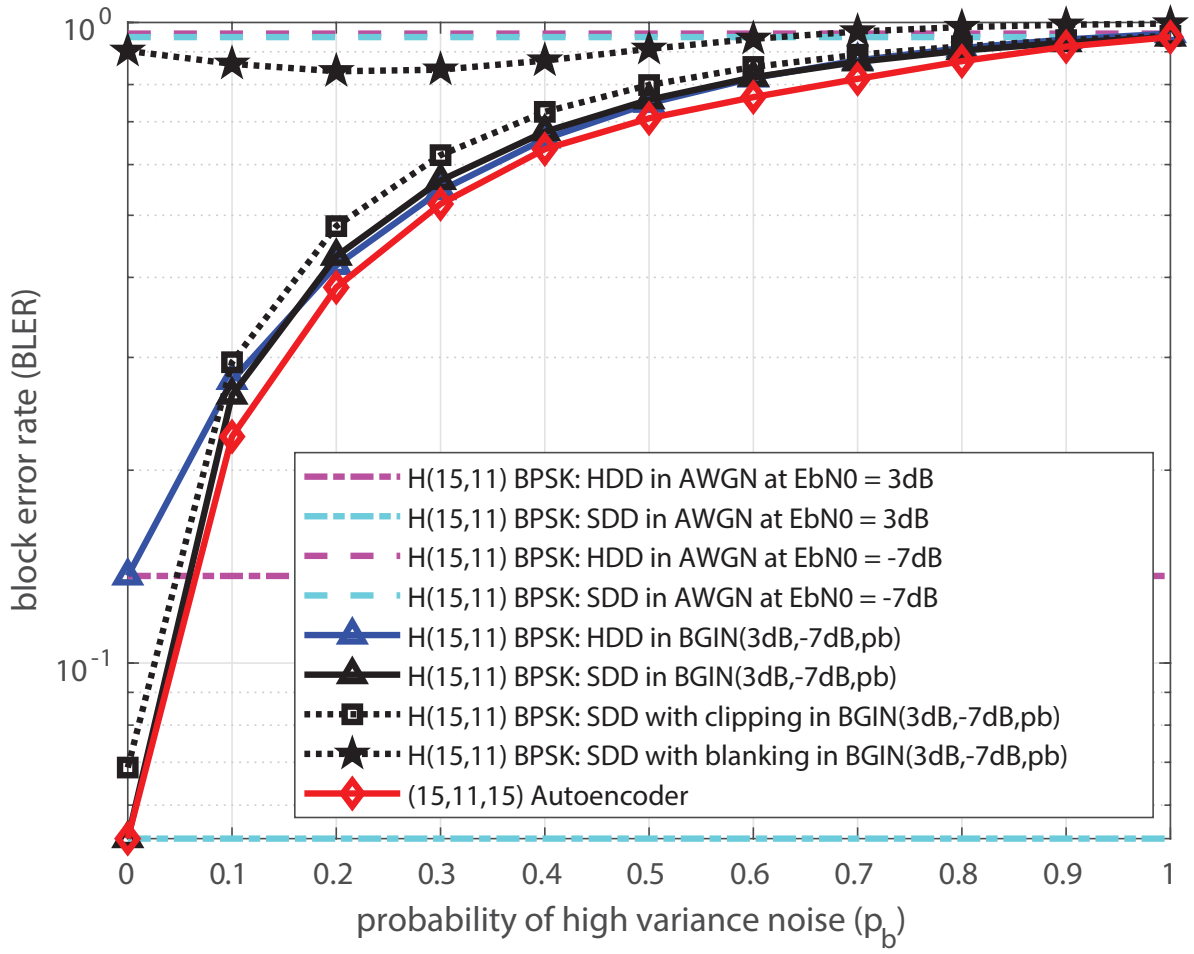

Figure 4.6: BLER comparison of the family of trained $(15, 11, 15)$ autoencoders with BPSK-modulated Hamming $(15, 11)$ in BGIN(3dB, -7dB, p_b) channels.

and higher-order modulations.

We also demonstrate that linear block codes generated by the autoencoder have better BLER than traditional methods for impulsive noise channels

4.6.1 Next Steps

For digital communication systems which use error control codes, this BGIN model is too simplistic since it fails to capture the noise memory. It can be extended to include the effect of the noise memory which, as it will be seen, affects the performance of digital communication systems that use error control codes. The extension used here assumes that memory keeps the noise, for a multiple of L (the memory length), in either the

background state (weak noise) or in the impulsive state (strong noise). We start thus defining an BGIN with memory (ABGNM) with four parameters

In a modulation scheme with memory, the mapping is from the set of the current k bits and the past $(L-1)k$ bits to the set of possible $M = 2^k$ messages. Parameter L is called the constraint length of modulation. The case of $L = 1$ corresponds to a memoryless modulation scheme.

Part II

Machine Learning for Tracking Dynamical Systems

Chapter 5

Oscillator Phase Predictions

Characterization and modeling of clock oscillator stability is important for many applications requiring an accurate time and/or frequency reference. Oscillator stability has been traditionally characterized by the Allan variance and stochastic models originally developed for high precision, high cost sources such as atomic clocks. Knowledge of model parameters allows development of tracking and prediction techniques which enable accurate prediction of and compensation for oscillator drift.

The Kalman Filter [37] provides the minimum mean squared error solution to linear system when the process under observation is completely represented by the state model. However, its success relies on the knowledge of the system models. When the models are unknown or partially known, it is hard to determine the covariances of the process and measurement noises [38]. This can happen when we do not have high-end oscillators for tracking. The traditional model-based approach does not adapt to the inherent parametric mismatch. So, we approach this from a machine learning point-of-view and develop methods that can solve this problem.

This chapter focuses on this application after considering a general dynamical system and developing a machine learning based approach to tackle parametric mismatch.

5.1 Key Contributions

In this context, the contributions are:

1. We formulate a dynamical system as a time-series forecasting problem and design and develop data-driven one-dimensional CNN-based approach to predict dynamic random processes. We use state observations to predict the next internal state.
2. We show a proof-of-concept demonstration with oscillator phase predictions. Here, $[q_1, q_2]$ are the actual short-term and long-term stabilities of the oscillators and r is the measurement noise parameter. We study the performance of a Kalman filter that uses mismatched parameters $[q_1', q_2', r']$ to better understand the sensitivities of the Kalman filter to parameter mismatches.
3. We use a *circular mean-squared-error* loss function to predict phases.
4. We use numerical examples to compare performance of machine learning approach using steady-state analysis as a reference for the optimal behavior of the Kalman filter and to compare our results

We demonstrate that our method, unlike traditional KF based methods, is robust to parametric mismatches by comparing the error covariances.

5.2 System Model

We consider a general framework for dynamical systems here. Under a fairly general discrete-time framework, the system state and measurement evolve according to

$$x[k+1] = \boldsymbol{f}(k, x[k], v[k])$$
$$y[k] = \boldsymbol{h}(k, x[k], w[k])$$

where $x[k]$ is the state vector, $y[k]$ is the measurement vector, $v[k]$ is the random process noise, $w[k]$ is the random measurement noise, $\boldsymbol{f}(\cdot)$ is a family of N vector functions

describing the state dynamics during mode θ_k, $\boldsymbol{h}(\cdot)$ is a family of N vector functions describing the measurement dynamics, and $\theta_k \in \{0, 1, \ldots, N-1\}$ denotes the stochastic mode in effect during the sample period ending at discrete time k. If, in addition, the system is linear, it admits the state space realization

$$x[k+1] = \boldsymbol{F}[k]x[k] + v[k] \tag{5.1}$$

$$y[k] = \boldsymbol{H}[k]x[k] + w[k] \tag{5.2}$$

where the non-linear functions $\boldsymbol{f}(\cdot)$ and $\boldsymbol{h}(\cdot)$ are replaced by the matrices $\boldsymbol{F}[k]$ and $\boldsymbol{H}[k]$. When the process and measurement noises can be modeled as Gaussian random processes, we assume they are distributed as $v[k] \sim \mathcal{N}(0, \boldsymbol{Q})$ and $w[k] \sim \mathcal{N}(0, \boldsymbol{R})$, respectively, with $\boldsymbol{Q}$ and $\boldsymbol{R}$ as the corresponding covariance matrices. The process noise covariance is $\boldsymbol{Q} = \text{diag}({q_1}^2, \ldots, {q_N}^2)$.

We study the effect of parametric mismatch in a two-state oscillator tracking system. From [71], the state of the oscillator for a two-state model can be written as $\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) & x_2(t) \end{bmatrix}^\top$ where $x_1(t)$ is the time offset in seconds and $x_2(t)$ is the rate or frequency offset. This can be seen in Figure 5.1.

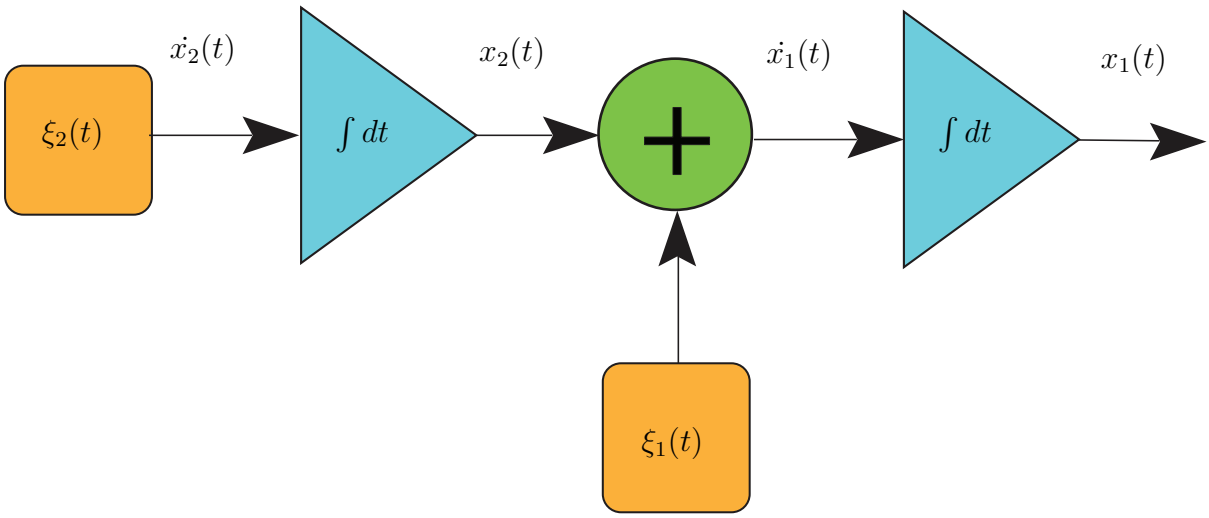

Figure 5.1: Schematic for a two-state model.

The continuous-time dynamics follow as

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x(t) + \xi(t) \tag{5.3}$$

with $\xi(t) \sim \mathcal{N}(0, \boldsymbol{Q})$ and process noise covariance $\boldsymbol{Q} = \text{diag}({q_1}^2, {q_2}^2)$ where q_1 and q_2 are the short-term and long-term stabilities of the oscillators.

The corresponding discrete-time dynamics are

$$x[k+1] = \underbrace{\begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}}_{\boldsymbol{F}} x[k] + u[k] \tag{5.4}$$

with $u[k] \sim \mathcal{N}(0, \boldsymbol{Q}(T))$ and the discrete process noise covariance

$$\boldsymbol{Q}(T) = q_1^2 \begin{bmatrix} T & 0 \\ 0 & 0 \end{bmatrix} + q_2^2 \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} \\ \frac{T^2}{2} & T \end{bmatrix} \tag{5.5}$$

The discrete-time observation equation is given by

$$y[k] = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{\boldsymbol{H}} x[k] + w[k] \tag{5.6}$$

with $w[k] \sim \mathcal{N}(0, r)$ and the sample period T. Here, the three parameters (q_1, q_2, r) describe the statistics of the dynamics of this system. To better understand the sensitivities of KF at steady state, the results are also checked with its steady state performance using the standard Riccati equation method.

The KF estimates and predicts the unwrapped phase naturally. However, the wrapped phase measurements can only correctly detect a phase change from one cycle to the next that is less than π [72]. The measurement for phases ϕ_k and ϕ_1 is $\min_k(2\pi k + \phi_1 - \phi_k)$ where $k \in \mathbb{Z}$.

5.3 Machine Learning Problem Formulation

In this section, we take a quick detour to discuss a typical problem formulation for a supervised learning problem. We discuss the ideas of features, labels, loss function and activation functions here.

Assuming a supervised learning scenario with input space $\mathcal{X}$, an output space $\mathcal{Y}$ and a training set $\mathcal{T}$ that includes N training examples $\mathcal{T} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $x_i \in \mathcal{X}$ is the feature vector and $y_i \in \mathcal{Y}$ is its label. A learning algorithm tries to find a function f_θ so that for each feature vector in $\mathcal{X}$ will be correspond with a value in $\mathcal{Y}$. In order to measure the fitting of the function with the training data, a loss function $\mathcal{L}$ is defined. For training example (x_i, y_i), the loss of predicting value $\hat{y}$ is $\mathcal{L}(y_i, \hat{y})$.

If we consider a neural network model with a weight matrix $\boldsymbol{W}$, bias vector $\boldsymbol{b}$ and a feature set matrix $\boldsymbol{X}$, the linear output units produce a vector $\hat{y} = \boldsymbol{W}^\top \boldsymbol{X} + \boldsymbol{b}$. To find the best approximation f_θ of some function, we can formulate our problem to the standard form yields

$$\min_{\theta} \quad \frac{1}{m} \sum_{t=1}^{m} \frac{1}{2} \parallel \hat{y} - y \parallel^2 \tag{5.7}$$

$$\text{subject to} \qquad \parallel \boldsymbol{W} \parallel^2 \leqslant \epsilon, \tag{5.8}$$

and some of the commonly used activation functions are

- **Sigmoid**:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.9}$$

- **Tanh**

$$\tanh(x) = 2\sigma(2x) - 1 \tag{5.10}$$

- **Rectified Linear Unit (ReLU)**

$$f(x) = \max(0, x) \tag{5.11}$$

- **Maxout**:

$$f(w^T x + b) = \max({w_1}^T x + b_1, {w_2}^T x + b_2) \tag{5.12}$$

- **Softmax**:

$$f(x) = \frac{e^x}{\sum_{j=1}^{n} e_j^x} \tag{5.13}$$

5.4 Methodology

We study the performance of a Kalman filter that uses parameters $[q_1', q_2', r']$. When these parameters match the actual parameters of the system, we know the Kalman filter generates MMSE estimates and predictions and we can solve for the steady-state performance of the Kalman filter using standard Riccati equation methods. When the Kalman filter uses different parameters, however, there will be some loss of performance. We would like to better understand the sensitivities of the Kalman filter to parameter mismatches.

As a first step toward understanding parametric sensitivities, we will assume two of the three parameters to be correct and run the Kalman filter on synthesized data with the remaining third parameter incorrectly chosen. We will let the Kalman filter converge and then compute the Monte-Carlo one-step prediction and estimation covariances. Note that the Monte-Carlo results will be different from the Kalman filter error covariance matrices due to the parameter mismatch. The Monte-Carlo results describe the actual prediction and estimation performance of the Kalman filter.

The covariance update with parameters $[q_1', q_2', r']$ is given as

$$\boldsymbol{\Sigma}[k+1 \,|\, k] = \boldsymbol{F}(T)\boldsymbol{\Sigma}[k \,|\, k]\boldsymbol{F}^\top + Q'(T) \tag{5.14}$$

with

$$Q'(T) = \omega_0^2 (q_1')^2 \begin{bmatrix} T & 0 \\ 0 & 0 \end{bmatrix} + \omega_0^2 (q_2')^2 \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} \\ \frac{T^2}{2} & T \end{bmatrix}. \tag{5.15}$$

We specify a "mismatch factor" γ corresponding to the ratio of the Kalman filter param-

eter to the actual parameter governing the state dynamics. For the oscillator stability parameters q_1 and q_2, we have

$$\gamma = \frac{(q_i')^2}{q_i^2} \tag{5.16}$$

When r is the mismatched parameter, we have $\gamma = \frac{r'}{r}$.

5.5 Bounded Loss Function

Neural networks (NNs) are universal function approximations over finite intervals [73]. This is important in our setting since the states get integrated with white noises at each step and are not going to be on a finite interval. Hence, we enforce a bounded-input bounded-output type of behavior on the data by wrapping the values around bounds $(-B, B)$. There are a few other approaches like differencing, scaling and normalizing, but they cannot be applied in this particular scenario because

- *Differencing* - This creates correlations between successive samples in the data, which will not allow fair comparison with KF because if we set it up to track differences, the process noise and the measurement noise would be individually temporally correlated.
- *Scaling* - For unbounded data and targets, we cannot use a constant scale for training and testing as they are contiguous and might have different scale of values.
- *Normalizing* - Normalizing is straightforward, but denormalizing needs information on the mean and standard deviation for each training example

Hence, we circularly wrap the data and targets to $(-B, B)$ [74]. This will require a customized loss function to compute the mean squared error since if the NN guesses the target to be $B - \epsilon$ and the true target is $-B + \epsilon$, these values are close on the circle and the wrapped error is 2ϵ. We consider this in Section 5.6 where we have an oscillator phase prediction problem where wrapping occurs naturally. Table 5.1 shows the architecture of the convolutional neural network used for training.

Table 5.1: The CNN architecture used for solving oscillator phase predictions problem.

Layer	**Input Size**	**Output Size**	**Activation**
Input	$(5, 1)$	$(5, 1)$	ReLU
Conv-1D	$(5, 1)$	$(5, 20)$	ReLU
Conv-1D	$(5, 20)$	$(5, 15)$	ReLU
Conv-1D	$(5, 15)$	$(5, 10)$	ReLU
Conv-1D	$(5, 10)$	$(5, 5)$	ReLU
Conv-1D	$(5, 5)$	$(5, 1)$	ReLU
Concatenate	$[(5, 1), (5, 1)]$	$(5, 2)$	
Flatten	$(5, 2)$	$(10, 1)$	
Output	$(10, 1)$	1	Linear

For the loss function, when the observations are M units apart, the wrapped phase change estimate is added to the number of rotations to form the wrapped error estimate. The weights and biases are denoted by $\boldsymbol{W}, \boldsymbol{b}$, and the number of measurements by N. Also considering the wrapping, we have the loss function as a circular mean squared error function between the predicted and estimated states

$$\mathcal{L}(x, \hat{x}) = \min_{k} \frac{1}{N} \sum_{n=1}^{N} \|\hat{x}_t(n) - x_t(n) \pm k * 2\pi\|_2^2 \quad k \in \mathbb{Z}. \tag{5.17}$$

5.6 Numerical Results

This section presents the numerical results outlining the performance of obtaining the Kalman filter predictions and the evaluation of the CNN implementation. The estimated prediction variances are compared to the variances provided by the KF error covariance matrices.

We select *Adam* as the optimizer with with initial learning rate lr $= 0.001$ and $\beta_1 = 0.9, \beta_2 = 0.999$ [75]. We use rectified linear units (ReLU) [76] as the activation function for each convolutional layer and a linear activation function for the final layer. Table 5.2 shows the typical mismatch parameters used.

We assume two of the three parameters to be correct and run the KF and the CNN on the synthesized data with the remaining third parameter incorrectly chosen. All of the

Table 5.2: Typical parameters for numerical results.

Parameter	**Mismatch Range**	**Units**	**Meaning**
q_1	$10^{-2} - 10^{2}$	sec	oscillator short-term stability
q_2	$10^{-2} - 10^{2}$	1/sec	oscillator long-term stability
r	$10^{-2} - 10^{2}$	nanosec^2	time offset meas. noise variance

simulations results shown below assume: sample size $T = 1$, nominal $q_1 = 1$, $q_2 = 1$ and $r = 1$. The Kalman filter is run with the unwrapped measurements and the CNN is run with the wrapped measurements to ensure the bounded-input bounded-output condition.

The results from the KF were averaged over 2×10^4 runs and the final predictions and estimates were then subtracted from the final states to form Monte-Carlo estimates of the prediction and estimation covariances, respectively. We let the KF converge and then compute the Monte-Carlo one-step prediction and estimation covariances. For the parametric mismatch cases, the Monte-Carlo results will be different from the KF error covariance matrices.

For the CNN approach, we train on a sequence of 2×10^4 wrapped phase noise measurements in radians and validate the results. After training, the weights of the neural network are fixed and are used for testing, for which we consider a new dataset containing 1000 measurements, each of length 200. We run the KF with different mismatched parameters and compare its results with the CNN. The train RMSE is calculated using (5.17) as 2.1.

Figures 5.2, 5.3 and 5.4 show the effect of q_1^2, q_2^2 and r parameter mismatch on the KF and the CNN performances. For all figures, the dashed black lines show the performance with no parametric mismatch and were obtained from solving the discrete-time algebraic Riccati equation. The dashed red lines shows the training RMSE for the CNN for each type of mismatch. In the case of no mismatch, KF performs slightly better than CNN as KF is optimal when it has full knowledge of the model, measurement and noise covariance. We can see that while the mismatches degrade the perfomance of KF, they have no effect on the CNN model.

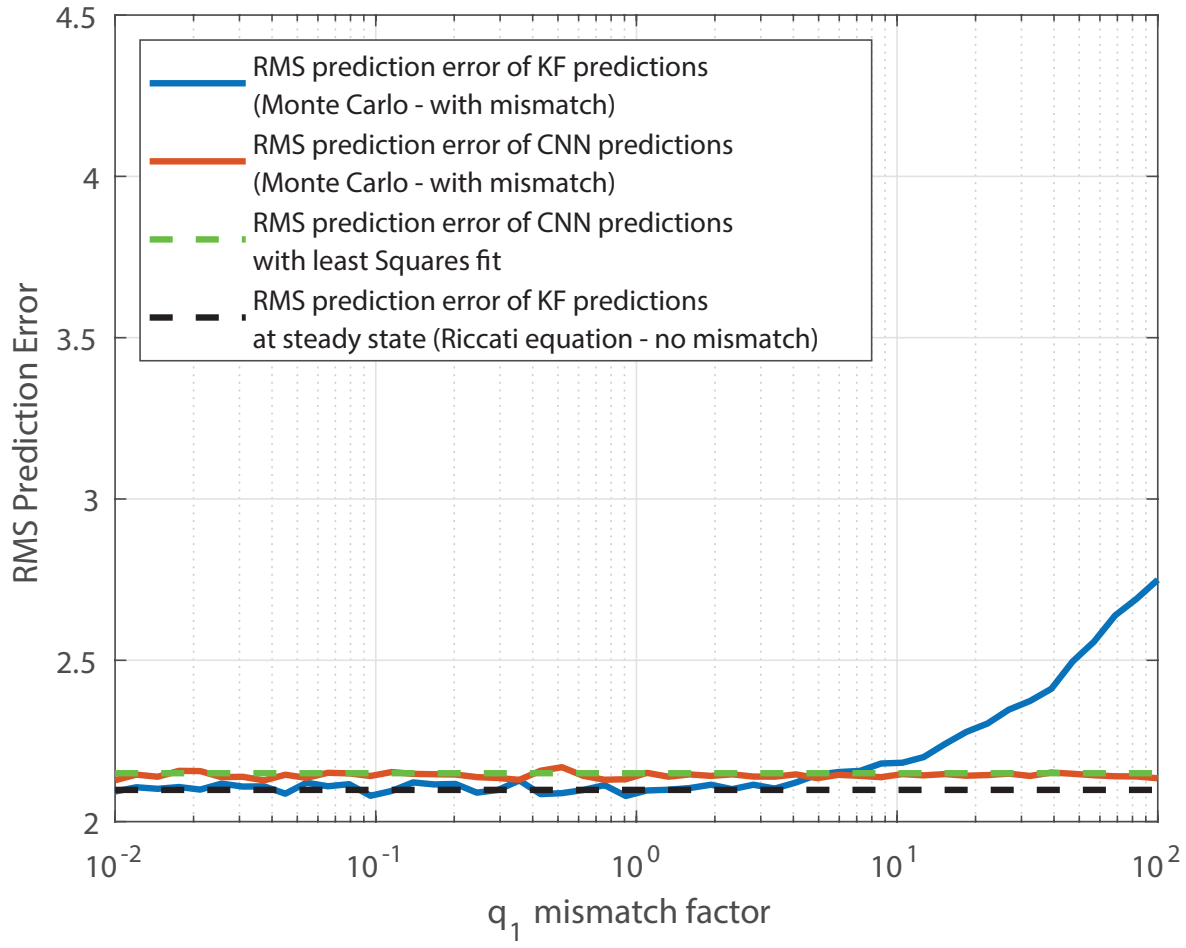

Figure 5.2: Effect of q_1 mismatch on the KF and the CNN performances.

The result from Figure 5.2 shows that the KF is sensitive to parametric mismatch of the q_1 parameter, especially if the parameter is underestimated. These results suggest that overestimation of the q_1 parameter also degrades performance, but the effect is not as severe as underestimation of q_1 by the same factor. However, the performance of the trained CNN is consistent over all the q_1 mismatches.

The result from Figure 5.3 suggests that there is almost no penalty in underestimation of the q_2 parameter by two orders of magnitude (or perhaps even more). Hence, for practical implementations with some parametric uncertainty, it may make sense to bias our estimates of the q_2 parameter toward zero. The trained CNN performs consistently again despite any mismatches.

The result from Figure 5.4 show that the filter is particularly sensitive to overestima-

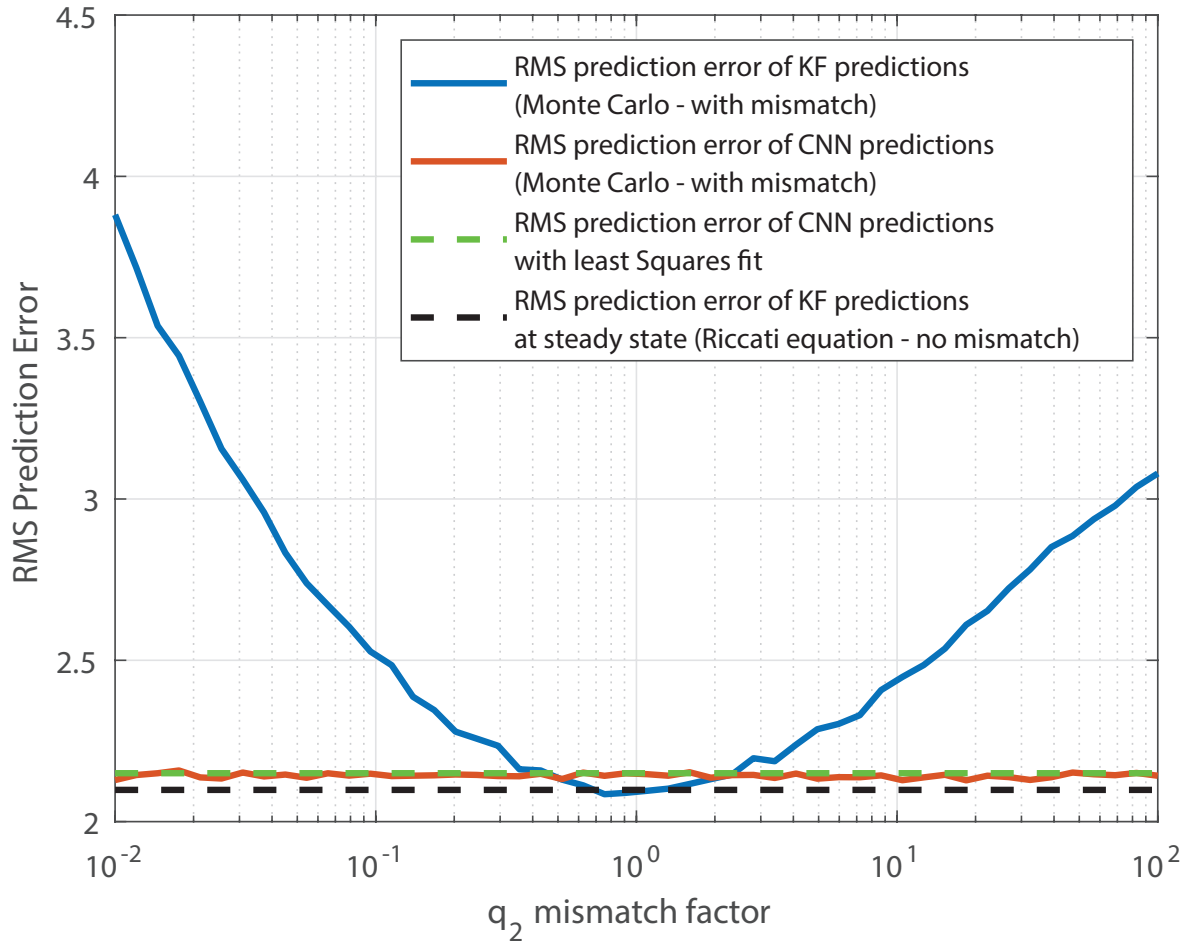

Figure 5.3: Effect of q_2 mismatch on the KF and the CNN performances.

tion of the r parameter but is less sensitive to underestimation of this parameter. The trained CNN performs consistently again despite any mismatches.

5.7 Conclusion and Next Steps

We studied the performance of the Kalman filter with mismatched noise covariances here. We devise a deep learning based approach for tackling the mismatched cases. We take an example of oscillator phase predictions where CNNs improve the performance of the phase predictions as compared to Kalman filter in the mismatched cases.

The trained model generalizes well over a large number of test samples. We demonstrate that our method, unlike traditional KF based methods, is robust to model mis-

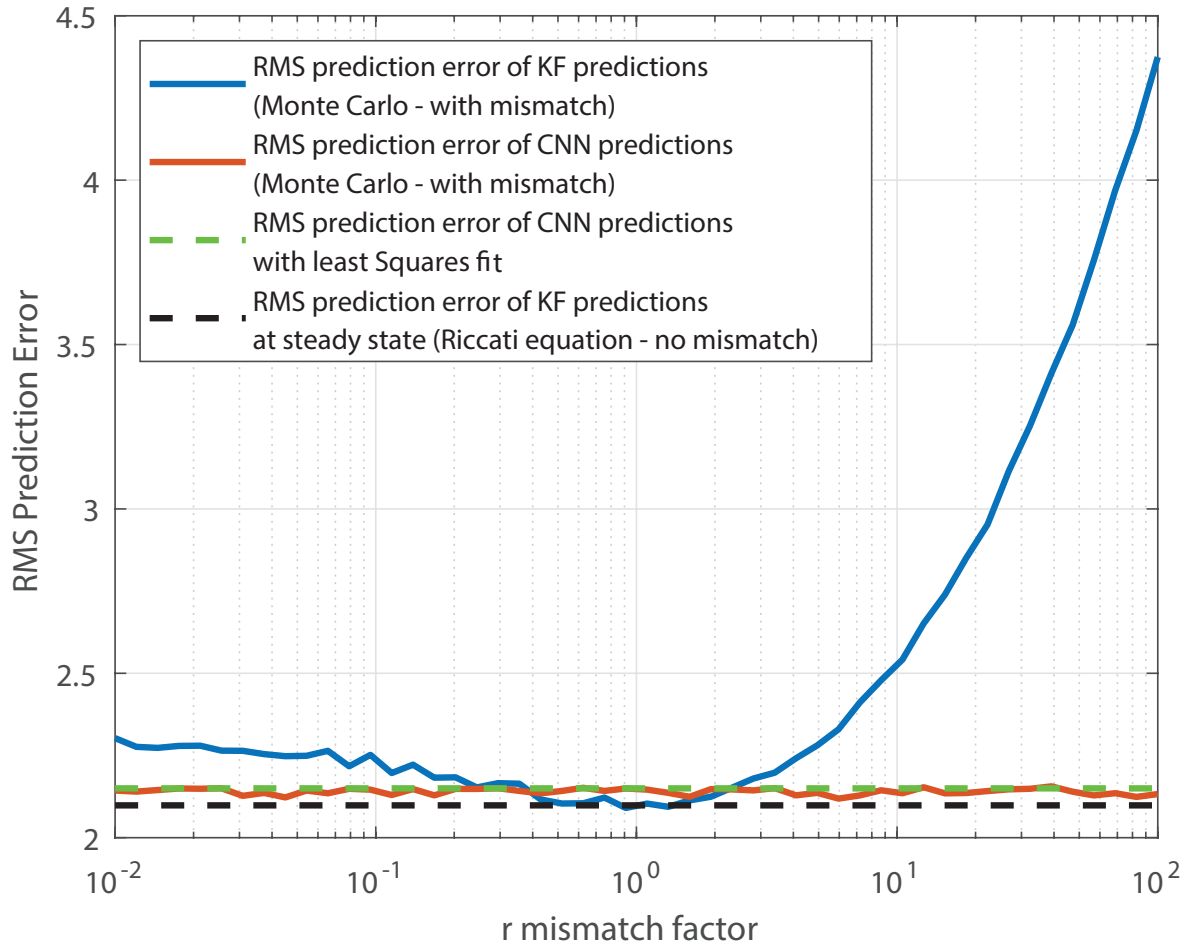

Figure 5.4: Effect of r mismatch on the KF and the CNN performances.

matches by comparing the error covariances. Furthermore, we use steady-state analysis as a reference for the optimal behavior of the Kalman filter and to compare our results.

The proposed approach is general and can be applied to a wide variety of dynamic random process. This will leads to biased estimates of the AR coefficients. Prediction algorithms that are robust to these kind of model mismatches can assist in building advanced tracking methods where the exact model and parameters are inaccurate.

There are many directions that we can extend this work in exploring few other mismatch cases where Kalman filter fails to reach an optimum solution like model mismatch, non-linearities, non-Gaussian and dimension mismatch. We could also consider few other deep learning prediction models like auto-encoders and Gated Recurrent Units. There are reinforcement learning based approaches like model-based, where we estimate the system

model from observations and solve it with reinforcement learning and model-free, where we learn the policy directly without estimating the system model. There are also few techniques where these two approaches can be alternated [77]. It would be interesting to consider these approaches to arrive at a general framework for predicting dynamic random processes.

Chapter 6

Manuevering Target Tracking

(Co-written with Matthew. L. Weiss and Prof. Randy Paffenroth of Worcester Polytechnic Institute and Arick Grootveld, Leah Lackey, Vlad I. Bugayev, Andrew G. Klein, Department of Engineering and Design, Western Washington University, Bellingham, WA 98225)

Kalman filter (KF) comes with a caveat that we need to know all the parameters such as process and measurement noise covariances. Here, in the context of KF, we have four such possible situations:

1. when we do not know the model exactly
2. when we know the model but we do not have the right parameters for the model
3. when we assume a linear model, but the actual model is non-linear
4. when we know the model, but it keeps changing over time - that is, for example, if we keep switching from one model to another. This is called *model switching* and we consider this situation here.

We consider the *tracking maneuvering targets* application in this chapter. Maneuvering targets are difficult to model due to inherent model switching and changes in dynamic behavior. Sharp maneuvers are especially challenging to handle [78].

For example, a turn should be accounted for as a separate mode with its own turn rate process noise. Lets consider an example trajectory of a flight, as shown in Figure 6,

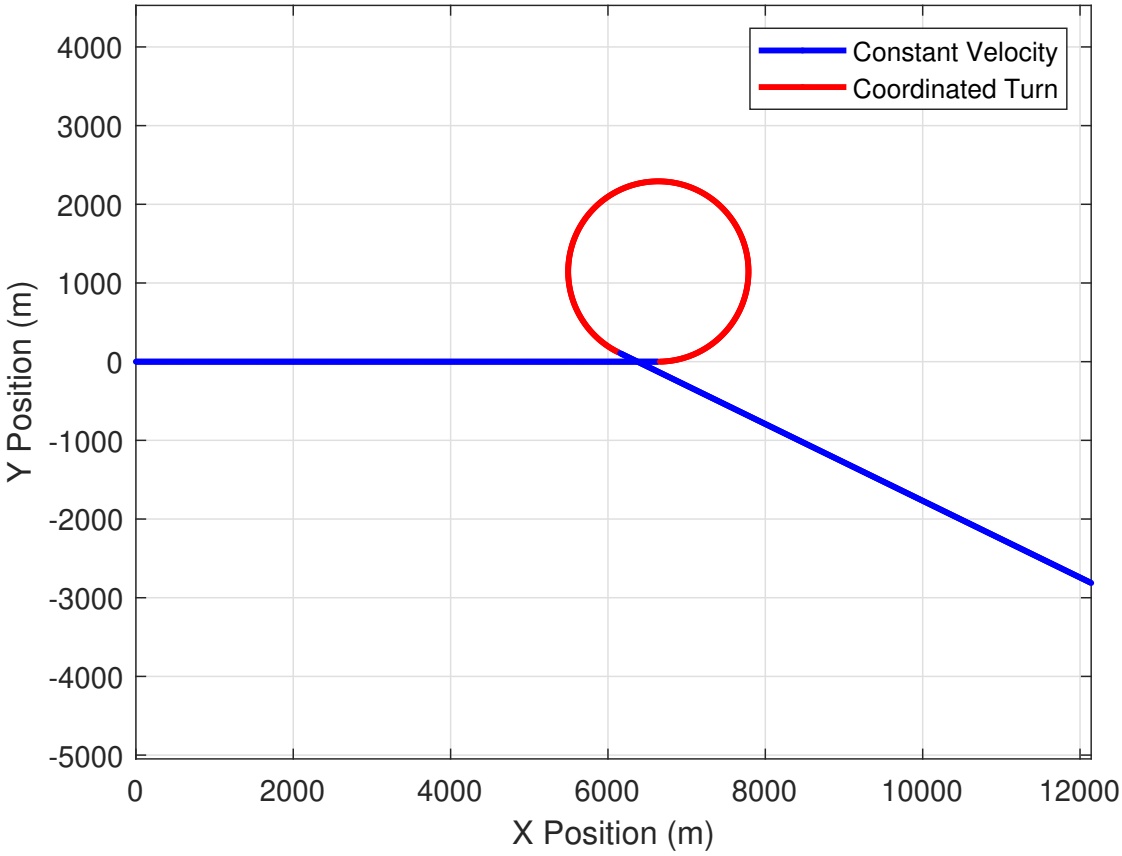

Figure 6.1: Example of a trajectory with Constant Velocity (CV) and Coordinated Turn (CT) modes.

that starts from $(0, 0)$ and moves along in x-axis. We know what mode we are in, KF knows the model and parameters for the constant velocity model and it optimally tracks the CV portion. Then we switch to CT or coordinated turn mode. Now, Kalman filter shifts to a new mode called the CT mode. Now, within the CT mode, KF knows the model and can optimally track the angular velocity and other states in this mode. Then, when we switch back to CV mode, KF needs to shift to CV mode again. Since, we do not know when the target is maneuvering, the performance of the KF degrades with a model-based approach.

We typically resort to heuristic methods such as an Interacting Multiple Model (IMM) which accounts for these modes and model switching and computes an average estimate iteratively using multiple filter models to account for varying target behavior. However, for more sophisticated targets, such as those guided by feedback control systems are more

difficult to model [39]. An IMM is prone to clutter, is not known to be generally optimal and does not offer any guarantees of robust performance.

Our primary goal in this chapter is to use machine learning to help solve this problem and overcome the limitations of the IMM. We approach this problem in two different ways.

1. *Temporal Convolutional Networks (TCN) based approach*: We propose a data-driven TCN [79] to predict dynamic random processes under model switching scenarios. TCNs use dilated convolutions to increase the receptive field of the network. They are more memory efficient than recurrent networks due to the shared convolution architecture which allows long sequences to be processes in parallel. In RNNs, the input sequences are processed sequentially, which results in higher computation time. However, TCNs are trained with the standard backpropagation algorithm, hence avoiding the gradient problems of the backpropagation-through-time algorithm used in RNNs. Here, we use a TCN with measurements as inputs and predicted states as outputs.
2. *Hybrid models such as Autoencoder Kalman Filter and Autoencoder Interacting Multiple Model*: The AEIMM places an Interacting Multiple Model Kalman Filter in the latent layer of a deep autoencoder. The IMM is similar to a standard Kalman Filter except now there is a bank of Kalman Filters, where the output of the IMM is a statistical weighting of the Kalman Filters. Similar to replacing a standard Kalman Filter with IMM for maneuvering target tracking, the transition from AEKF [4] to AEIMM is a natural extension. These models outperform model-based approaches like KF and IMM and learning based approaches like LSTMs.

6.1 Key Contributions

In this context, the contributions are:

1. We apply TCNs to predict states of dynamical systems with model switching.

We demonstrate that the TCN achieves a better mean-squared prediction error compared to classical algorithms such as IMM and least squares.

2. We start by developing an Autoencoder Interacting Multiple Model (AEIMM). We demonstrate that merging an autoencoder with IMM allows the AEIMM to be effectively used even in the context where the theoretical conditions for optimality of the Kalman Filter (KF) and IMM are not met. This is an extension of Autoencoder Kalman Filter (AEKF) [3]. This is further discussed in Section 6.4
3. We apply domain randomization for designing a robust learning model and avoid overfitting. This is discussed in Section 6.4.2.
4. Numerical demonstration of AEIMM outperforming model-based and learning-based approaches for the system model in Section 6.2. This is presented in the results in Section 6.4.5

6.2 System Model

We consider a framework for dynamical systems that undergo switching in time among several modes or sub-systems. Such dynamical systems have a long history in control theory, and are sometimes referred to as "jump systems" [80]. Under a fairly general discrete-time framework, the system state and measurement evolve according to

$$x[k+1] = f_{\theta_k}(k, x[k], v[k])$$
$$y[k] = h_{\theta_k}(k, x[k], w[k])$$

where $x[k]$ is the state vector, $y[k]$ is the measurement vector, $v[k]$ is the random process noise, $w[k]$ is the random measurement noise, $f_{\theta_k}(\cdot)$ is a family of N vector functions describing the state dynamics during mode θ_k, $h_{\theta_k}(\cdot)$ is a family of N vector functions describing the measurement dynamics, and $\theta_k \in \{0, 1, \ldots, N-1\}$ denotes the stochastic mode in effect during the sample period ending at discrete time k. We assume throughout

that the current mode θ_k in effect is not known, though the statistics of the random process may be known.

Under certain assumptions about the model dynamics and the stochastic switching, several important special cases of this system model emerge. When the switching process θ_k can be described by a Markov chain with time-invariant transition matrix P, so-called Markov Jump Systems arise [81]. If, in addition, the system is linear, the system is called a Markov Jump Linear System (MJLS) [44] and it admits the state space realization

$$x[k+1] = F_{\theta_k}[k]x[k] + v[k] \tag{6.1}$$

$$y[k] = H_{\theta_k}[k]x[k] + w[k] \tag{6.2}$$

where the non-linear functions $f_{\theta_k}(\cdot)$ and $h_{\theta_k}(\cdot)$ are replaced by the matrices $F_{\theta_k}[k]$ and $H_{\theta_k}[k]$. When the the process and measurement noises can be modeled as Gaussian random processes, we assume they are distributed as $v[k] \sim \mathcal{N}(0, \boldsymbol{Q})$ and $w[k] \sim \mathcal{N}(0, \boldsymbol{R})$, respectively, with $\boldsymbol{Q}$ and $\boldsymbol{R}$ as the corresponding covariance matrices. Again, while we assume that the mode θ_k in effect at time k is not known, at times we will assume knowledge of the transition matrix $\boldsymbol{P}$ that completely characterizes the statistics of the underlying Markov chain.

We consider a specific application within this class of switching systems in this chapter: tracking maneuvering targets that switch between a near constant velocity mode and a coordinated turn mode.

In maneuvering target tracking, the primary objective is using noisy measurements of a moving object (acquired, for example, via radar) to estimate or predict state trajectories. The body of literature concerned with dynamical models of target tracking is rich, and there are numerous models for describing the dynamics of target motion (see [82] for a comprehensive survey). Target motions generally fall into two classes: non-maneuvering and maneuvering. A non-maneuvering motion is uniform motion at a nearly constant velocity, whereas a maneuvering motion is most any other motion. Here, we consider

a popular discrete-time maneuvering target tracking model [39] that switches between $N = 2$ such modes. Specifically, we assume that the target is either in a CV mode or in a CT mode. Moreover, we again assume that the switching dynamics are described by a two-state Markov chain so that the mode switching model shown in Figure 7.1 applies to this application, as well. While the model for CV mode turns out to be linear, the CT mode obeys a nonlinear model. We now describe the dynamical model for each of the two modes for state dimension compatibility when switching between modes.

6.2.1 Constant Velocity Model

The model for CV mode has four states $x[k] = [\xi[k]\, \dot{\xi}[k]\, \eta[k]\, \dot{\eta}[k]]^\top$ with ξ and η denoting Cartesian coordinates in the horizontal plane, and $\dot{\xi}$ and $\dot{\eta}$ denoting the velocity. The model for operation in CT mode is linear and is described by

$$x[k] = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} x[k-1] + \begin{bmatrix} \frac{1}{2}T^2 & 0 \\ T & 0 \\ 0 & \frac{1}{2}T^2 \\ 0 & \Omega[k] \end{bmatrix} v[k] \tag{6.3}$$

with $v[k] \sim \mathcal{N}(0, \boldsymbol{Q})$ as in [39]. The process noise here serves to model turbulence or other non-idealities. Because the CT model below has five states, it is common to append a fifth state to this model that is always zero.

6.2.2 Coordinated Turn Model

The coordinated turn (CT) model has nearly constant speed and turns at a constant rate. It adds a fifth state to the CV model and is defined as $x[k] = [\xi[k]\, \dot{\xi}[k]\, \eta[k]\, \dot{\eta}[k]\, \Omega[k]]^\top$

where $\Omega[k]$ denotes the turn rate. The CT model is *nonlinear* and can be written as

$$x[k] = \begin{bmatrix} 1 & \frac{\sin(\Omega[k]T)}{\Omega[k]} & 0 & -\frac{1-\cos(\Omega[k]T)}{\Omega[k]} & 0 \\ 0 & \cos(\Omega[k]T) & 0 & -\sin(\Omega[k]T) & 0 \\ 0 & \frac{1-\cos(\Omega[k]T)}{\Omega[k]} & 1 & \frac{\sin(\Omega[k]T)}{\Omega[k]} & 0 \\ 0 & \sin(\Omega[k]T) & 0 & \cos(\Omega[k]T) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} x[k-1] + \begin{bmatrix} \frac{1}{2}T^2 & 0 & 0 \\ T & 0 & 0 \\ 0 & \frac{1}{2}T^2 & 0 \\ 0 & T & 0 \\ 0 & 0 & T \end{bmatrix} v[k]. \tag{6.4}$$

Because the matrix multiplying $x[k-1]$ in the above equation is a nonlinear function of one of the state variables, $\Omega[k]$, it is clear that this is a nonlinear model. Because the turn rate $\Omega[k]$ is a state variable, we assume that it is an unknown parameter to be estimated. In models where the turn rate is known, however, this CT model reduces to a linear system.

The model for the measurement $y[k]$ in both CT mode as well as the CV mode (with an appended fifth zero state, as mentioned above) is given by

$$y[k] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} x[k] + w[k]$$

where $w[k] \sim \mathcal{N}(0, \boldsymbol{R})$. That is, the measurement consists of noisy observations of the Cartesian coordinates of the target.

6.3 Machine Learning Algorithms

6.3.1 Temporal Convolutional Networks

CNN comprise of a sequence of convolutional layers that act as a series of filter banks learning their weights by computing the convolution between the sliding filter and the data, as the filter moves across the data. This helps to learn repeating patterns in data, which updates the weight matrix at each step. This also helps the network to improve local connectivity and learn long-term relationships in noisy data. They are especially advantageous in terms of weight sharing which reduces the number of connections and parameters in a network and the fact that each filter can look for a particular entity or pattern within the entire image which adds ability to translate about weights learned from one area of image to other areas.

While CNNs have shown great promise as an effective machine learning architecture, particularly in image recognition applications [83], they have been recently used as *temporal convolutional networks* [79] for time-series modeling and sequence prediction problems. TCNs have demonstrated state-of-the-art performance over a wide range of prediction applications, including weather prediction [84], traffic prediction [85], audio [86], and action segmentation [87].

TCNs are generally used in one of two configurations: (i) for sequence estimation where the output sequence is the same length as the input sequence, just as with RNNs, or (ii) for autoregressive prediction of samples at some time in the future, which is typically accomplished by adding a fully connected layer to the output of the sequence prediction. We consider the latter configuration and the corresponding architecture is shown in Figure 6.2.

Essentially, a TCN employs a series of connected residual blocks consisting of 1D fully-convolutional networks (FCNs) [88] where the convolutions are chosen to be causal, primarily through appropriate choice of zero-padding, and dilation [87]. In TCNs, di-

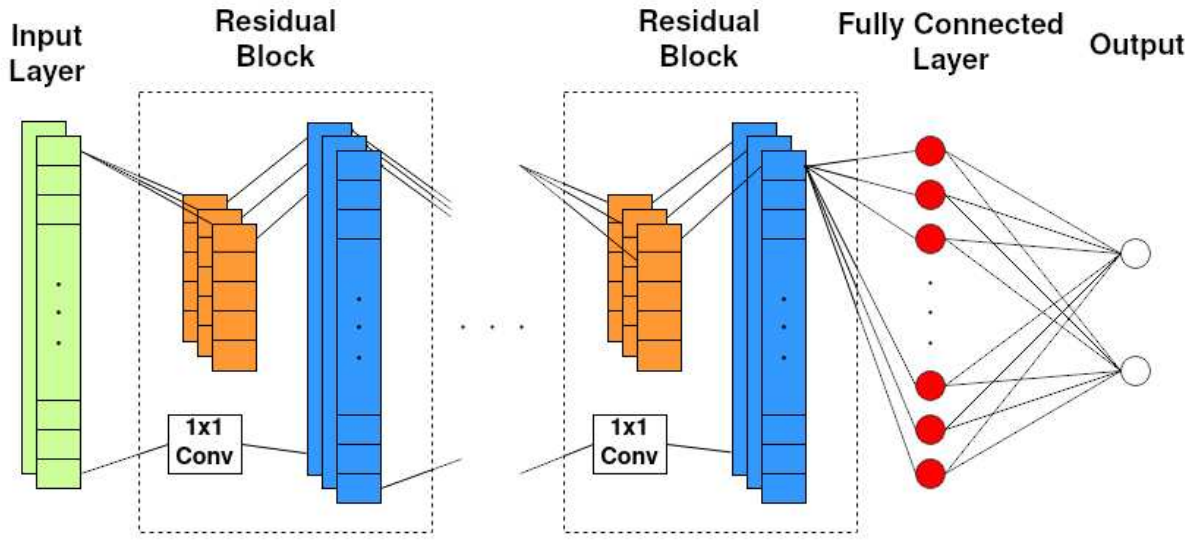

Figure 6.2: Schematic of a Temporal Convolutional Network with a series of residual blocks with increasing dilation followed by a fully connected layer.

lation takes the place of "pooling" which is commonly used in more traditional CNN architectures, and it allows the TCN to have a large receptive field.

A TCN can accept a multi-dimensional input sequence (i.e. a matrix). In the two applications considered in Chapter 7 – Gilbert-Elliott channel prediction and maneuvering target tracking – the input vectors are two-dimensional as shown in Figure 6.2. The specific TCN implementation that we consider is the implementation described in [79].

6.3.2 Test Protocol

For our simulations, Mode 0 was chosen to be the CV model described by (6.3), while Mode 1 was chosen to be the CT model described by (6.4). The CV-to-CT and CT-to-CV transition probabilities were selected as $p_{01} = p_{10} = 0.01$, and the remaining system parameters were selected to be $T = 1$ sec, $Q = I_2$, $R = 576 \cdot I_2$ where I_2 is a 2×2 identity matrix. The initial speed of the target was normally distributed with a mean of 120 m/s and a standard deviation of 30 m/s. The magnitude of the initial turn rate at the start of each turn was normally distributed with a mean of 4 rad/sec and a standard deviation of 1 rad/sec, and with left and right turns being equally likely; thus, $\Omega[k]$ at the start of each

turn has a folded Gaussian distribution. We perform 4×10^4 Monte-Carlo simulations with 10^3 samples to ensure a consistent performance across multiple mode transitions.

The network is trained on 4×10^6 samples with 774,302 parameters comprised of 1,302 bias terms and 773,000 network weights. Additional details concerning other TCN hyperparameters can be found in the code repository [89]. For each prediction, the TCN was provided the two Cartesian coordinates of the 20 most recent noisy observations of the target, and thus the input was from $\mathbb{R}^{20\times 2}$. Just prior to the TCN input, the coordinate system of every input was translated so that each length-20 input sequence terminated at the origin; at the TCN output, this translation was reversed to put the outputs back on the original coordinate system. The reason for this translation is that data pre-processing has been shown to improve the performance of machine learning systems, particularly when the pre-processing serves to limit the inputs and outputs to a narrower or more balanced range of values.

6.3.3 Results

We now present the simulated performance of the TCN when used for prediction. The performance of the TCN is compared to the IMM, LS, and a "Genie Kalman Filter" (GKF) which is a time-varying KF with perfect knowledge of the current mode θ_k. When the current mode θ_k is known and the mode dynamics are linear, the system reduces to a standard linear time-varying dynamical model for which the KF is optimal. Hence, for linear mode dynamics, the MSE of the Genie KF establishes a lower bound by which other algorithms can be compared. We report the average prediction RMSE in Table 6.1, where the RMSE is computed only over the two Cartesian coordinate states of the target as

$$\text{RMSE} = \sqrt{\mathbb{E}[(\xi - \hat{\xi})^2 + (\eta - \hat{\eta})^2]}$$

The performance of the algorithm is evaluated similar to the Gilbert-Elliott scenario. Any samples occurring 25 samples or fewer after a mode transition are defined as as being in a "transition regime". The CV and CT mode performance in the table indicates an average RMSE only over those time slots which are not in a transition regime, i.e., where no transition occurred within the previous 25 time slots.

Here, however, the GKF is additionally provided knowledge of the current turn rate $\Omega[k]$ (i.e., the fifth element in the state vector). With this knowledge, the CT model in (6.4) becomes a time-varying *linear* model and thus the GKF is the optimal estimator. We note that providing this additional information to the GKF may result in its prediction MSE being a rather loose lower bound for CT mode operation.

Table 6.1: Maneuvering target prediction RMSE (in meters).

	With mode switching	**Only mode** 0 **(CV)**	**Only mode** 1 **(CT)**
Genie KF	**19.5**	**19.4**	**19.6**
IMM	28.0	**20.0**	**24.8**
TCN	**25.4**	20.6	25.9

The results in Table 6.1 show that the prediction RMSE for the GKF, IMM, and TCN are all quite close during CV mode, suggesting that both schemes are achieving near-optimal RMSE performance. During CT mode, the GKF lower bound is likely rather loose, as knowledge of the true turn rate provides it a considerable advantage during this mode; however, the prediction RMSEs for IMM and TCN are again rather comparable during this mode, with IMM prediction RMSE being just slightly lower than the TCN. As for the case with switching, overall the TCN has lower prediction RMSE than the IMM. This is largely due to the superior RMSE performance of the TCN during CV-to-CT transitions, as we will now show.

Figures 6.3 and 6.4 depict the prediction RMSE values before and during the transitions from time steps 0 through 25. While the TCN exhibits a slightly higher RMSE than the IMM during the transition from CT mode to CV mode in Figure 6.3, the TCN

has significantly lower RMSE when transitioning from CV mode to CT mode as shown in Figure 6.4. For the chosen transition probabilities and with the definition of a mode "transition" as lasting 25 samples, the target spent 38.5% of its time in the CV mode,

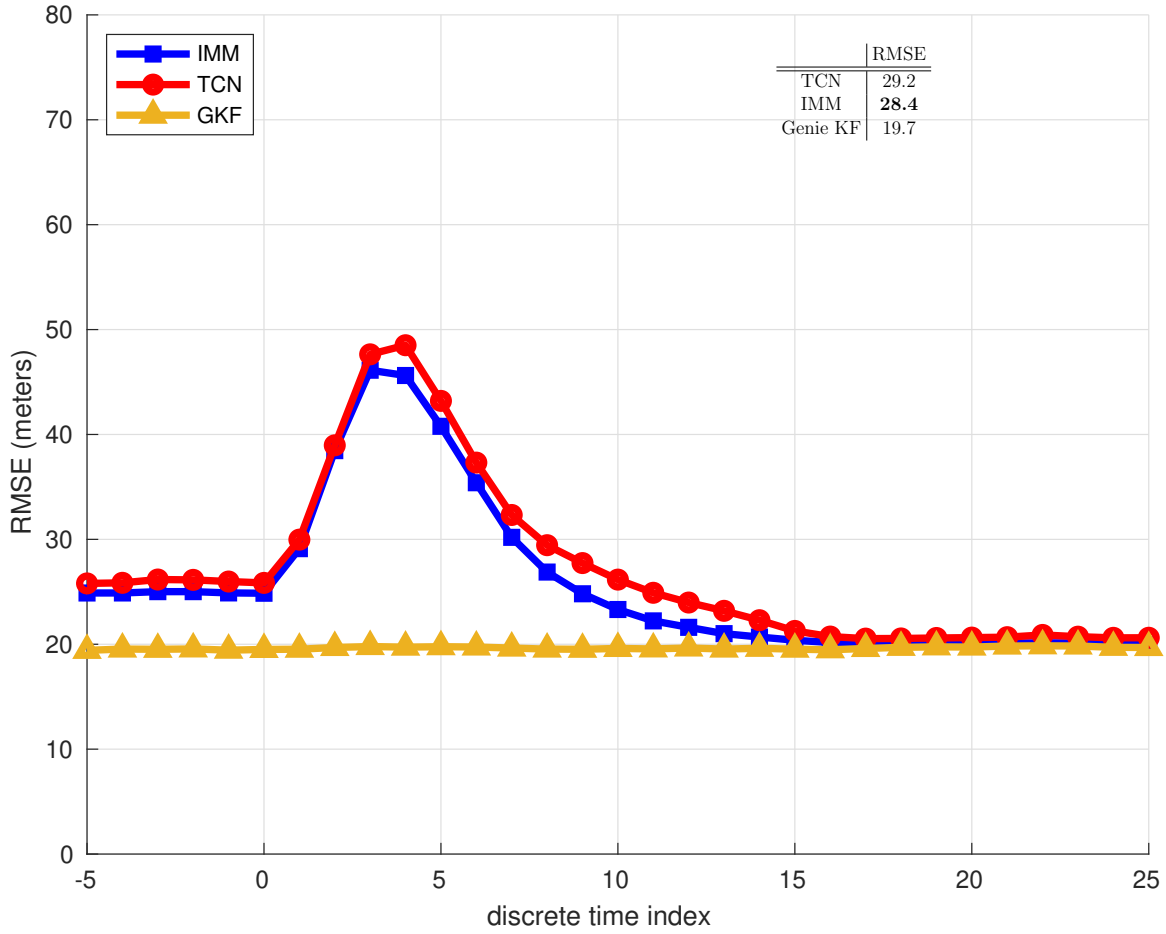

Figure 6.3: Root Mean-Square Prediction Error During CT-to-CV transitions. RMSE values in table computed over discrete time values $0 \leq k \leq 25$.

In summary, while the TCN is actually slightly inferior to the IMM in terms of prediction RMSE during long stretches of CV and/or CT modes as well as during transitions from CT-to-CV mode, the ability of the TCN-based approach to more quickly recognize that the target has entered CT mode leads, overall, to a performance advantage over the IMM.

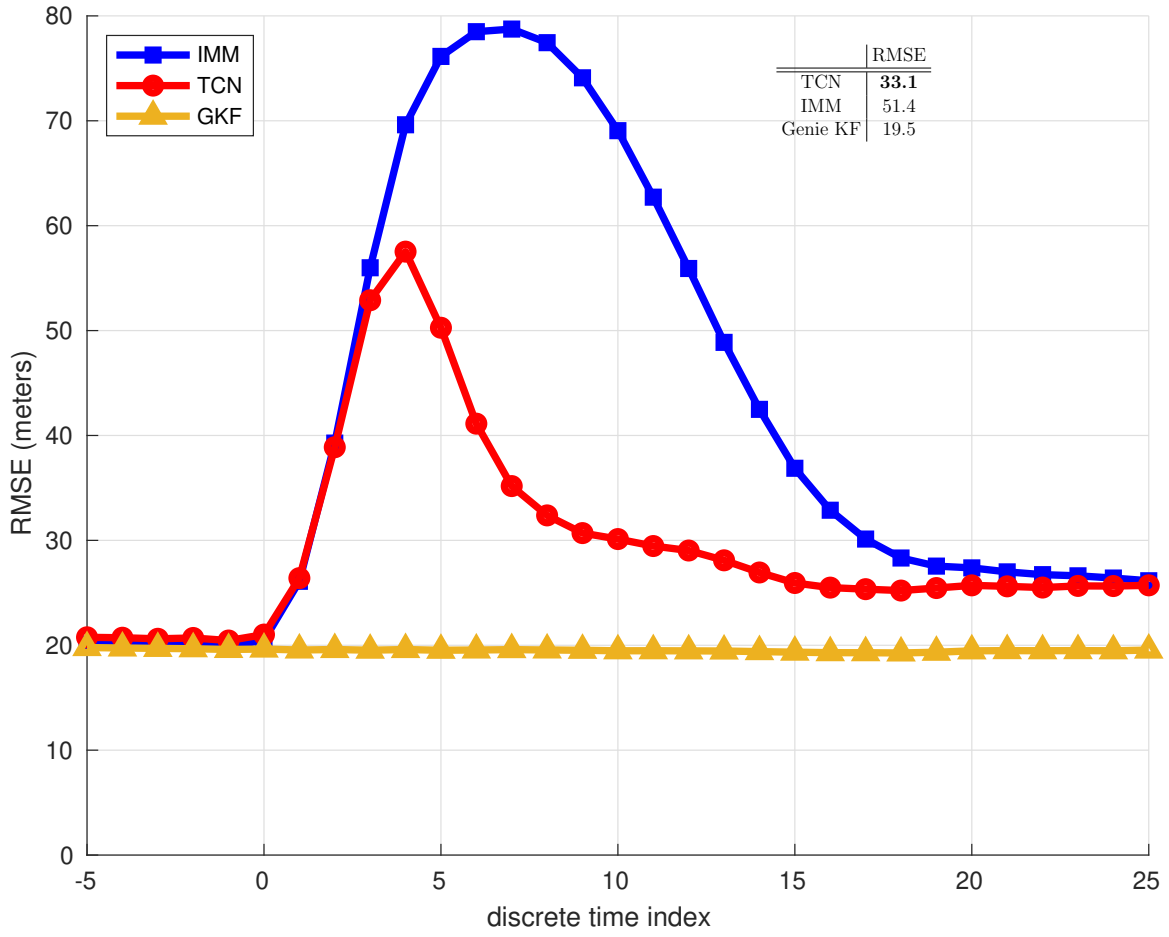

Figure 6.4: Root Mean-Square Prediction Error During CV-to-CT transitions. RMSE values in table computed over discrete time values $0 \leq k \leq 25$.

6.4 Hybrid Algorithms

We now consider hybrid algorithms where we combine deep learning methods with traditional methods. When combined with well-known mathematical models such as the KF or IMM, certain design and training issues can be addressed in the Kalman Filter itself, where the decision was informed by the Kalman Filter's well-known theoretical basis. At the same time, the computational power of deep learning allows a hybrid model such as the AEKF to outperform a traditional Kalman Filter [3].

Autoencoders are a class of unsupervised deep learning algorithms often used for dimensionality reduction. They have been successfully applied for end-to-end communication system design in channels with Gaussian and non-Gaussian noise [59] and for object tracking applications [90]. The combination of autoencoders with AEKF/AEIMM merges the computational power of deep learning with the theoretical understanding of a simple and elegant linear system. Neural networks lack mathematical principles that could be useful in guiding training. We first consider AEKF and then present AEIMM as an extension of AEKF.

6.4.1 Autoencoder Kalman Filter

The AEKF [3], [4] is a hybrid autoencoder KF algorithm that places a KF in the latent layer of a traditional autoencoder, as depicted in Figure 6.5.

Here the measurements, $\boldsymbol{\phi}$, are first transformed by the encoder portion of an autoencoder, where each layer of the autoencoder is represented by $\mathcal{E}_l$. The encoder outputs two sequences, $\mathbf{z}[k]$ and $\mathbf{R}[k]$, which are passed to a KF as measurements and associated measurement covariances. The KF's state estimate of $\mathbf{z}[k]$, represented by $\hat{\mathbf{z}}[k]$, is then mapped back to the measurement space via the decoder portion, whose layers are similarly represented by $\mathcal{D}_l$. The final output of the decoder is the state estimate of the original measurements, $\hat{\boldsymbol{\phi}}$. Note the vertical ellipses represent the number of dimensions in a layer and horizontal ellipses represent the depth of the encoder and decoder.

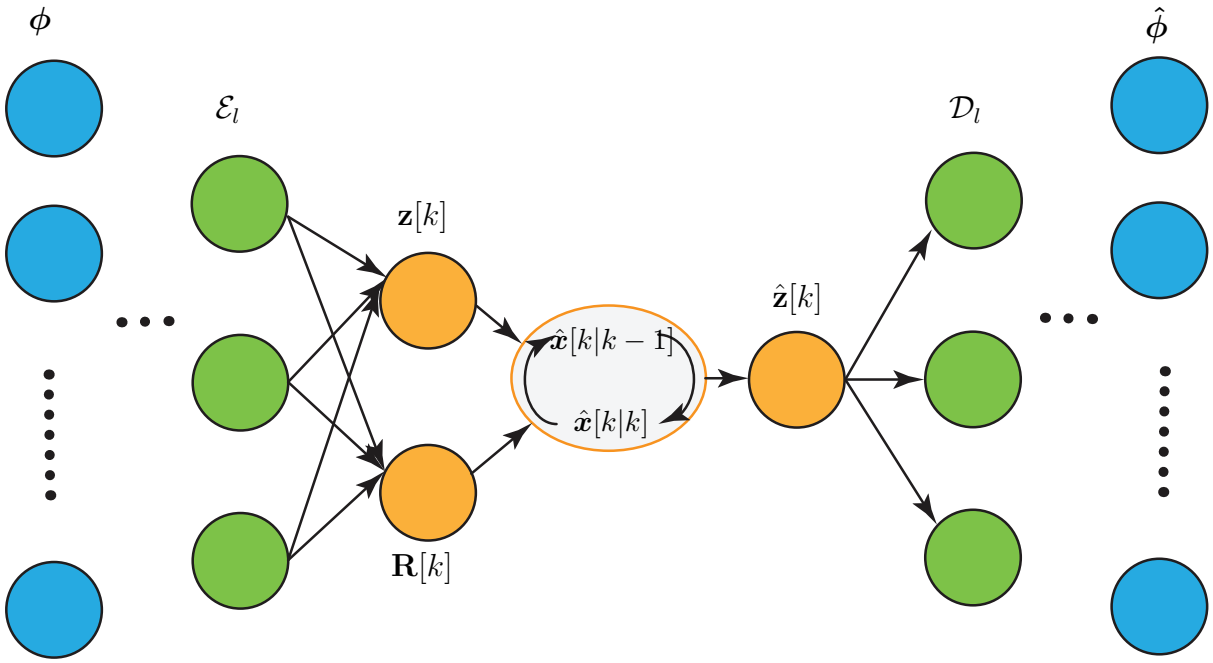

Figure 6.5: The Autoencoder Kalman Filter (AEKF).

A detailed description of the AEKF, along with a comparison to other deep learning-KF hybrid models, appears in [3], [4]. Here we present a brief overview and some details on the training of the AEKF. As shown in Figure 6.5, the actual measurements are now represented by $\boldsymbol{\phi}$, which is mapped via a composition of neural network layers to two new variables: $\mathbf{z}[k]$ and $\mathbf{R}[k]$. These comprise the input measurements and associated measurement covariance to the KF. The output of the KF, $\hat{\mathbf{z}}[k]$, is then mapped back to the measurement space via the decoder portion of the AEKF. Here the output, $\hat{\boldsymbol{\phi}}$, is a filtered version of the original input $\boldsymbol{\phi}$.

Since the encoder and decoder portions of the AEKF are trained via back propagation, the cost function should compare $\boldsymbol{\phi}$ and $\hat{\boldsymbol{\phi}}$ in some way. However, since the actual ground truth is not known, we train with simulated data where the ground truth is known. In the context of deep learning, this technique is known as domain randomization [3], [4], [91].

In the context of domain randomization, the loss function for this architecture is

$$\mathcal{L} = \min_{\boldsymbol{\theta}} \sum_{i} \|\boldsymbol{\phi}_i - \hat{\boldsymbol{\phi}}_i(\hat{\boldsymbol{\phi}}_{i-1}, \hat{\boldsymbol{\phi}}_{i-2}, \dots)\|_F^2 \tag{6.5}$$

where $\boldsymbol{\theta}$ represents the parameters learned by encoder and decoder.

We propose to leverage domain randomization as done in [3], [4] which allows training over a range of parameter values in simulation, thus overcoming the need to have knowledge of the real-data ground truth to train a deep learning model.

6.4.2 Domain Randomization

Domain randomization is an *a priori* approach in which the parameters of the dynamics and/or observations are randomized during training, but the final policy does not explicitly perform system identification at test time. Randomization of dynamics parameters in simulation has been shown to increase generalization ability [92], [93]. Random textures and colors in the visual input have been used for successful sim-to-real transfer for indoor navigation [94] and simple manipulation tasks [95], [96].

The main philosophy behind domain randomization is that if there is enough variability in the simulated model, the real world is just one among the many variations learned in simulation [3]. So, the goal is to achieve a high enough degree of variability in simulation. This makes sure that the real world being modeled is present among the variations. This is possible when the range of parameter randomization is bounded and informed periodically by domain knowledge. By randomizing training parameters, the learned model will be robust enough to perform well on test cases whose parameters fall within the range of the randomized training parameters.

Figure 6.4.2 shows an example of a simulated training curve with bimodal noise. The smooth line is the ground truth Taylor Polynomial ϕ_{true} with the points representing the ground truth with added noise ϕ_{noise}. At each training epoch a new simulated curve was generated and passed to the AEKF. As a result, the AEKF effectively never saw the

same curve twice during training. Our use of domain randomization can be thought of as training a neural network to perform state estimation for increasingly more general families of functions in Hilbert Space. It covers a particular parameter space by randomizing a parameter over its entire domain. Since the data is simulated, access to the ground truth for training is not problematic.

Domain randomization is appealing in its simplicity, and has performed well in the literature and in our own experiments. We train deep learning models completely in simulation which then generalize to real-world data without any further training or parameter tuning. Instead of training a neural network to learn parameters from a fixed training set and then generalize to a testing set, we utilize domain randomization to train a neural network to learn parameters within a specified range of values in simulation. In some sense, this process can be thought of as learning a covering of a subspace in a Hilbert Space. However, it implicitly assumes that, given the observed state of the system, there exists an action that will produce acceptable behavior over all possible values of the unknown system identification parameters. This assumption may not hold if the set of possible test environments is diverse.

6.4.3 Autoencoder Interacting Multiple Model

The AEIMM is conceptually similar to the AEKF in that the KF portion of the AEKF is replaced by an IMM, as shown in Figure 6.7. Both the AEKF and AEIMM address the issue of estimating measurement covariances. However, in the case of the AEIMM, we allow the encoder portion to learn different measurements and their associated measurement covariances *for each of the two KFs in the AEIMM*, represented by $\mathbf{z}_A, \mathbf{z}_B$ and $\mathbf{R}_A, \mathbf{R}_B$ respectively.

The primary motivation for this feature is that learning different measurements and associated measurement covariances for each KF in the IMM will assist the MDP mixer in weighting the appropriate KF. This, in turn, will help the IMM weight the system

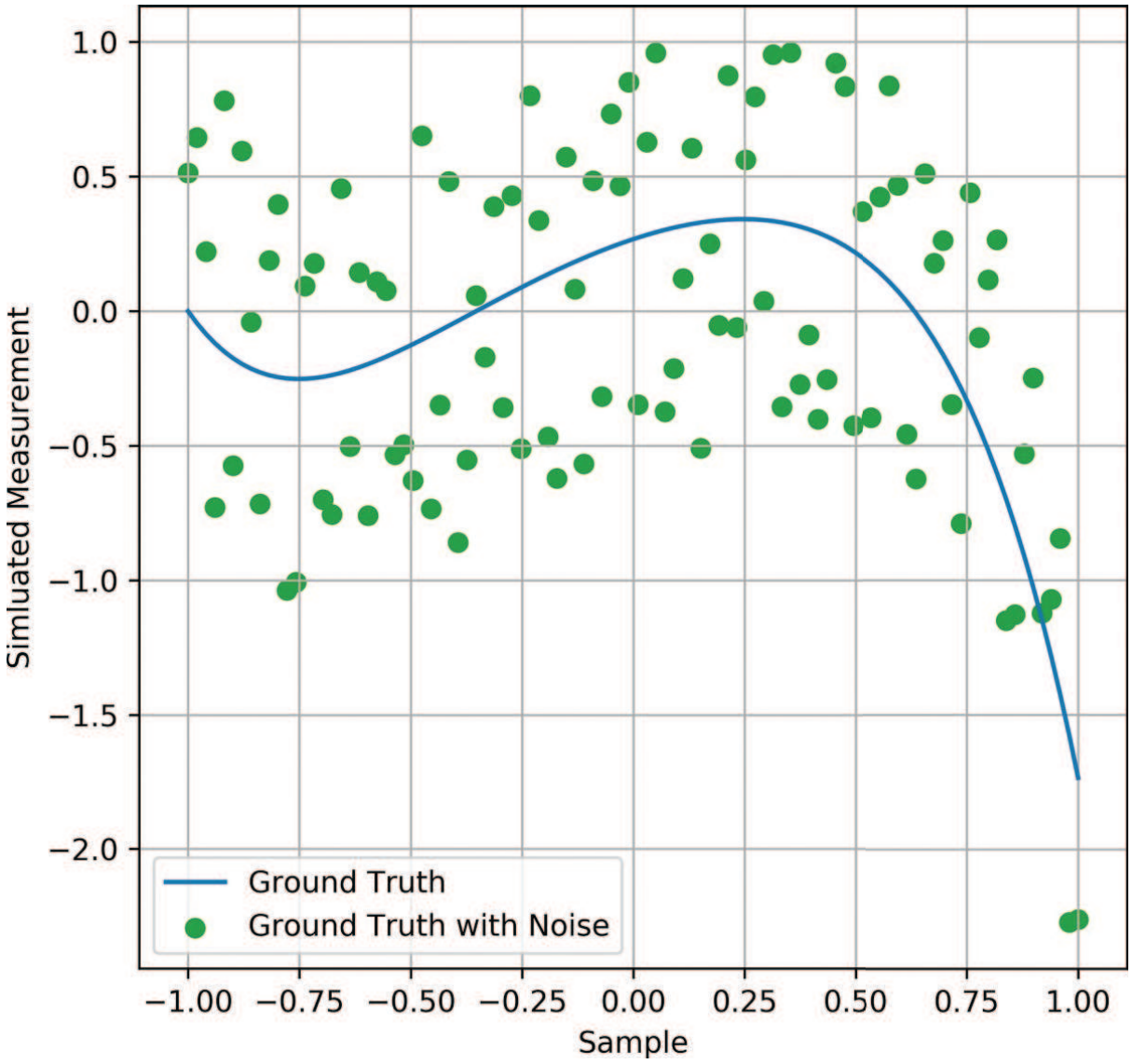

Figure 6.6: Domain randomization for polynomials [3].

dynamics appropriately. The multiple encoder outputs are passed to two KFs: KF-CV and KF-CT, which output $\hat{x}_A, \hat{\Sigma}_A$ and $\hat{x}_B, \hat{\Sigma}_B$ respectively.

These are then passed into a Markov Decision Process (MDP) mixer that returns $\hat{x}_M, \hat{\Sigma}_M$. Lastly, $\hat{x}_M, \hat{\Sigma}_M$ are

1. passed back to the IMM for the next iteration
2. passed to the decoder, which maps these values to the final output $\hat{\boldsymbol{\phi}}$.

We consider two neural networks here: a neural network that provides noisy states z with varied noise covariance matrix $\boldsymbol{R}$ to Kalman filters KF-CV and KF-CT and another neural network at the decoder. We have the outputs from the state space model $\boldsymbol{\phi}$ being sent into the encoder part of the autoencoder. The encoder connects to two Kalman filters KF-CV and KF-CT that give out $\hat{x}_A, \hat{\Sigma}_A$ and $\hat{x}_B, \hat{\Sigma}_B$ respectively. These will go into a MDP mixer that gives out $\hat{x}_M, \hat{\Sigma}_M$. These go into the decoder part of the autoencoder, another neural network that takes in $\hat{x}_M, \hat{\Sigma}_M$ and gives the outputs $\hat{\boldsymbol{\phi}}$. These will be sent back to KF-CV and KF-CT at each iteration as a feedback.

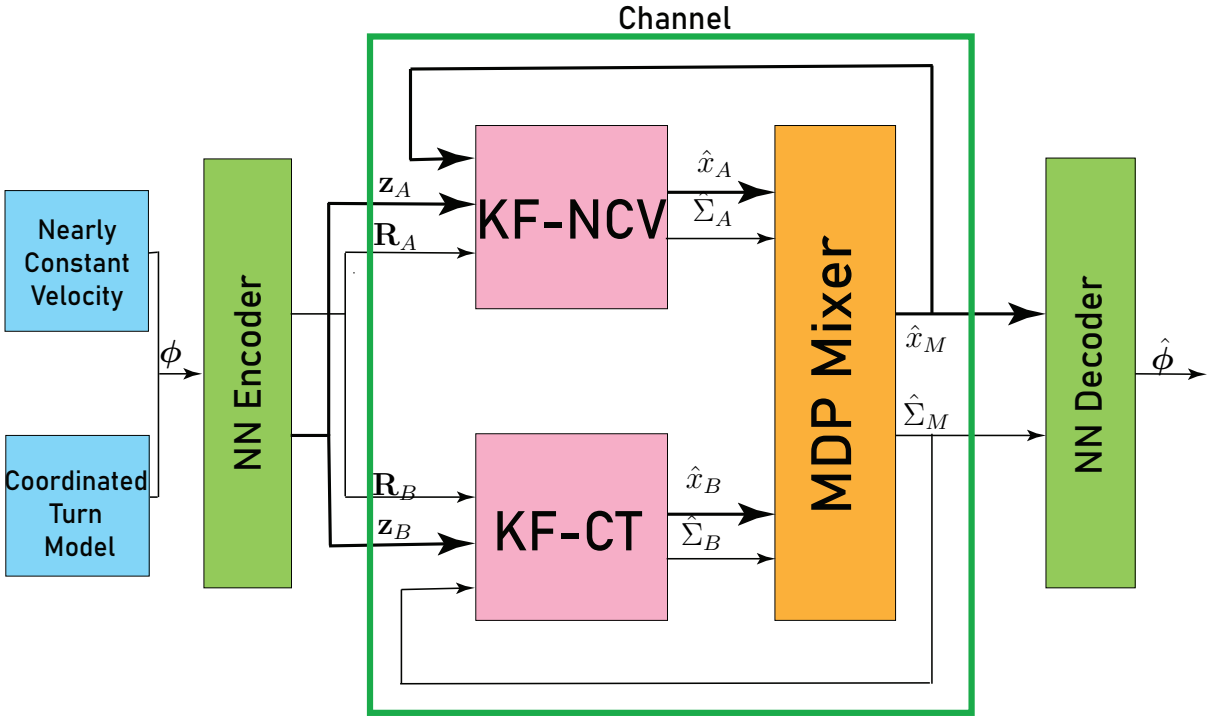

Figure 6.7: Block diagram for the Autoencoder Interacting Multiple Model (AEIMM) filter.

We can use the same loss functions that we used in AEKF calculations. From an autoencoder perspective, we can view the IMM portion as a channel with continuous-time and discrete-time dynamics. Physically, the encoder portion of the neural network takes the real space and makes it easier for the IMM in the middle to have better priors. The decoder portion of the neural network makes sure the results are generalized to the test sets well.

6.4.4 Test Protocol

We test our approach on simulated flight paths consisting of constant velocity segments interspersed with coordinated turns. They are based on physical models according to the kinematics equations for constant linear motion and coordinated turns. A sample flight path with Gaussian noise is shown in Figure 6.8.

All simulated flight paths begin with constant velocity motion in the horizontal direction. At each turn, the corresponding turn radii is chosen randomly (within a predefined range), along with the turn direction (clockwise or counter clockwise). Gaussian and non-Gaussian noises are then added to these smooth ground truth flight paths. Using these noisy simulated flight paths, we compare the state estimation capabilities of the following models: KF, IMM, AEKF, AEIMM and LSTM.

Each model's state estimation is then compared with the actual ground truth and the corresponding MSE is reported for each model. In this phase, the ground truth is used *only for model evaluation* and does not affect each model's state estimate in any way. As the state estimate and ground truth are two dimensional, the MSE is calculated by taking the Frobenius norm between the state estimate and ground truth.

While we use simulated flight paths to demonstrate the efficacy of the AEIMM, it should be noted the AEIMM is designed in a general manner to make it applicable to any application or scenario where an IMM is appropriate.

We also compare our approach with Long-Short Term Memory (LSTM) network here.

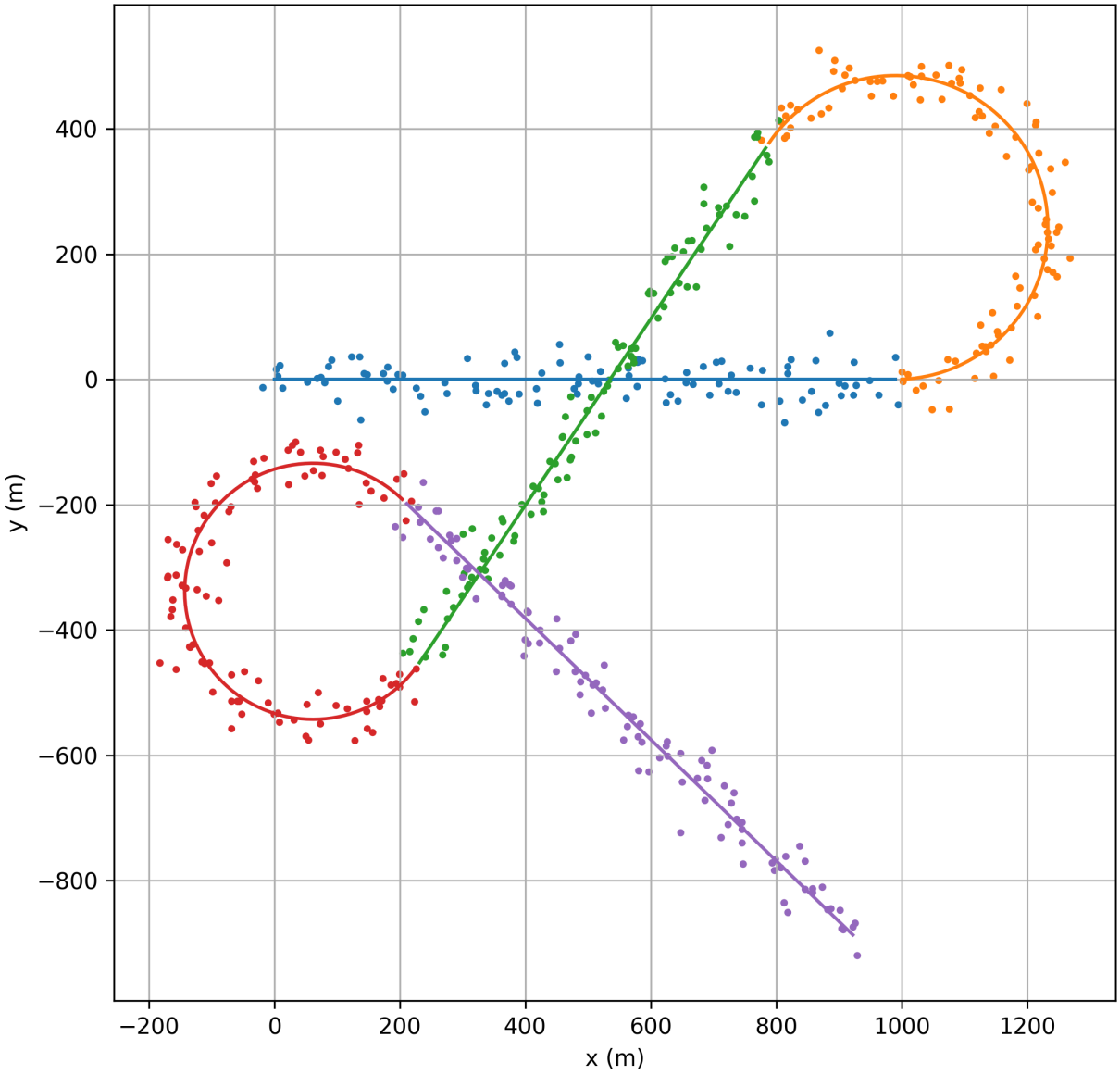

Figure 6.8: Sample simulated two-turn flight path with Gaussian noise.

An LSTM is a type of recurrent neural network (RNN) initially developed to solve the vanishing gradient problem in RNNs. As they are able to learn long term temporal dependencies, LSTMs are well suited for time series classification, prediction and state estimation. They make it easier to model time series problems and learn non-linear dependencies among multiple inputs, and we do not make certain assumptions that are made in classical approaches.

Here we present results for single turn flight paths with an initial velocity in the horizontal direction of 100 m/s, a turn radii uniformly selected between 200 and 300 meters and added Gaussian noise $\mathcal{N}(0, 20)$. Each of the three flight segments lasts 100 seconds with a sampling frequency of 10 Hz. The transition probabilities for models involving the IMM are 0.9 and 0.1.

We compare five models: KF, IMM, LSTM, AEKF and AEIM where

1. the KF and AEKF models consist of CV models with process noise covariance given by $\boldsymbol{Q} = 0.5\boldsymbol{I}$.
2. the IMM and AEIMM consist of CV and CT models, both with $vecQ = 0.5\boldsymbol{I}$.

The test set consists of 1000 simulated single turn flight paths. For each model, the reported RMSE is computed by averaging the RMSE on each of the 1000 test paths computed using the ground truth and state estimates. Results are shown in Table 6.2, where the MSE ratio is the ratio of each model's MSE to the KF's MSE. Here the models, from highest to lowest MSE, are LSTM, KF, IMM, AEKF and AEIMM.

6.4.5 Results

For visualization, an example of one test trial with the ground truth, noisy simulated measurements and state estimate is shown in Figure 6.9. For 1000 such simulations, the average MSE values are shown in Table 6.2.

The state estimation of each model is then compared with the actual ground truth

Table 6.2: Single turn test MSE results.

Model	MSE	MSE Ratio
KF	98.44	1.00
IMM	70.45	0.72
LSTM	357	3.63
AEKF	62.17	0.63
AEIMM	50.48	0.51

and the corresponding MSE is reported for each model. Note that in this phase, the ground truth is used only for model evaluation and does not affect each model's state estimate in any way.

The fact that the IMM shows better performance than the KF is not surprising since IMM operates on a bank of KFs instead of a single KF. However, both these models show improvement when combined with a neural network. We can see that against KF, we have AEKF and AEIMM performing better than their counterparts KF and IMM. We can also notice that the position MSE for the hybrid models is much smoother than the traditional or the pure ML models. The KF and IMM have more difficulty estimating the ground truth on the turn than the AEKF and AEIMM.

Finally, we notice that LSTM is not performing well. We theorize that the LSTM has trouble on the turns because it is simply looking at history and doesn't have a dynamical model based on physics and Taylor expansion like the KF does. Furthermore, in the KF, the contributions of the a priori estimate and the measurement innovation to the a posteriori estimate are weighted by the Kalman Gain which might be contributing towards its better performance.

6.5 Conclusion

The conclusions for this chapter can be divided into two categories:

1. *Machine Learning Approach*: We applied TCNs to predict states of dynamical systems with model switching. We demonstrated that the TCN achieves a better

mean-squared prediction error compared to classical algorithms such as IMM and least squares. While the TCN is actually slightly inferior to the IMM in terms of prediction RMSE during long stretches of CV and/or CT modes as well as during transitions from CT-to-CV mode, the ability of the TCN-based approach to more quickly recognize that the target has entered CT mode leads, overall, to a performance advantage over the IMM.

2. *Hybrid Algorithms*: We combine the computational power of deep learning with a theoretically well-established filtering method such as a Kalman filter or an IMM. We extend the concept of AEKF in this paper and develop a hybrid model for tracking maneuvering targets. Training the AEIMM on simulated single-turn flight paths with added Gaussian noise, we show the AEIMM achieves superior state estimation compared with a standard Kalman Filter, IMMKF, and AEKF. While we use simulated flight paths to demonstrate the efficacy of the AEIMM, it should be noted the AEIMM is designed in a general manner to make it applicable to any application or scenario where an IMM is appropriate e.g. nonlinear target dynamics, sudden starts/stops of maneuvers.

In this chapter, we restrict to single turns of maneuvering targets. For future research, it would be interesting to see the performance of TCN and AEIMM on simulated curves with multiple turns. Possible future directions include investigating the use of TCNs in other dynamical systems with model switching, such as automotive traffic modeling, power plant control, wireless energy transfer, and scheduling information transfer in communications channels. Because research on TCNs more broadly is advancing at a fast pace, further research could also include recently enhancements to the TCN architecture and training approach.

It would be interesting to perform a rigorous theoretical analysis of AEIMM and hybrid algorithms in general and bounds on their optimality and limitations. This would be help answer a few questions such as:

1. If we replace MDP with a neural network and keep the KFs intact, does the neural

network mix or augment the outputs from the two KFs?

2. Is it better to use two neural networks on the encoder side, one for each dynamic model that output $\mathbf{z}_A, \mathbf{R}_A$ and $\mathbf{z}_B, \mathbf{R}_B$ respectively instead of a single encoder?

Finally, a natural extension to this work would be to compare hybrid approaches against a TCN and see how they would compare and contrast against each other.

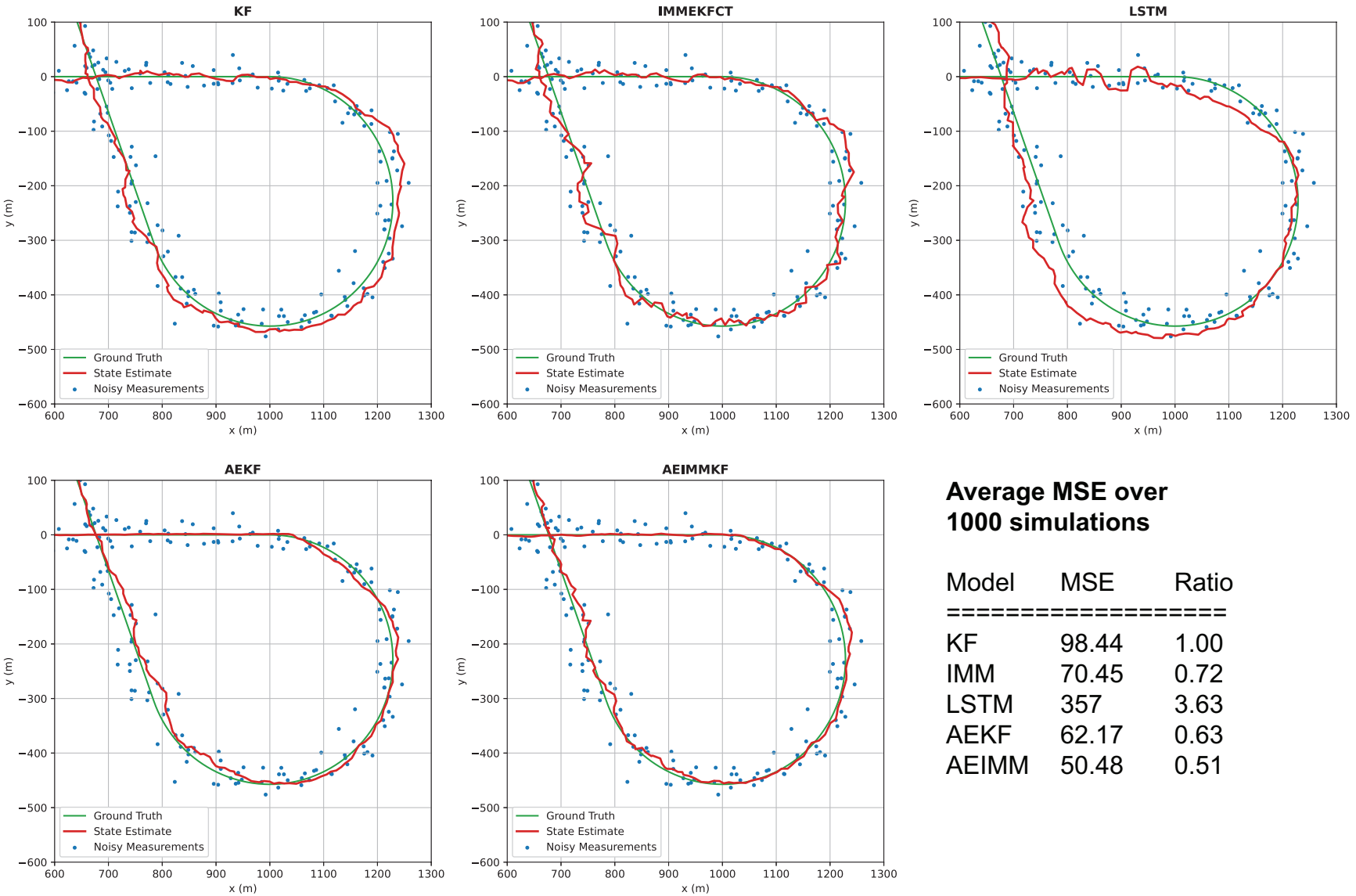

Figure 6.9: Turn segment from a Gaussian noise test set sample trial. Here the (a) Kalman Filter, (b) IMMKF, and (c) LSTM estimates have large MSE values and generally, are less smooth than the (d) AEKF and (e) AEIMM estimates.

Chapter 7

Tracking Dynamical Processes on Gilbert-Elliott Channels

(Co-written with Arick Grootveld, Leah Lackey, Vlad I. Bugayev, Andrew G. Klein, Department of Engineering and Design, Western Washington University, Bellingham, WA 98225)

Channels governed by Markov chains have been studied in the context of communication channels [97], [98]. A Gilbert-Elliott channel model is useful for simulating burst noise channels [45], [99] typically prevalent in Wireless Local Area Network specifications [100]. Similarly, target tracking is used in many practical applications to accurately track objects with trajectories that have significant position derivatives of several orders. When not detected and compensated, the maneuvering target can degrade the performance of the tracker and might lead to filter divergence. Maneuvering target tracking estimates or predicts aircraft motion, often by a radar or other detection and ranging sensor to control its flight path, or direct its movement in accordance with other operations. For a comprehensive survey of maneuvering target models, we point the reader to [82].

In practical communication systems, it is often common to have burst noise on the channel that makes communication using non-adaptive methods ineffective. A simple

but effective model for simulating burst noise in a channel is the Gilbert-Elliott two state Markov Model [45], which uses a Markov Chain with two states, a good channel state and a bad channel state - alternatively, mode 0 and mode 1. During the models' run time, there is a possibility to transition between the states, depending on the current state of the model.

We propose a data-driven temporal convolutional network (TCN)-based approach [79] to predict dynamic random processes under model switching scenarios. TCNs use dilated convolutions to increase the receptive field of the network. They are more memory efficient than recurrent networks due to the shared convolution architecture which allows long sequences to be processes in parallel. In RNNs, the input sequences are processed sequentially, which results in higher computation time. However, TCNs are trained with the standard backpropagation algorithm, hence avoiding the gradient problems of the backpropagation-through-time algorithm used in RNNs. Here, we use a TCN with measurements as inputs and predicted states as outputs.

The trained TCN model generalizes well over a large number of test samples. We demonstrate that our method, while suboptimal, outperforms other popular suboptimal state predictors. Furthermore, we use steady-state analysis as a reference for the optimal behavior of an omniscient Kalman filter and to compare our results. The proposed approach is sufficiently general that it can be applied to a wide variety of dynamical systems with model switching.

7.1 Key Contributions

In this context, the contributions are:

1. We apply TCNs to predict states of dynamical systems with model switching. We demonstrate that the TCN achieves a better mean-squared prediction error compared to classical algorithms such as IMM and least squares.
2. Numerical demonstration of TCN outperforming both model-based and learning-

based approaches for the system model in Section 7.2. This is presented in the results in Section 7.3

7.2 System Model

Similarly to Chapter 6, we consider a general framework for dynamical systems for model switching with the system state and measurement evolving according to

$$x[k+1] = \boldsymbol{f}_{\theta_k}(k, x[k], v[k])$$
$$y[k] = \boldsymbol{h}_{\theta_k}(k, x[k], w[k])$$

where $x[k]$ is the state vector, $y[k]$ is the measurement vector, $v[k]$ is the random process noise, $w[k]$ is the random measurement noise, $\boldsymbol{f}_{\theta_k}(\cdot)$ is a family of N vector functions describing the state dynamics during mode θ_k, $\boldsymbol{h}_{\theta_k}(\cdot)$ is a family of N vector functions describing the measurement dynamics, and $\theta_k \in \{0, 1, \ldots, N-1\}$ denotes the stochastic mode in effect during the sample period ending at discrete time k.

Similar to Chapter 6, we consider a Markov Jump Linear System (MJLS) [44] and it admits the state space realization

$$x[k+1] = \boldsymbol{F}_{\theta_k}[k]x[k] + v[k] \tag{7.1}$$
$$y[k] = \boldsymbol{H}_{\theta_k}[k]x[k] + w[k] \tag{7.2}$$

where the non-linear functions $\boldsymbol{f}_{\theta_k}(\cdot)$ and $\boldsymbol{h}_{\theta_k}(\cdot)$ are replaced by the matrices $\boldsymbol{F}_{\theta_k}[k]$ and $\boldsymbol{H}_{\theta_k}[k]$. When the process and measurement noises can be modeled as Gaussian random processes, we assume they are distributed as $v[k] \sim \mathcal{N}(0, \boldsymbol{Q})$ and $w[k] \sim \mathcal{N}(0, \boldsymbol{R})$, respectively, with $\boldsymbol{Q}$ and $\boldsymbol{R}$ as the corresponding covariance matrices. Again, while we assume that the mode θ_k in effect at time k is not known, at times we will assume knowledge of the transition matrix $\boldsymbol{P}$ that completely characterizes the statistics of the underlying

Markov chain.

We now discuss a specific system model for an application within this class of switching systems: communication through a time-varying Gilbert-Elliott channel which models switching between two different modes.

7.2.1 Gilbert-Elliott Channel

A Gilbert-Elliott (GE) channel models communication through narrowband channels that tend to induce burst errors in the received data. Generally, these bursts of contiguous erroneous symbols arise when the channel makes a jump from a "good" mode to a "bad" mode. Thus, such a channel model has $N = 2$ modes where $\theta_k = 0$ represents the situation where the system is in the "good" channel mode, and $\theta_k = 1$ represents the "bad" channel mode that induces burst errors.

In this paper, the unknown baseband channel gain is assumed to be complex, representing in-phase and quadrature components. In addition, we assume that the channel is time-varying according to an autoregressive (AR) process, and the measurement $y[k]$ represents noisy observations of the complex channel gain. While the channel gain itself is time-varying, the family of matrices $\boldsymbol{F}_{\theta_k}$ governing the AR system dynamics for each mode is assumed to be static in this application. Specifically, the scalar complex channel gain $h[k]$ at time k is modeled by an mth-order AR process

$$h[k] = \sum_{j=1}^{m} a_{j,\theta_k} h[k-j] + v'[k] \tag{7.3}$$

where a_{j,θ_k} denotes the AR parameters of mode θ_k for $j \in \{1, \ldots, m\}$ and $v'[k]$ is an i.i.d. zero-mean complex Gaussian scalar process with variance σ_v^2. Because Mode 0 represents the "good" channel mode, the AR coefficients $a_{j,0}$ corresponding to this mode are expected to lead to more reliable communication than the set of AR coefficients $a_{j,1}$ corresponding to the "bad" Mode 1. For example, the Mode 0 coefficients might be easier to estimate or might lead to higher average received signal-to-noise ratio.

Putting the AR model in the dynamical systems framework above, this leads to a MJLS governed by these equations:

$$\begin{aligned} x[k+1] &= \left[\begin{array}{ccc|c} a_{1,\theta_k} & a_{2,\theta_k} & \cdots & a_{m,\theta_k} \\ \hline & & & 0 \\ & \boldsymbol{I}_{m-1} & & \vdots \\ & & & 0 \end{array}\right] x[k] + v[k] \\ y[k] &= \left[\begin{array}{cccc} 1 & 0 & \cdots & 0 \end{array}\right] x[k] + w[k] \end{aligned}$$

where the state vector is defined in terms of the scalar AR process above as $x[k] = [h[k-1] \cdots h[h-m]]^\top \in \mathbb{C}^m$ and $\boldsymbol{I}_{m-1}$ is the identity matrix of size $m-1$. The process noise is distributed as $v[k] \sim \mathcal{CN}(0, Q)$, though in this case $\boldsymbol{Q}$ is all zeros except for the top left corner which equals $[\boldsymbol{Q}]_{0,0} = \sigma_v^2$, so that process noise is only added to the topmost state, making the model match the scalar AR process model (7.3). The observation $y[k]$ is a complex scalar, and the measurement noise is distributed as $w[k] \sim \mathcal{CN}(0, \sigma_w^2)$. Finally, the statistics of the Markov mode switching are uniquely defined by the initial state probabilities and the state transition matrix $\boldsymbol{P}$, as depicted in Figure 7.1, where the elements of the matrix $\boldsymbol{P}$ are given by p_{ij}.

7.3 Results

We now present the simulated performance of the TCN when used for prediction. The performance of the TCN is compared to the IMM, LS, and a "Genie Kalman Filter" which is a time-varying KF with perfect knowledge of the current mode θ_k. When the current mode θ_k is known and the mode dynamics are linear, the system reduces to a standard linear time-varying dynamical model for which the KF is optimal. Hence, for linear mode dynamics, the MSE of the Genie KF establishes a lower bound by which other algorithms can be compared.

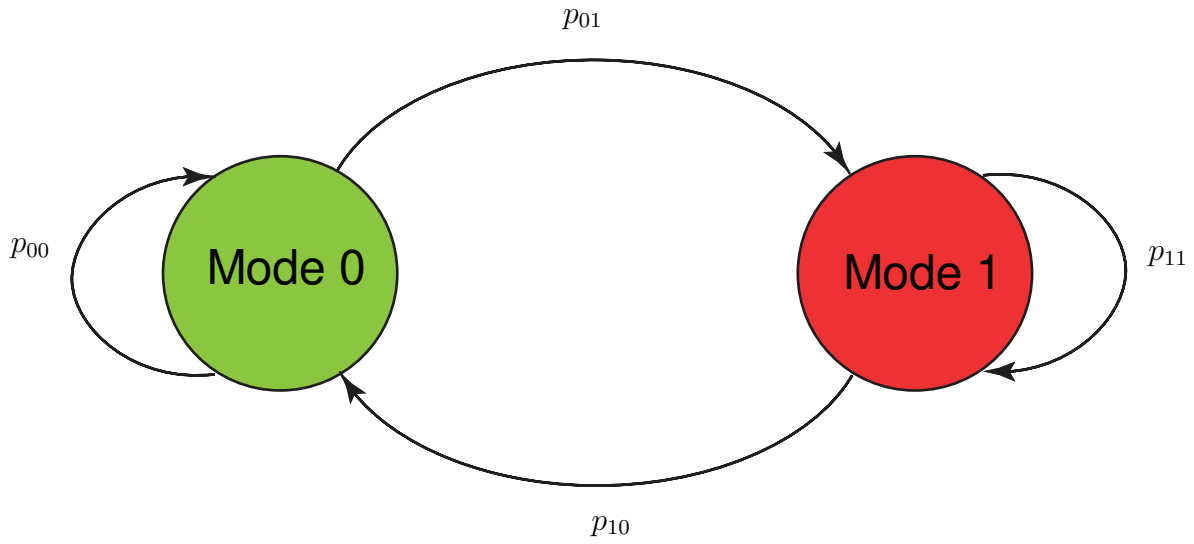

Figure 7.1: Two-state Markov chain model switching in the Gilbert-Elliott channel

For our simulations we selected our AR coefficients as

$$\text{Mode 0 (good)} : a_{1,0} = 0.3,\ a_{2,0} = 0.1$$

$$\text{Mode 1 (bad)} : a_{1,1} = 1.949,\ a_{2,1} = -0.95.$$

The transition probabilities of the Markov chain that define the mode were set to $p_{00} = p_{11} = 0.9995$ and $p_{01} = p_{10} = 0.0005$, forming a homogeneous Markov chain. We selected these parameters so that the probability of switching would be low, such that the model would stay in either mode for an extended period of time in order to model a burst noise channel [45]. The Kalman filters matching Mode 0 and Mode 1 are denoted as "KF0" and "KF1", respectively. For additional details about our implementation of the Gilbert-Elliott channel, we refer the reader to the code repository [89].

The TCN used for this problem was designed with a total of 911,882 trainable parameters comprising 1,322 bias terms and 910,560 network weights. We trained a TCN on 2×10^7 samples to predict a pair of numbers representing the real and imaginary portion of the complex state of the channel. For each prediction, the TCN was provided an input containing the 10 most recent complex noisy channel observations. The complex channel

observations were split into real and imaginary parts [101] such that, for each prediction, the TCN used an input from $\mathbb{R}^{10\times 2}$.

Table 7.1: Gilbert-Elliott channel prediction MSE.

	With mode switching	**Only mode** 0 **("good")**	**Only mode** 1 **("bad")**
Genie KF	**0.216**	**0.106**	**0.321**
KF0	291	**0.106**	560
KF1	0.427	0.352	**0.321**
Least Squares	0.409	0.294	0.344
TCN	**0.292**	**0.106**	**0.322**

Results for the Gilbert-Elliott scenario described in Table 7.1 were obtained from Monte Carlo simulations with 10^5 output pairs. The results are further broken down into performance on a model with mode switching and two models without mode switching ("only mode 0" and "only mode 1"). As expected, KF0 is optimal for a system with "only mode 0" dynamics and KF1 is optimal for a system with "only mode 1" dynamics, while both perform badly in a system with mode switching. The Genie KF is optimal in all cases since it has perfect knowledge of the mode. The TCN is close to optimal in both cases without mode switching and outperforms all algorithms with switching, achieving an MSE close to that of the Genie KF. The poor performance of LS under mode 0 can be explained in by the fact that the LS method is a data driven method that can converge to Kalman Filter like performance.

It is notable that KF0 has poor performance when making predictions of states generated under mode 1. The large errors experienced by KF0 under mode 1 are due to the extreme coefficient mismatch between what KF0 expects and the actual model coefficients in mode 1. Intuitively, KF0 assumes that the current and previous states will have a small effect on the next prediction, which is optimal for mode 0 but far from optimal under mode 1.

As for the LS predictor, with mode switching, LS trains on a combination of mode 0 and mode 1 data. As can be seen from the performance of KF1 and KF0 under the

respective mismatch scenarios, a KF with coefficients that align with the higher error rate model will perform better when model mismatch occurs, compared to a KF with coefficients more effective under a lower error rate model. Hence, LS finds a model such that the equivalent KF would perform best across all modes. This will of course result in performance closer aligned to KF1 than KF0, since the performance of KF0 under mode 1 is so poor.

Figures 7.2 and 7.3 demonstrate the algorithms' performance from initialization to steady state averaged over 5000 Monte Carlo simulations.

Figure 7.2 demonstrates the mean squared prediction error performance for predictions of the first state of the AR process by the TCN, LS, and KF0 for the "only mode 0" case. These results show that the TCN closely tracks the (optimal) performance of KF0. Figure 7.3 similarly shows the achieved performance for the "only mode 1" case. In this case, the TCN mean squared prediction error performance lags approximately one sample behind the (optimal) performance of KF1. In both of these examples, the TCN achieves the steady state performance shown by the black dashed "Ricatti line" within a handful of samples, despite the TCN being initialized with zeros.

7.4 Conclusion

In this chapter, we applied TCNs to predict states of dynamical systems with model switching. We considered Gilbert-Elliott channel as an example and demonstrated that the TCN achieves a better mean-squared prediction error compared to classical algorithms. Possible future directions include investigating the use of TCNs in other dynamical systems with model switching, such as automotive traffic modeling, power plant control, wireless energy transfer, and scheduling information transfer in communications channels. Because research on TCNs more broadly is advancing at a fast pace, further research could also include recently enhancements to the TCN architecture and training approach. Additionally, it may be beneficial to compare performance of the TCN to other

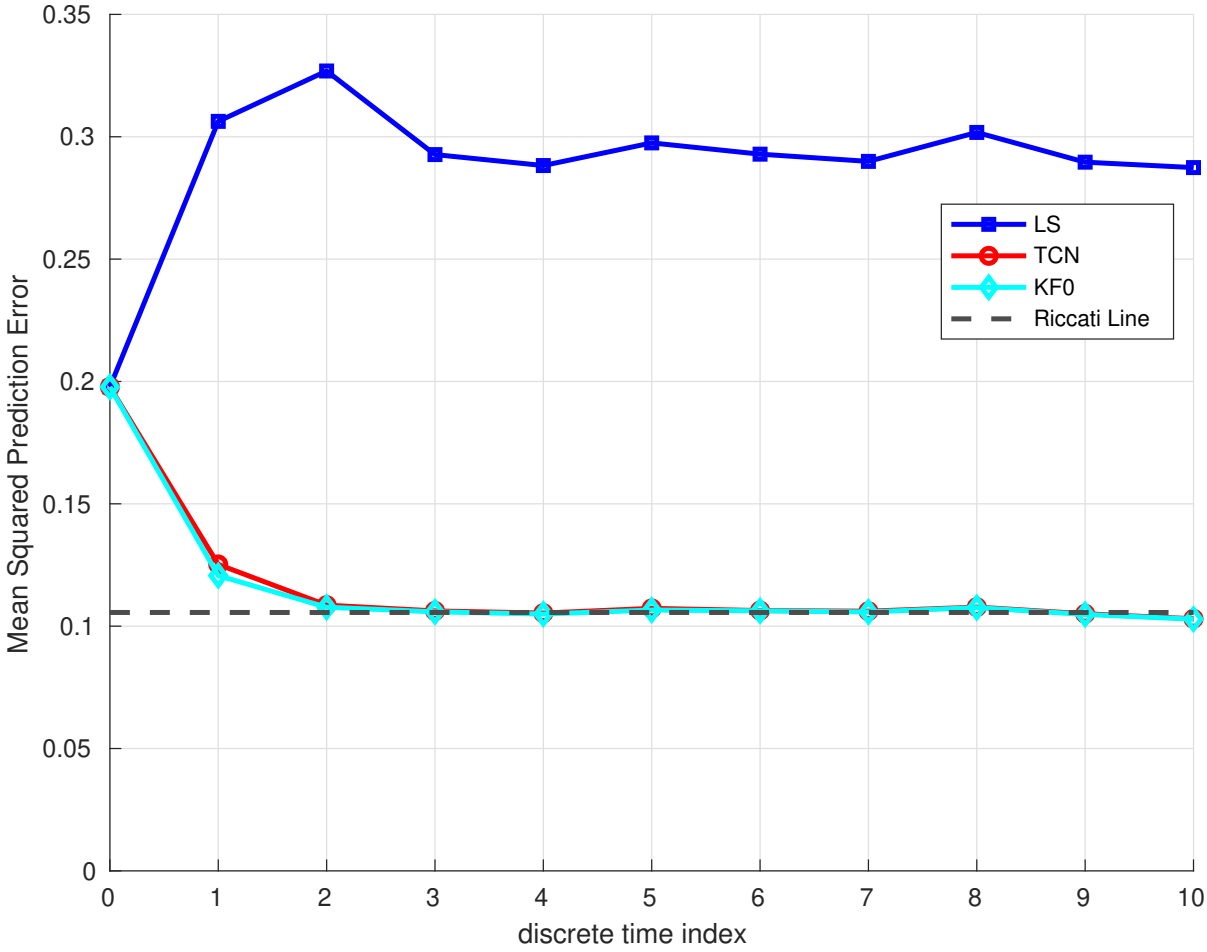

Figure 7.2: Gilbert-Elliott mode 0 mean squared prediction error.

deep learning architectures to give a more complete performance evaluation.

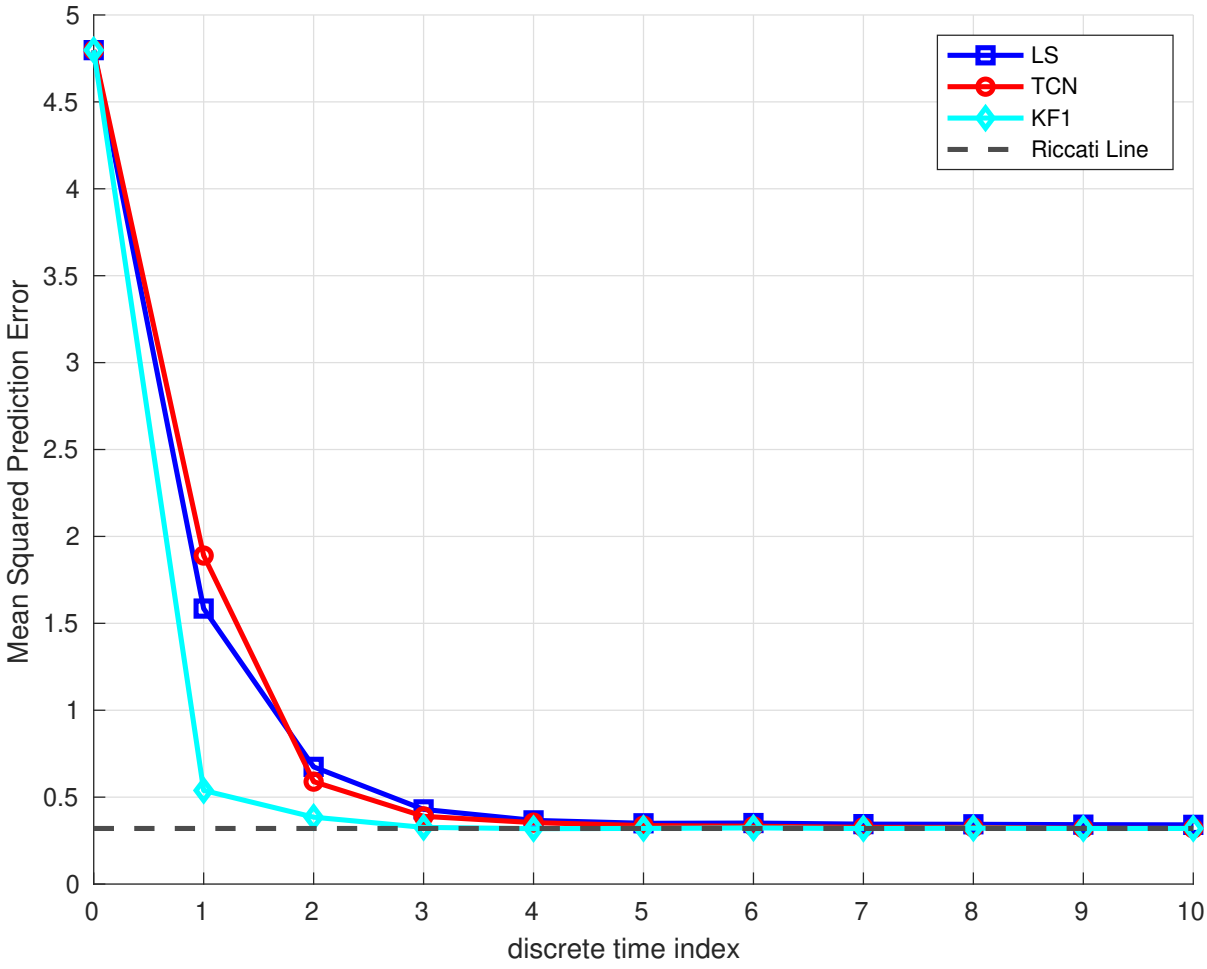

Figure 7.3: Gilbert-Elliott Mode 1 ("bad channel") prediction error, averaged over 5,000 realizations.

Chapter 8

Conclusion and Future Work

8.1 Machine Learning for Reliable Communication

We demonstrate that end-to-end deep neural network based learning can be adapted effectively for physical layer radio systems to achieve state-of-the-art performance in numerous scenarios.

First, we considered Additive White Gaussian Noise (AWGN) channels and formulated efficient parameterizations for linear block codes. We discovered that the transmitter side is just an embedding for linear block codes and the receiver side is just a matched filter for AWGN channels. We demonstrated the performance in three cases in Hamming codes: $(2, 4), (7, 4), (15, 11)$ and Extended Golay $(24, 12)$ codes and show the nuances in how they work and present an in-depth analysis of why they work well - specifically, on how autoencoders depend on overall distance properties, not on minimum distance. We made improvements on training speed and reducing parameters in autoencoders.

Then, we apply this framework to Bernoulli-Gaussian (BGIN) channels and show that the trained autoencoder uniformly outperforms classical block codes in the BGIN channel even when impulsive noise mitigation techniques such as blanking and clipping are employed. We also propose a parsimonious architecture with Bernoulli probabilities.

For future research, we can consider communicating through channels with memory

such as Gilbert-Elliot channels and consider Markov-Gaussian noise models. We can further reduce training load with domain adaptation and randomization and transfer learning. We can consider more channels with model and algorithmic deficit such as interference channels, Rayleigh block fading channels, deletion channels. Few other directions of research can comprise learning of probabilistic shaping as well as schemes for multiuser communications, i.e., multiple access and broadcast channels.

We believe that learning-based optimization of the full physical layer for a point-to-point link could be completely automated and that this would be one of the key ingredients of next-generation communication systems.

8.2 Machine Learning for Improved Tracking of Dynamical Systems

In this part, we applied machine learning to predict states of dynamical systems under parametric mismatch and model switching. We propose two different flavors of using machine learning: to *assist* Kalman filters, to *replace* Kalman filters. We present results for both cases.

We improved the performance under mismatched noise covariances by proposing a deep learning approach. We demonstrate this in the context of oscillator phase predictions where CNNs improve the performance of the phase predictions as compared to Kalman filter in the mismatched cases.

Then, we introduced a new deep learning Kalman filter hybrid framework the Autoencoder Interacting Multiple Model, as an extension to the Autoencoder Kalman Filter, to solve challenging maneuvering target tracking problems. We provide a proof-of-concept demonstration with simulated flight tracking data and compare it against state-of-the-art methods in tracking such as the Interacting Multiple Model, traditional deep learning methods such as Long-Short Term Memory network along with the Autoencoder Kalman

Filter.

We also considered Gilbert-Elliott channels and demonstrated that a temporal convolutional network (TCN) achieves a better mean-squared prediction error compared to classical algorithms.

The proposed approach is general and can be applied to a wide variety of dynamic random process. It would be interesting to perform a rigorous theoretical analysis of AEIMM and hybrid algorithms in general and know more about the bounds on their optimality and limitations.

There are many directions that we can extend this work in exploring few other mismatch cases where Kalman filter fails to reach an optimum solution like non-linearities, non-Gaussian and dimension mismatch. We could also consider few other deep learning prediction models like autoencoders and Gated Recurrent Units. There are reinforcement learning based approaches like model-based, where we estimate the system model from observations and solve it with reinforcement learning and model-free, where we learn the policy directly without estimating the system model.

Future directions can also include investigating the use of TCNs in other dynamical systems with model switching, such as automotive traffic modeling, power plant control, wireless energy transfer, and scheduling information transfer in communications channels.

We believe that prediction algorithms that are robust to parametric mismatches and model switching can assist in building advanced tracking methods where the exact model and parameters are inaccurate.

Bibliography

[1] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 9458–9468.

[2] A. Balatsoukas-Stimming and C. Studer, "Deep unfolding for communications systems: A survey and some new directions," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 266–271.

[3] M. L. Weiss, R. C. Paffenroth, and J. R. Uzarski, "The autoencoder-kalman filter: Theory and practice," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 2176–2179.

[4] M. Weiss, R. C. Paffenroth, J. R. Whitehill, and J. R. Uzarski, "Deep learning with domain randomization for optimal filtering," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019, pp. 1779–1786.

[5] Z. Qin, G. Y. Li, and H. Ye, "Federated learning and wireless communications," *arXiv preprint arXiv:2005.05265*, 2020.

[6] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing*

Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[8] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, Dec. 2017, ISSN: 2332-7731. DOI: 10.1109/TCCN.2017.2758370.

[9] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel, "Backprop kf: Learning discriminative deterministic state estimators," in *Advances in Neural Information Processing Systems*, 2016, pp. 4376–4384.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[11] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[13] F. A. Aoudia and J. Hoydis, "End-to-End Learning of Communications Systems Without a Channel Model," 2018. arXiv: 1804.02276.

[14] L. Ambrogioni, U. Güçlü, E. Maris, and M. van Gerven, "Estimating nonlinear dynamics with the convnet smoother," *arXiv preprint arXiv:1702.05243*, 2017.

[15] 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*, 2020.

[16] J. Proakis, *Digital Communications*, ser. McGraw-Hill series in electrical and computer engineering : communications and signal processing. McGraw-Hill, 2001, ISBN: 9780071181839.

[17] S. Dörner, S. Cammerer, J. Hoydis, and S. t. Brink, "Deep learning based communication over the air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, Feb. 2018, ISSN: 1941-0484.

[18] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. 12, pp. 3371–3408, 2010.

[19] T. J. O'Shea, T. Roy, N. West, and B. C. Hilburn, "Physical layer communications system design over-the-air using adversarial networks," in *2018 26th European Signal Processing Conference (EUSIPCO)*, IEEE, 2018, pp. 529–532.

[20] 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*, Oct. 2018, pp. 298–303.

[21] D. Gündüz, P. de Kerret, N. D. Sidiropoulos, D. Gesbert, C. R. Murthy, and M. van der Schaar, "Machine learning in the air," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2184–2199, 2019.

[22] A. Felix, S. Cammerer, S. Dörner, J. Hoydis, and S. Ten Brink, "Ofdm-autoencoder for end-to-end learning of communications systems," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Jun. 2018, pp. 1–5.

[23] G. D. Forney, "Trellis shaping," *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 281–300, Mar. 1992, ISSN: 0018-9448.

[24] G. Böcherer, F. Steiner, and P. Schulte, "Bandwidth efficient and rate-matched low-density parity-check coded modulation," *IEEE Transactions on Communications*, vol. 63, no. 12, pp. 4651–4665, Dec. 2015, ISSN: 0090-6778.

[25] R. Pighi, M. Franceschini, G. Ferrari, and R. Raheli, "Fundamental performance limits of communications systems impaired by impulse noise," *IEEE Trans. on Comm.*, vol. 57, no. 1, pp. 171–182, 2009, ISSN: 00906778.

[26] Z. Mei, M. Johnston, S. Le Goff, and L. Chen, "Error probability analysis of m-qam on rayleigh fading channels with impulsive noise," in *2016 IEEE 17th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Jul. 2016, pp. 1–5. DOI: 10.1109/SPAWC.2016.7536755.

[27] T. Shongwey, A. H. Vinck, and H. C. Ferreira, "On impulse noise and its models," in *Power Line Communications and its Applications (ISPLC), 2014 18th IEEE International Symposium on*, IEEE, 2014, pp. 12–17.

[28] D. Fertonani and G. Colavolpe, "On reliable communications over channels impaired by bursty impulse noise," *IEEE Transactions on Communications*, vol. 57, no. 7, pp. 2024–2030, 2009, ISSN: 00906778. DOI: 10.1109/TCOMM.2009.07.070638.

[29] H. Hamad and G. M. Kraidy, "Performance analysis of convolutional codes over the bernoulli-gaussian impulsive noise channel," in *2017 15th Canadian Workshop on Information Theory (CWIT)*, Jun. 2017, pp. 1–5.

[30] K. M. Rabie and E. Alsusa, "Improving blanking/clipping based impulsive noise mitigation over powerline channels," in *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Sep. 2013, pp. 3413–3417. DOI: 10.1109/PIMRC.2013.6666738.

[31] S. Tseng, D. Tseng, T. Tsai, and Y. S. Han, "Robust turbo decoding in single-carrier systems over memoryless impulse noise channels," in *2016 International Conference on Advanced Technologies for Communications (ATC)*, Oct. 2016, pp. 344–349.

[32] S. M. Kabir, A. Mirza, and S. A. Sheikh, "Impulsive noise reduction method based on clipping and adaptive filters in awgn channel," *International Journal of Future Computer and Communication*, vol. 4, no. 5, p. 341, 2015.

[33] J.-M. Kang, C.-J. Chun, I.-M. Kim, and D. I. Kim, "Channel tracking for wireless energy transfer: A deep recurrent neural network approach," *arXiv:1812.02986*, 2018.

[34] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289, ISBN: 1-55860-778-1.

[35] L. Ambrogioni, U. Güçlü, E. Maris, and M. van Gerven, "Estimating nonlinear dynamics with the convnet smoother," *arXiv preprint arXiv:1702.05243*, 2017.

[36] R. G. Krishnan, U. Shalit, and D. Sontag, "Deep kalman filters.(2015)," *arXiv preprint arXiv:1511.05121*, 2015.

[37] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[38] R. Mehra, "On the identification of variances and adaptive kalman filtering," *IEEE Transactions on automatic control*, vol. 15, no. 2, pp. 175–184, 1970.

[39] Y. Bar-Shalom, X. Rong Li, and T. Kirubarajan, *Estimation with Appications to Tracking and Navigation*. Wiley, New York, 2001, p. 584.

[40] H. A. P. Blom and Y. Bar-Shalom, "The interacting multiple model algorithm for systems with markovian switching coefficients," *IEEE Transactions on Automatic Control*, vol. 33, no. 8, pp. 780–783, 1988.

[41] P. R. Mahapatra and K. Mehrotra, "Mixed coordinate tracking of generalized maneuvering targets using acceleration and jerk models," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 3, pp. 992–1000, 2000.

[42] M. Roth, G. Hendeby, and F. Gustafsson, "Ekf/ukf maneuvering target tracking using coordinated turn models with polar/cartesian velocity," in *17th International Conference on Information Fusion (FUSION)*, IEEE, 2014, pp. 1–8.

[43] M. E. Farmer, Rein-Lien Hsu, and A. K. Jain, "Interacting multiple model (imm) kalman filters for robust high speed human motion tracking," in *Object recognition supported by user interaction for service robots*, vol. 2, 2002, 20–23 vol.2.

[44] O. L. V. Costa, M. D. Fragoso, and R. P. Marques, *Discrete-time Markov jump linear systems*. Springer Science & Business Media, 2006.

[45] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell system technical journal*, vol. 39, no. 5, pp. 1253–1265, 1960.

[46] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[47] A. Martinez and A. G. i Fàbregas, "Saddlepoint approximation of random-coding bounds," in *2011 Information Theory and Applications Workshop*, Feb. 2011, pp. 1–6.

[48] Y. Polyanskiy, H. V. Poor, and S. Verdu, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, May 2010.

[49] Y. Polyanskiy, "Saddle point in the minimax converse for channel coding," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 2576–2595, May 2013. DOI: 10.1109/TIT.2012.2236382.

[50] F. Peng, J. Zhang, and W. E. Ryan, "Adaptive modulation and coding for ieee 802.11 n," in *2007 IEEE Wireless Communications and Networking Conference*, IEEE, 2007, pp. 656–661.

[51] C. Fischione, K. H. Johansson, A. Sangiovanni-Vincentelli, and B. Z. Ares, "Minimum energy coding in cdma wireless sensor networks," *IEEE transactions on wireless communications*, vol. 8, no. 2, pp. 985–994, 2009.

[52] T. Fehenberger, A. Alvarado, G. Böcherer, and N. Hanik, "On probabilistic shaping of quadrature amplitude modulation for the nonlinear fiber channel," *Journal of Lightwave Technology*, vol. 34, no. 21, pp. 5063–5073, 2016.

[53] B. D. Anderson and J. B. Moore, *Optimal Filtering*. Dover Publications, 1979, p. 368.

[54] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960, ISSN: 0021-9223. DOI: 10.1115/1.3662552.

[55] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*, 1st. Hoboken: Wiley, 2006, p. 552, ISBN: 0471708585.

[56] P. A. Ruymgaart, T. T. Soong, and T. S. T., *Mathematics of Kalman-Bucy Filtering*, 1st. Berlin Heidelberg New York Tokyo: Springer-Verlag, 1985, vol. 136, p. 170, ISBN: 3-540-13508-1.

[57] Y. Zhang, "Hourly traffic forecasts using interacting multiple model (imm) predictor," *IEEE Signal Processing Letters*, vol. 18, no. 10, pp. 607–610, 2011.

[58] M. Sandell and U. Raza, "Application layer coding for iot: Benefits, limitations, and implementation aspects," *IEEE Systems Journal*, vol. 13, no. 1, pp. 554–561, 2018.

[59] K. Vedula, R. Paffenroth, and D. R. Brown, "Joint coding and modulation in the ultra-short blocklength regime for bernoulli-gaussian impulsive noise channels using autoencoders," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 5065–5069.

[60] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–27, 2009, ISSN: 19358237.

[61] G. E. Hinton* and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *International Encyclopedia of Education*, vol. 313, no. July, pp. 504–507, 2010. DOI: 10.1016/B978-0-08-044894-7.00081-6.

[62] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. arXiv: 1412.6980 [cs.LG].

[63] M. Sadeghi and E. G. Larsson, "Physical adversarial attacks against end-to-end autoencoder communication systems," *IEEE Communications Letters*, vol. 23, no. 5, pp. 847–850, 2019.

[64] N. Wu, X. Wang, B. Lin, and K. Zhang, "A cnn-based end-to-end learning framework toward intelligent communication systems," *IEEE Access*, vol. 7, pp. 110 197–110 204, 2019.

[65] L. Rugini, "Symbol error probability of hexagonal qam," *IEEE communications letters*, vol. 20, no. 8, pp. 1523–1526, 2016.

[66] M. Tanahashi and H. Ochiai, "A multilevel coded modulation approach for hexagonal signal constellation," *IEEE Transactions on Wireless Communications*, vol. 8, no. 10, pp. 4993–4997, 2009.

[67] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms*. New York, NY, USA: Cambridge University Press, 2002, ISBN: 0521642981.

[68] T. Erseghe, "Coding in the finite-blocklength regime: Bounds based on laplace integrals and their asymptotic approximations," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 6854–6883, Dec. 2016. DOI: 10.1109/TIT.2016.2616900.

[69] S. Lin and D. J. Costello, *Error control coding: fundamentals and applications*. Upper Saddle River, NJ: Pearson/Prentice Hall, 2004.

[70] G. Ndo, P. Siohan, and M. Hamon, "Adaptive noise mitigation in impulsive environment: Application to power-line communications," *IEEE Transactions on Power Delivery*, vol. 25, no. 2, pp. 647–656, Apr. 2010.

[71] L. Galleani, "A tutorial on the two-state model of the atomic clock noise," *Metrologia*, vol. 45, no. 6, S175, 2008.

[72] M. E. Rasekh, U. Madhow, and R. Mudumbai, "Frequency tracking with intermittent wrapped phase measurement using the rao-blackwellized particle filter," in *2014 48th Asilomar Conference on Signals, Systems and Computers*, Nov. 2014, pp. 247–251. DOI: 10.1109/ACSSC.2014.7094438.

[73] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[74] A. Lee, "Circular data," *WIREs Comput. Stat.*, vol. 2, no. 4, pp. 477–486, Jul. 2010, ISSN: 1939-5108.

[75] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[76] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[77] B. Kiumarsi, H. Modares, and F. L. Lewis, *Optimal Tracking Control of Uncertain Systems : On-Policy and Off-Policy Reinforcement Learning Approaches*. Elsevier Inc., 2016, pp. 165–186, ISBN: 9780128052464.

[78] R. Visina, "Tracking highly-maneuvering targets and data fusion from imm estimator tracks," 2019.

[79] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018.

[80] N. N. Krasovskii and E. A. Lidskii, "Analytical design of controllers in systems with random attributes," *Automat. Remote Contr.*, vol. 22, no. 9, pp. 1021–1025, 1961.

[81] P. Zhao, Y. Kang, and Y. Zhao, "A brief tutorial and survey on markovian jump systems: Stability and control," *IEEE Systems, Man, and Cybernetics Magazine*, vol. 5, no. 2, pp. 37–C3, 2019.

[82] X. R. Li and V. P. Jilkov, "Survey of maneuvering target tracking. part i. dynamic models," *IEEE Transactions on aerospace and electronic systems*, vol. 39, no. 4, pp. 1333–1364, 2003.

[83] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[84] J. Yan, L. Mu, L. Wang, R. Ranjan, and A. Y. Zomaya, "Temporal convolutional networks for the advance prediction of enso," *Scientific Reports*, vol. 10, no. 1, pp. 1–15, 2020.

[85] W. Zhao, Y. Gao, T. Ji, X. Wan, F. Ye, and G. Bai, "Deep temporal convolutional networks for short-term traffic flow forecasting," *IEEE Access*, vol. 7, pp. 114 496–114 507, 2019.

[86] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[87] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 156–165.

[88] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[89] A. Grootveld *et al.* (). "Source code used in 'Tracking of dynamical processes with model switching using temporal convolutional networks'," [Online]. Available: `https://github.com/aspectlab/ModelSwitching`.

[90] B. Beşbınar and A. A. Alatan, "Visual object tracking with autoencoder representations," in *2016 24th Signal Processing and Communication Application Conference (SIU)*, 2016, pp. 2041–2044.

[91] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2017, pp. 23–30.

[92] R. Antonova, S. Cruciani, C. Smith, and D. Kragic, "Reinforcement learning for pivoting task," *CoRR*, vol. abs/1703.00472, 2017. arXiv: `1703.00472`. [Online]. Available: `http://arxiv.org/abs/1703.00472`.

[93] Y. Zhu, Z. Wang, J. Merel, A. A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, and N. Heess, "Reinforcement and imitation learning for diverse visuomotor skills," *CoRR*, vol. abs/1802.09564, 2018.

[94] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," 2017.

[95] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *arXiv preprint arXiv:1703.06907*, 2017.

[96] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," *CoRR*, vol. abs/1707.02267, 2017.

[97] J. Chakravorty and A. Mahajan, "Structure of optimal strategies for remote estimation over gilbert-elliott channel with feedback," in *2017 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2017, pp. 1272–1276.

[98] X. Ren, J. Wu, K. H. Johansson, G. Shi, and L. Shi, "Infinite horizon optimal transmission power control for remote state estimation over fading channels," *IEEE Transactions on Automatic Control*, vol. 63, no. 1, pp. 85–100, 2017.

[99] E. O. Elliott, "Estimates of error rates for codes on burst-noise channels," *The Bell System Technical Journal*, vol. 42, no. 5, pp. 1977–1997, 1963.

[100] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai, "Ieee 802.11 wireless local area networks," *IEEE Communications magazine*, vol. 35, no. 9, pp. 116–126, 1997.

[101] N. Taşpinar and M. N. Seyman, "Back propagation neural network approach for channel estimation in ofdm system," in *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*, IEEE, 2010, pp. 265–268.