Securing Physical Layer Passband Signals Via Machine Learning

by

Kyle W. McClintick

A Dissertation
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

____

December 2021

APPROVED:

____

Professor Alexander Wyglinski, WPI ECE, Major Advisor

____

Professor Andrew Clark, WPI ECE

____

Professor Ziming Zhang, WPI ECE

____

Dr. Jacob Harer, MIT Lincoln Laboratory

Abstract

Machine Learning (ML)-based Wireless Sensor Networks (WSNs) need secure algorithms to avoid human and financial costs in vehicular, agricultural, and national defense applications. Thus, there are several challenges within the context of ML-based WSNs that require novel solutions from the technical community. For instance, there does not currently exist a stationary wireless localization algorithm capable of classifying the source of signals detected by WSNs. Another challenge involves the active obfuscation of transmission modulation class from eavesdropping ML-based WSNs, where power inefficiency and the lack of a Quality-of-Service (QoS) guarantee makes the realization of a practical solution difficult to achieve. Finally, implementations of obfuscation techniques and their countermeasures within a WSN application do not possess the same level of advances and sophistication when compared to other domains such as image processing, which have been employing ML-based solutions for several decades. To address these challenges, this dissertation presents three separate (yet connected) novel contributions. First, we present a novel localization framework that is self-organizing, opportunistic, and jointly localizes stationary transmitters while classifying signal detections. Experimental results from this framework demonstrated the ability to locate an unknown number of low-power beacons with 1.2-meter error bias. Second, we devised two new constraints on ML solutions to help increase power efficiency and guarantee a minimum QoS in WSNs, namely, constructive and Bandwidth Equivalency (BWE) perturbation constraints. Based on these novel constraints, we showed an improvement of up to 80% with respect to the Bit Error Rate (BER) and a reduction of up to 60% with respect to adversarial classification precision. Finally, we implemented the first-ever real-world hardware experimentation of an adversary-trained, ML-based, modulation classifying sensor, and evaluated a case study involving wireless adversarial training, regularization, model width, label leaking, parallel and cascade variations, and iterative attacks. Results from this case study using this hardware setup showed that strong Gradient Sign Method (GSM) attacks can reduce classification precision down to guessing precision, but that moderate strength attacks can be resisted by up to 35% while simultaneously increasing non-adversarial precision by up to 1%.

Acknowledgements

The work presented in this dissertation was generously funded by MIT Lincoln Laboratories, with special thanks to Allyn Dullighan, Tim Gallagher, Jacob Harer, Mark Smith, and Mike Tracey.

The formation of this dissertation was guided by my adviser Alexander Wyglinski and committee members Andrew Clark and Ziming Zhang, as well as the Wilab, with special thanks to Kuldeep Gill.

Some of the research presented in this dissertation has received significant guidance and intellectual contributions from Bryse Flowers of the University of California San Diego and William Headley of Virginia Technology's Hume Center.

I would also like the thank my wife Gigi for her love and support.

Contents

List of Figures

List of Tables

Acronyms

ACM	Automatic Coding and Modulation
AD	Anderson-Darling
ADC	Analog-to-Digital Conversion
ADS-B	Automatic Dependent Surveillance Broadcast
AGC	Automatic Gain Control
AI	Artificial Intelligence
ASK	Amplitude Shift Keying
AUC	Area Under the Curve
AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BER	Bit Error Rate
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
BPF	Band Pass Filter
BPSK	Binary Phase Shift Keying
BS	Base Station
BWE	Band Width Equivalence
CDF	Cumulative Distribution Function
CDMA	Code Division Multiple Access
CFAR	Constant False Alarm Rate
CFGSM	Constructive FGSM
CFO	Carrier Frequency Offset
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
CR	Cognitive Radio
CRLB	Cramer Rao Lower Bound
CSI	Channel State Information
CUE	Cellular User Equipment
DA	Data Augmentation
DAC	Digital-to-Analog Conversion

DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DC	Data Choice
DPGMM	Dirichlet Process Gaussian Mixture Model
DPP	Data Pre-Processing
DSA	Dynamic Spectrum Access
DT	Decision Tree
DTV	Digital Television
DUE	Device-to-Device User Equipment
EM	Expectation Maximization
FFT	Fast Fourier Transform
FGSM	Finite Gradient Sign Method
FIR	Finite Impulse Response
FSK	Frequency Shift Keying
GAN	Generative Adversarial Network
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit
GR	GNU Radio Companion
GSM	Global System for Mobile Communications
IE	Input Equivariance
IF	Intermediate Frequency
IoT	Internet of Things
IQ	In-Phase/Quadrature
ISI	Inter-Symbol Interference
iterLL	Iterative Least Likely
KKT	Karush-Kuhn-Tucker
KM	Kernel Method
KNN	K-Nearest Neighbor
KW	Kruskal-Wallis
LDPC	Low-Density Parity-Check

LIDAR	Light Detection and Ranging
LNA	Low Noise Amplifier
LOS	Line of Sight
LPF	Low Pass Filter
LSB	Least Significant Bits
LSR	Least Squares Regression
LSTM	Long Short-Term Memory
MAC	Medium Access Control
MCMC	Markov Chain Monte Carlo
MHT	Multi-Hypothesis Tracking
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MLE	Most Likely Estimation
MSE	Mean Squared Error
NP	Neyman-Pearson
NLL	Negative Log Likelihood
NLOS	Non Line of Sight
NN	Neural Network
OOB	Out-of-Band
OOK	On-Off Keying
OOT	Out-of-Tree
OPTICS	Ordering Points To Identify Cluster Structure
PCA	Principal Component Analysis
PDF	Probability Density Function
PGD	Projected Gradient Descent
PHY	physical
PLL	Phase-Locked-Loop
PPV	Positive Predictive Value
QoS	Quality-of-Service
QQ	Quantile-Quantile

R	Regularization
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RF	Radio Frequency
RFFE	Radio Frequency Front End
RL	Reinforcement Learning
RMS	Root Mean Square
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
RRC	Root Raised Cosine
RSS	Received Signal Strength
RSU	Road Side Unit
SDR	Software Defined Radio
SGD	Stochastic Gradient Descent
SL	Supervised Learning
SNR	Signal-to-Noise Ratio
SON	Self Organizing Networks
SOP	Signal of Opportunity
SPS	Samples Per Symbol
SRO	Symbol Rate Offset
stepLL	One-Step Least Likely
SUMO	Simulation of Urban Mobility
SVM	Support Vector Machine
TDD	Time Division Duplexing
TDoA	Time Difference of Arrival
TPMS	Tire Pressure Monitoring System
UAP	Universal Attack Perturbation
UE	User Equipment
UL	Unsupervised Learning
USRP	Universal Software Radio Peripheral

UWB	Ultra Wide-Band
V	Validation
V2I	Vehicle-to-Infrastructure
VAE	Variational Auto-Encoder
VGG	Visual Geometry Group
VI	Variational Inference
WI	Weight Initialization
WMMSE	Weighted Minimum Mean Square Error
WSN	Wireless Sensor Networks
WSR	Weighted Sum Rate
WU	Weight Updates

Chapter 1

Introduction

1.1 Motivation

Parameter estimation and classification in the presence of noise is a ubiquitous problem in wireless signal processing. Machine Learning (ML) models have a good track record in predicting these unknown parameters in real-world wireless systems with few assumptions about data. However, only recently have they become practical due to faster hardware leveraging graphics processing units and cloud computing [7], more data from increasingly permeating wireless networks collected by cheaper sensors [8], and improved algorithms that train faster [9] while performing more accurate predictions [10]. ML-based wireless signal processing solutions tout a number of advantages, including the ability to implement real-world Cognitive Radio (CR) networks with dynamic spectrum access capabilities [11], Wireless Sensor Networks (WSN) with less re-programming and longer service life [12], Self Organizing Networks (SON) with robust routing protocols [13], and more secure Internet of Things (IoT) networks that fully utilize resources to stay online longer [14], and improve any communication system with lower costs and higher Quality-of-Service (QoS) [15].

Scientific publications re-implementing, analyzing, and clarifying the choices and results of ML applied to wireless communications are relatively difficult to find despite their widespread use. A summary of ML employed in wireless communications is presented in Table 1.1, which describes what paradigm and stack layers are discussed by each work.

Due to training and decision rule concerns raised during the “Artificial Intelligence (AI) winter” [25], ML in wireless communications did not begin to widely appear in published works until the late 1980s. New works at that time were published digital signal processing

Table 1.1: A comparison of surveys and tutorials discussing ML-based wireless communication systems.

References	**ML Paradigms**			**Wireless Stack Layers**				
	SL	**UL**	**RL**	**PHY**	**LNK**	**NET**	**TRAN**	**APP**
[16]	✓			✓	✓	✓	✓	✓
[11]	✓	✓	✓	✓	✓	✓		
[12]	✓	✓	✓		✓	✓	✓	✓
[17]	✓	✓					✓	✓
[13]	✓	✓	✓			✓	✓	✓
[18]	✓					✓	✓	✓
[14]	✓							✓
[19]	✓	✓	✓	✓	✓	✓	✓	✓
[15]	✓	✓	✓	✓	✓	✓	✓	✓
[20]	✓						✓	
[21]	✓	✓				✓	✓	✓
[22]	✓	✓	✓			✓	✓	✓
[23]	✓	✓	✓					✓
[24]	✓			✓				

papers, which often made use of NNs as a non-linear estimator, primarily for equalization [26]. Previously, the Volterra series [27] and nonlinear auto-regressive moving average filters [28] had been used for non-linear estimation. This trend continued until the year 2000 when advances in computing, wireless communications, and ML allowed for the application of ML to other layers of the protocol stack besides the PHY layer. A timeline of the history of ML used in the PHY layer is presented in Table 1.2.

In this dissertation, we present novel contributions to the state-of-the-art applications of ML to estimate the location of wireless transmitters by observing noisy received signals. Specifically, we choose to showcase our results using Signal of Opportunity (SOP), which do not contain timing or location information, or whose timing and location information cannot be observed due to a lack of protocol information. Additionally, we investigate the often overlooked fragility of trained ML algorithms and contribute to the state-of-the-art research of transmitting signals in a way such that ML-based eavesdroppers cannot classify and learn about observed signals.

Table 1.2: Timeline of survey papers exploring the use of SL models in PHY layer applications

Year	**Event**
1986	← Tank [29] implements an ADC NN during what is called the "AI winter", a time when researchers were disillusioned by gradient descent issues.
1989	← AI winter ends, and many DSP works are published leveraging NNs. Bruck *et al.* [30] use a NN to perform MLE decoding of Hamming and Reed-Muller block codes.
1990	← Chen *et al.* [31] and others make use of NNs as a non-linear estimator for equalization.
1991	← Kunz [32] uses a softmax classifier to decide which channel to transmit on within a cellular network, using interference patterns and channel demand as determinants.
1992	← Aazhang *et al.* [33] use a NN to demodulate CDMA signals, additionally "assisting" training by learning with interference in the samples (on-line training).
	← Fang *et al.* [34] use a variational auto-encoder NN to compress data, learning the latent space of data by minimizing reconstruction loss.
1995	← Ibnkahla *et al.* begin publishing works that use a NN to model the inverse non-liner function of traveling wave tubes, wireless channels, and amplifiers. They summarize three years of these works in their 1998 paper [35].
	← Iba [36] and others optimize the power usage of a power distribution system, a technique later used for low power wireless networks such as IoT. Genetic algorithms are used at this time, in favor over initial RNNs (Hopfield NN [37]) due to learning issues present in the latter.
	← Southall *et al.* [38] use an RBF NN to perform beam steering towards far-field targets.
1998	← Nandi *et al.* [39] uses a NN to classify the modulation type of observed signals, and another to classify modulation order.
2000	← Hoppensteadt *et al.* [40] train a NN-based Phase-Locked-Loop (PLL) for phase synchronization of multiple signals.
2010	← Daniels *et al.* [41], among the rise of MIMO technologies, publish an ACM work with an SL model that leverages the estimated matrix utilized in MIMO systems.

1.2 State-of-the-Art

Driverless vehicles could save the U.S. economy an estimated $450 billion per year [42] by improving traffic safety [43,44], increasing the mobility of the impaired and the elderly [45], and reducing pollution to the environment [46]. In order for robust autonomous operation to be realized, these vehicles require nearly comprehensive and continuous situational awareness to make appropriate driving decisions. This awareness is obtained via the use of on-board sensing technologies such as Light Detection and Ranging (LIDAR) [47], computer vision algorithms [48, 49], and radar [50]. A Global Navigation Satellite System (GNSS) such as Global Positioning System (GPS) or similar provides vehicle location data [51]. Accurate location information is critical, especially during maneuvers such as lane changing, navigating an intersection, or merging with high speed traffic [52]. GNSS-based techniques may be unable to handle complex road conditions upon which these vehicles operate, such as urban canyons [145] and foliage [146] that cause significant signal attenuation, as well as being susceptible to unintentional interference [147] and malicious jamming signals [148]. Therefore methods to supplement or replace GNSS have generated significant interest [149].

Alternatively, the position and velocity of vehicles can also be determined by leveraging Signal of Opportunity (SOP)-based navigation and SOP beacon localization methods which assume that source transmitters detected by the WSN can be distinguished using a priori metadata (*e.g.*, physical layer attributes such as carrier frequency). In adversarial wireless localization problems, this meta-data may be limited or unavailable. In this work, we introduce a framework to estimate and classify the following unknown parameters using only physical pass-band signal characteristics detected by the WSN: *i*) the number of transmitters; *ii*) their location; *iii*) which transmitter sent each detected signal. State-of-the-art WSN frameworks that estimate a subset or all of these unknowns are ubiquitously showcased using Automatic Dependent Surveillance Broadcast (ADS-B) signals [53–58]. These solutions estimate Radio Frequency (RF) features that are highly correlated to hardware noise such that the transmitters can be identified [53, 54], perform mobility analysis via Multi-Hypothesis Tracking (MHT) [55], use heuristic thresholds in multilateration or range estimate based verification methods [56, 57], or jointly utilize multiple techniques [58].

In 2014, Goodfellow [59] presented a picture of a panda the world's state-of-the-art Machine Learning (ML) algorithms confidently decided is a gibbon. Ever since, an arms race has been ongoing between crafting adversarial perturbations and developing countermeasures [6, 60–72]. It is important that high-risk wireless communications systems such as autonomous vehicle [73], agricultural [74], and military [75] networks are secure from perturbation attacks because incorrect classifications can result in catastrophic financial and/or human costs. Thus the design of trained ML algorithms for these wireless networks must prevent the creation of additional attack surfaces used by potential adversaries, which requires an understanding of effective perturbation designs in realistic, physical scenarios.

The study of adversarial perturbations in the wireless data domain is relatively new and lagging behind that of leading data domains such as computer vision. Sadeghi *et al.* [76] synchronously added Finite Gradient Sign Method (FGSM) [59] and UAP [63] attacks to received signals over an Additive White Gaussian Noise (AWGN) channel model, highlighting how considerably less power is needed to fool a Convolutional Neural Network (CNN) with perturbations when compared with random jamming signals, and that the UAP [63] attack is robust to time shifts between it and the received signal that simulates synchronization errors. Kim *et al.* [77] analyzed the same threat model and explored how the adversary can used the channel state information matrix to synchronously deliver power and error-optimized white and black box perturbations. Flowers *et al.* [78] use a different threat model, where FGSM [59] perturbations were added by a transmitter to its own signal to fool a CNN-based eavesdropper, and investigated the trade-off between BER and adversarial accuracy. The authors find that perturbations strong enough to be effective at lowering eavesdropper classification accuracy come at the cost of a significant number of communication errors (especially higher order modulation signals), that frequency and timing errors have a small effect on perturbation effectiveness, and that relatively large, single-step perturbations do not always increase loss because of unstable gradient ascent. These lessons motivated Flowers *et al.* [79] to design a feedback loop to the transmitter from the adversary to optimize multi-objective loss functions, which design the perturbations to minimize power consumption, minimize BER, and minimize eavesdropper accuracy. DelVecchio *et al.* [80] similarly optimized the frequency-domain power and bandwidth of perturbations to maximize communication effectiveness without increasing eavesdropper accuracy. Maroto *et al.* [81] implemented adversarial training robust to iterative attacks, but experienced label leaking and weak models due to crafting ground-truth-based perturbations that are overly correlated to trained models, ground truth class, and the non-adversarial data. Zhang *et al.* [82] performed defensive distillation to protect the network from single-step adversarial perturbations, but the process of fooling these networks is well understood [83,84]. Finally, Sahay *et al.* [85] performed a 4-class modulation classification adversarial training simulation using both time and frequency domain features, showing a clear improvement over using time-based features alone. However, they did not show evidence that their novel feature extraction offered improvement upon the moment and cumulant-based features used in state-of-the-art works [3].

1.3 Technical Challenges

Localization algorithms that classify source radio IDs face substantial challenges. Radio-frequency (RF) feature analysis solutions such as the algorithms presented by Moser *et al.* [53] and Schafer [54] require protocol meta-data, such as transmission frequency, to correct for Doppler effects in order when computing their frequency-based feature. Additionally, Moser's phase-based feature requires the detection of spoofing signals transmitted during the tuning interval of the spoofer's Phase-Locked-Loop (PLL), during which the transmitter-specific transient behaviors can be observed. Mobility analysis works such as [55] reliably detect sudden changes to tracks, but can be vulnerable to slowly diverging spoofed tracks and spoofers who are knowledgeable of the propagation delay to trusted transmitters. Multilateration and range-estimate based verification methods have several drawbacks, including being heuristic threshold based classifiers that do not generalize well, along with requirements for non-opportunistic signals to contain position and time stamp information. Moreover, they employ inference techniques that focus only on the binary unknown parameter representing the presence or absence of a second transmitter or spoofer.

As a relatively new field of research, wireless adversarial perturbation papers present many open challenges that have yet to be resolved. Papers that synchronously add physical perturbations to other physical transmissions do not model realistic synchronization errors between the two, with the exception of the time shifted Universal Attack Perturbation (UAP) [63] attacks simulated in [76]. Works that simulate transmitters that add optimized perturbations to their own signals [79, 80] generously assume a white box eavesdropper and have not made attempts to design deterministic perturbations that improve various loss metrics. Additionally, several works [78–80, 86] have noted the frequently used dataset for these studies are generated with several ratio and labeling errors. There has also yet to be an investigation on the effects of the wireless channel applied to the ML-based detection and isolation of these perturbations [64], commonly performed by statistical distribution estimating algorithms such as Variational Auto Encoders (VAEs) [87] or Generative Adversarial Network (GANs) [88]. Additionally, there remains a need for experimentation to confirm the overwhelmingly simulated works published so far with respect to realistic real-world scenarios involving clock drift, Radio Frequency Front End (RFFE) noise, and other real-world phenomena. Finally, many simulated countermeasures [81, 82, 85] do not consider state-of-the-art adversarial attacks published in the computer vision domain.

1.4 Contributions

Our research is comprised of two different ML-based wireless system topics: secure localization and evasion. Our UL-based wireless localization works leverage the Gaussian distribution of wireless beacon multilateration estimates in order to perform ground-up parameter estimation without the use of heuristic thresholds. Our evasion research injects transmissions with adversarial perturbations that exploit the fragility of trained SL models to fool SL-based eavesdropping radios while minimizing the communication loss of a receiver. See Table 1.3 for a summary of technical challenges proposed by the state-of-the-art and our contributions to those areas of research.

1.5 Contents & Structure

This dissertation is structured as follows: chapter 2 overviews the basics of the UL and SL algorithms used in the research contribution sections. Specifically, CNNs and related SL algorithms are presented by Section 2.1, guidelines for implementing SL by Section 2.2, DPGMMs and related UL algorithms by Section 2.3, and adversarial perturbations by Section 2.4. Our research contributions are then presented by chapters 3-4, and a broader discussion of this dissertation's findings is presented in chapter 5.

Table 1.3: A listing of the challenges faced by the state-of-the-art solutions to the research problems discussed in our dissertation, and our contributions to these problems.

Topic	References	Challenges	Contributions
Ground-up localization using multilateration analysis	[55–58]	→ Transmitters with directional antennas	
	[56–58]	→ Frequentist, heuristic threshold-based classifications	→ A Bayesian approach
	[56–58]	→ Require non-opportunistic signals that contain timing and location information in conjunction with range or location estimates	→ Uses multilateration estimates only
	[56–58]	→ Gaussian hypothesis testing infers only one binary unknown parameter, the presence of a second transmitter for a population of detections	→ Classify sub-populations of a population of detections, estimate number of sub-populations, estimate mean and covariance of each sub-population
Crafting adversarial perturbations to avoid SL-based eavesdroppers	[76, 77]	→ Add separate OTA perturbations robust to realistic synchronization errors	
	[79, 80, 89]	→ Adversarial perturbations large enough to be robust to channel effects disrupt communications.	→ Two novel perturbation crafting constraints
	[79, 80, 89]	→ Data set is too over sampled, too specific, too collinear, and contains signal, perturbation, and noise ratio computation errors	→ Data set that fixes these issues
	[79, 80]	→ Non-optimization based improvements to communication, power, and adversarial loss	→ Pulse shaping perturbations, constructive-only perturbations
	[80]	→ Countermeasure case study, specifically UL-based detection of OTA perturbations	→ Adversarial training case study
	[76, 77, 79, 80, 89]	→ OTA perturbation experiment	→ USRP N210 experiment

1.6 Related Works

The chapters of this dissertation contain figures and writings adapted from drafts or final versions of the following peer-reviewed publications:

- **K. McClintick**, G. Wernsing, P. Ferreira, A. Wyglinski, "Parameter Estimation and Classification via Supervised Learning in the Wireless Physical Layer: A Survey and Tutorial." IEEE Access, 2021.

- **K. McClintick**, A. Wyglinski, "Reproduction of Evaluating Adversarial Evasion Attacks in the Context of Wireless Communications and Convolutional Radio Modulation Recognition Networks," Proceedings of the 2021 CPS-IoTBench Conference, Spring 2021.

- **K. McClintick**, M. Page, T. Wickramarathne, A. Wyglinski, "Machine Learning-Based Roadside Vehicular Traffic Localization via Opportunistic Wireless Sensing," Proceedings of the 2019 IEEE GlobalSIP Conference.

Additionally, the following submitted works have been incorporated in this dissertation:

- **K. McClintick**, J. Tolbert, A. Wyglinski, "Physical Layer Wireless Localization of an Unknown Number of Beacons using Unsupervised Learning," IEEE Open Journal of Vehicular Technology, 2021.

- **K. McClintick**, J. Harer, B. Flowers, W. Headley, A. Wyglinski, "Physical Eavesdropper Evasion: Signal Dependent Perturbation Design and Adversarial Training" IEEE Transactions on Cognitive Communications and Networking, 2021.

Chapter 2

An Overview of Related Machine Learning Topics

This section presents the basic underlying concepts of the ML algorithms used in our contribution chapters. Specifically, we present a statistical approach to neural networks and SL in Section 2.1 and guidelines for their design and training in Section 2.2 to provide the reader with a foundation for chapters 3-4. Next, we describe the algorithmic assumptions and implementation details of sampling mixture models in Section 2.3 to provide the reader with a foundation for chapter 3. Finally, we present the theory, and taxonomy of adversarial perturbations in Section 2.4 to prepare the reader for chapter 4.

2.1 SL Overview

The core mechanic of a NN is regression. We begin the derivation of a NN by presenting the Least Squares Regression (LSR) algorithm to establish terminology, model assumptions, and key statistical concepts such as likelihood, unknown parameter estimation, parametric models, and training.

2.1.1 LSR

LSR is a data-fitting solution that is determined by minimizing the squared sum of residuals on training data such that a test data set may have an unknown parameter estimated via a set of noisy observed features. The true value of an unknown parameter y, or ground truth, can be estimated from noisy samples of data $x = s + \epsilon$, such that a linear function can estimate the unknown parameter as:

$$\hat{y} = f(x, w, b) = x^T w + b = [1|x]^T [b|w], \tag{2.1}$$

where b is a scalar and all vectors x, w are length p column vectors; with the number of input features equal to p. The bias b is consolidated into the weights matrix w as a unit input 1 is multiplied with a bias b, often written instead as w_0. This linear regression model is a predictive algorithm with the following assumptions:

- A linear relationship between data labels and input features;
- Normality of each input feature ($\epsilon \sim \mathcal{N}(0, \sigma^2)$);
- Low multicollinearity, or that each input feature X of the same sample s is independent;
- Low auto-correlation, or that each sample is independent from the others;
- Homoscedasticity, or that the variation ϵ of data is constant over samples;
- Statistical equivalence between exclusive training and testing data partitions, where training data is used to solve for parameters, testing data is used to evaluate the quality of trained parameters, and partitions are exclusive sets.

These significant constraints provide a starting point for the development of more applied models, which will be studied as they are introduced throughout this chapter.

One approach to finding the best weights w for more than two training samples (Figure 2.1) is to pick the values that are most likely to re-create the dataset. This is known as the probabilistic approach to optimization.

If the data is subject to random variation described by a Gaussian random variable $y \sim \mathcal{N}(x^T w, \sigma^2 I)$, the likelihood of N states is the product of N Gaussian PDFs:

$$L(y|x, w, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{N/2}} e^{\frac{1}{2\sigma^2} \sum_{i=0}^{N-1} (y_i - x_i^T w)^2}, \tag{2.2}$$

where the likelihood decreases as $N \to \infty$. This can present an issue in computing, as likelihoods become numerically unstable, meaning they are roughly on scale with the resolution of the floating-point values computers use to represent numbers. For instance, a

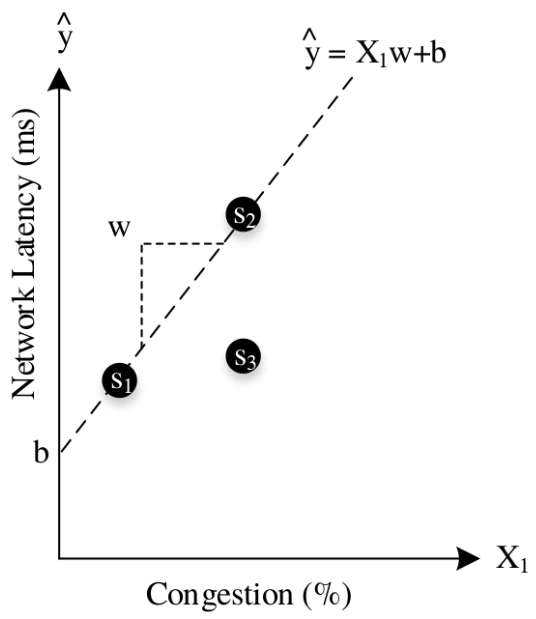


Figure 2.1: An illustration of how fitting a linear model (w, b) for accurate network latency predictions has a clear solution for two samples s_1, s_2 until a third sample s_3 is observed. A linear solution can be given by MLE, where the probability of recreating the data $x \in \{s_1, s_2, s_3\}$ is maximized choosing w, b.

32-bit single precision float value uses 24 fraction bits, allowing a fractional resolution of $\log_{10}(2^{24}) \approx 7.225$. This means, as likelihood approaches $L(y|x, w, \sigma^2) = 10^{-7}$, a computer will begin to quantize the computed likelihood to the nearest valid value, resulting in a shift of similar magnitude as the likelihood. These large rounding errors make likelihood computations inaccurate towards selecting high-probability weights. The natural logarithm of the likelihood is used to mitigate this numerical instability. Since the log of probabilities gives negative values, the log-likelihood is often multiplied by -1 to compute the Negative Log Likelihood (NLL), which maintains positivity given the goal to maximize the likelihood, NLL reverses the objective argument from maximization to minimization, which is given by:

$$\begin{aligned}&\arg\min_w\{l(y|x, w, \sigma^2)\} = \\ &\arg\min_w\{-\frac{N}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=0}^{N-1}(y_i - x_i^T w)^2\}.\end{aligned} \tag{2.3}$$

Since $-\frac{N}{2}\ln(2\pi\sigma^2)$ is not a function of w, the values for w that are most likely to recreate the training data are:

$$\arg\min_{w} \sum_{i=0}^{N-1} (y_i - x_i^T w)^2. \tag{2.4}$$

This solution is given the name least squares regression for its objective of minimizing the squared error between predicted and actual outcomes inferred by noisy observations. Similarly, the NLL function of a set of data is referred to as the objective function and loss function. A LSR model is evaluated by this metric, where if predictions $x_i^T w$ and ground truths y are similar, the model is performing well. The NLL computed assuming the linear model can sometimes look like Figure 2.2.

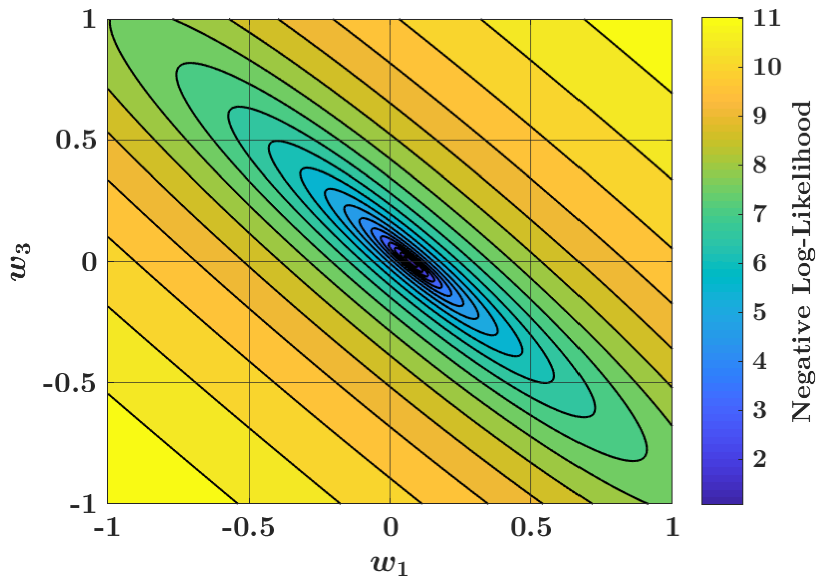


Figure 2.2: The lowest NLL is given for about $w_1 = 0.01$ and $w_2 = -0.005$. Since the data model (2.1) is linear, its NLL is convex. This attribute can be leveraged to avoid iterative approaches that compare predictions made by the infinite range of inputs $-\infty < w_1, w_2 < \infty$ to some loss function. Instead, a closed form solution exists for w.

Visually, the NLL in Figure 2.2 has a single minimum, or is convex. For a convex function, every local minimum is the global minimum, or formally $\frac{\delta^2}{\delta x^2} f(x) \geq 0, \forall x$. Linear data always give convex NLL contours, and as such the global minimum can be solved for in closed form where the slope of the line is zero:

$$\frac{\delta}{\delta w} l(y|x, w, \sigma^2) = 0, \tag{2.5a}$$

$$\frac{\delta}{\delta w}(y - Xw)^T(y - Xw) = 0, \tag{2.5b}$$

$$X^T(Xw - y) = 0, \tag{2.5c}$$

where solving for w using input feature matrix $X \in \mathbb{R}^{N,p}$ yields:

$$\hat{w} = (X^T X)^{-1} X^T y. \tag{2.6}$$

Intuitive use of SL algorithms requires an understanding of a wide array of statistical topics including regression, concavity, Bayes theorem, MLE, chain rule, and gradient descent. Consequently, the NN algorithm — a fundamental building block for many SL algorithms — will be derived and defined by first discussing the LSR algorithm and then softmax regression.

NN models estimate or classify unknown parameters y by observing data matrices x with N samples and some dimensionality p. Each element $x_i, i \in p$ is also called a feature. The set of all values data can take $x \in \Omega$ is called the state space of the data.

Given the continuous nature of many real-life state-spaces, it is impossible to train a model with every value in Ω. When the state-space is continuous, a scientist may choose to parameterize a predictive model, or utilize a set of weights w so that a continuous trend-line can be approximated from just a few observed feature states. Such models allow generalization under the assumption that states observed are descriptive of the underlying data distribution.

In reality, few estimation problems can be approached effectively using a linear model. Non-linear, polynomial regression solutions present a possible solution, where:

$$\begin{aligned} \hat{y} =& w_0 + w_{1,1}X_1^1 + ... + w_{1,p}X_1^p + \\ & ... + w_{n,1}X_n^1 + ... + w_{n,p}X_n^p, \end{aligned} \tag{2.7}$$

although these models assume the input features follow a specific polynomial trendline. If the assumed polynomial trendline is too low of an order (*i.e.,* linear when the true trendline is quadratic) the model is said to under fit and residuals will be high. Conversely, an assumed polynomial trendline with too high of an order (*i.e.,* cubic when the true trendline is quadratic) is said to over fit, and will also result in a high residual.

2.1.2 Logistic/Softmax Regression

Logistic regression was developed as an alternative algorithm to polynomial regression with greater resilience against both over- and under-fitting a model to a set of data [90]. Robustness to over- and under-fitting is achieved by "boosting" [90], a set of small, serialized, and parallel nonlinear regression models, or neurons (Figure 2.3), solve a union of small estimation problems to solve a high-dimension estimation problem. At last we arrive at the distinction between a regression model and a neural network: a regression model becomes a neuron when it joins a network of other regression models by serializing and/or parallelizing them, and a neural network is the set of serialized and parallelized regression models. Additionally, a neural network is considered deep if there exist more than two serial layers of neurons, and considered shallow otherwise.

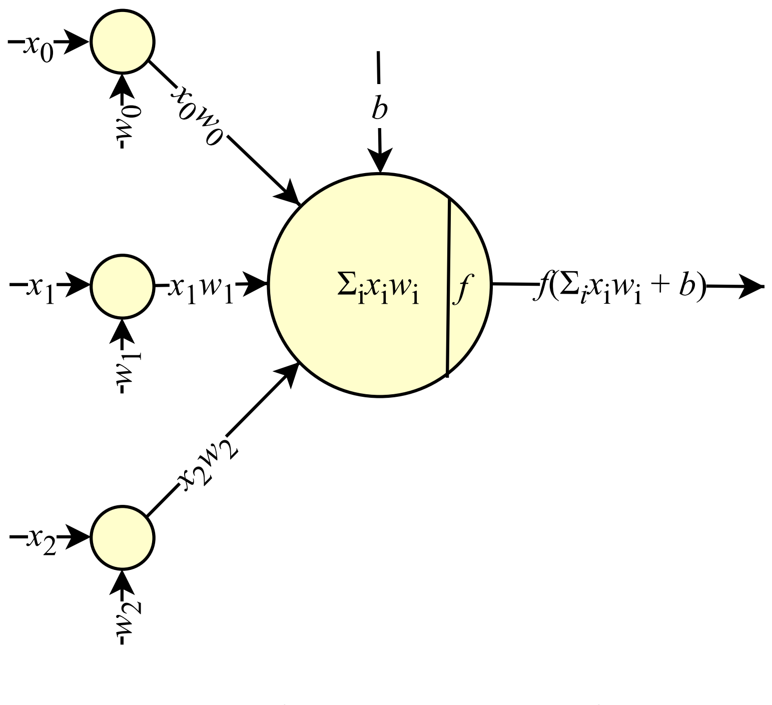


Figure 2.3: A neuron is composed of a biased dot product of inputs and weights. In this mathematical model inspired by the biological neuron, the neuron is changed from a linear regression model to a non-linear one by the arbitrary activation function f, which in logistic regression is given by the sigmoid function.

In logistic regression, each layer or parallel set of neurons performs two computations, including the linear logit:

$$z = x^T W + b, \tag{2.8}$$

and the non-linear activation:

$$a = f(z). \tag{2.9}$$

In logistic regression, the sigmoid activation function:

$$f(x) = \frac{1}{1 + e^{-(x^T w + b)}}, \tag{2.10}$$

is chosen as the activation function to satisfy the non-constructive universal approximation proof [91].

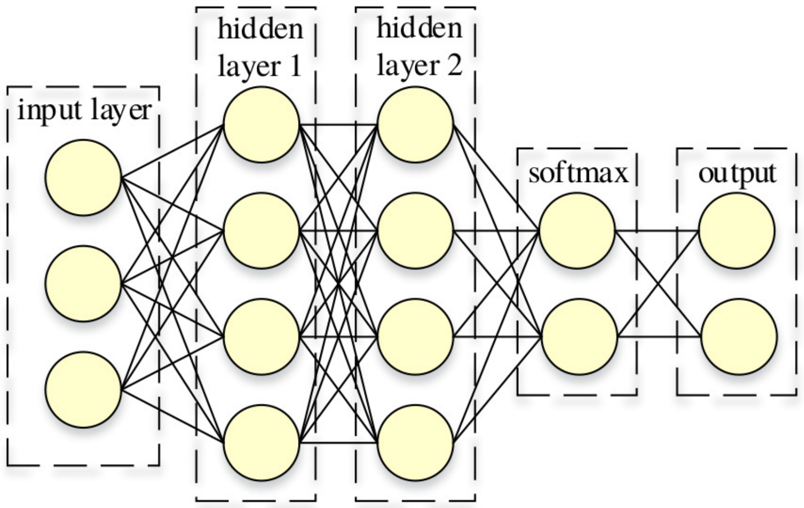


Figure 2.4: A binary class softmax regression model with two serialized neuron layers $i = 1, 2$. Each layer has four parallel neurons (Figure 2.3), and input data has three states.

Softmax regression (Figure 2.4) is an algorithm similar to logistic regression, designed for detection tasks instead of estimation. This change in objective is made by replacing the sigmoid activation function of the final serial layer of neurons with the softmax activation function, which is a function of the linear logit of every neuron in the current layer:

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}}, i = 1, ..., C. \tag{2.11}$$

The activation is designed to be positive and monotonically increasing for the purpose of gradient descent, presented by equation (2.13), and is normalized so that each activation may be interpreted as a pseudo-probability, or the probability of a data sample belonging to class $i = 1, ..., C$, where a class is an exclusive outcome from a set in detection theory. In NN models, the computation of a model's output is also called a forward pass, because all computations flow sequentially through the model. In the softmax regression model, the loss function is computed by NLL, defined as categorical cross entropy loss:

$$\begin{aligned} f_{CE} &= -\log P(y|x, w), \\ f_{CE} &= -\sum_{i=1}^{n}\sum_{k=1}^{C} y_{i,k} \log \hat{y}_{i,k}, \end{aligned} \tag{2.12}$$

where the minimum NLL weights are solved. Unlike in LSR, a non-convex loss function gradient makes a closed form solution impossible due to the existence of local minima. While many iterative approaches exist to locate the global minimum in the loss function (*i.e.*, Newton's method), the dominant method in recent years has been the SGD method (Figure 2.5).

In SGD, the weights are iteratively improved by subtracting the loss functions gradient with respect to the weights:

$$w = w - \eta \nabla_w f_{CE}, \tag{2.13}$$

where updates are reduced by a learning rate $\eta < 1$. Each update using all training data (Gradient Descent) or set of updates making use of all mini-batches of the training data (SGD) is defined as an epoch. Training sessions are commonly performed for some number of defined epochs, n_e. The computation of a models' gradients is also called a backwards pass because all computations flow sequentially from the model output to the input due to chain rule. For a single layer softmax regression model, we have the following expression given equations (2.8, 2.11, 2.12):

$$\nabla_w f_{CE} = \frac{\delta f_{CE}}{\delta a}\frac{\delta a}{\delta z}\frac{\delta z}{\delta w}, \tag{2.14}$$

where:

$$\nabla_w f_{CE} = -\frac{1}{n}\sum_{i=1}^{n} x_i(y_i - \hat{y}_i). \tag{2.15}$$

Armed with equation (2.15), we can iteratively update the weights of the single-layer soft-

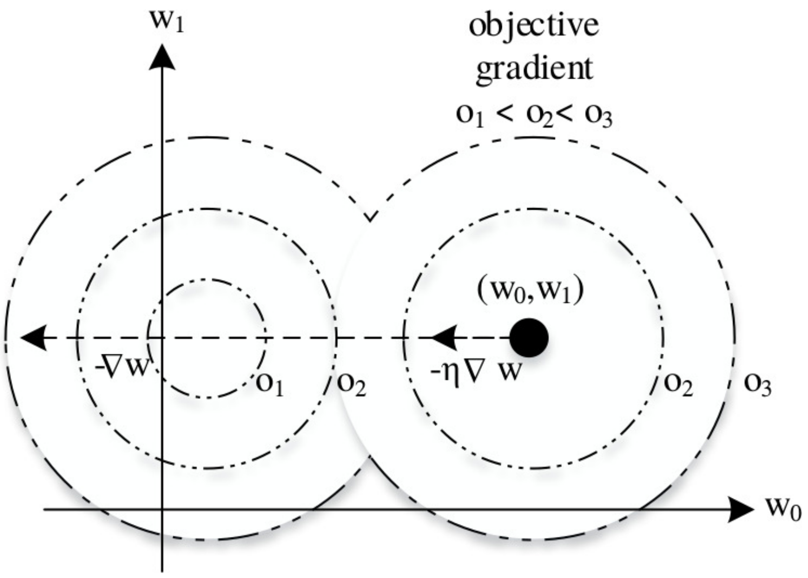


Figure 2.5: An illustration of a two weight softmax regression objective function and its gradient ∇_w. Nonlinear models have local minima/maxima (*i.e.*, o_2). A learning rate scalar η is needed to reduce weight updates, as large weight updates can cause the objective function to go to infinity or significantly increase the number of updates needed to reach the global objective minimum o_1. If η is too small, SGD becomes computationally expensive and can cause the weights to be unable to move away from local minima.

max model until the cross entropy loss has converged. However, gradients are tedious to compute analytically so they are often computed numerically using software libraries. Such libraries make use of the centered difference formula:

$$\frac{\delta f}{\delta x} = [f(x+h) - f(x-h)]/2h, \tag{2.16}$$

where the spacing $h << 1$ (typically 10^{-5} [92]). For insight into this process, mathematical circuit diagrams can show how software libraries use numerical methods to compute gradients for any NN architecture.

The softmax regression model has several improvements over LSR [93]. There no longer exists a linear relationship between the data and labels. Additionally, the residuals ϵ are no longer required to be normally distributed. Finally, homoscedasticity is not required. Guidelines for softmax regression training and testing are given by Algorithm 1 and Algorithm 2.

Why is there a need for ML algorithms given the existing Bayesian and frequency-based estimators and classifiers? The answer is that many ML models are less constrained.

Algorithm 1 Softmax model training protocol [92]

```
1: procedure Given training data $X$ with labels $Y$, a model with some number of layers $L$, learning rate $\eta$, and number of epochs $n_e$:
2:     initialize model parameters $w, b$
3:     for $n_e$ do
4:         for each $x$ in $X$, $y$ in $Y$ do
5:             for layer $l$ in $L$ do
6:                 compute linear logits $z_l(x)$
7:                 compute sigmoid $a_l(z_l)$
8:             end for
9:             compute softmax $\hat{y} = \arg\max(a_i(z_L))$
10:            compute loss $f_{CE}(y, \hat{y})$
11:            compute gradients $\frac{\delta}{\delta w} f_{CE}$, $\frac{\delta}{\delta b} f_{CE}$
12:            update model parameters $w, b$
13:        end for
14:    end for
15: end procedure
```

Algorithm 2 Softmax classification prediction protocol [92]

```
1: procedure Given test data $X$, trained weights $w$ and bias $b$ of a model with some number of layers $L$:
2:     for each $x$ in $X$, $y$ in $Y$ do
3:         for layer $l$ in $L$ do
4:             compute linear logits $z_l(x)$
5:             compute sigmoid $a_l(z_l)$
6:         end for
7:         compute softmax $\hat{y} = \arg\max(a_i(z_L))$
8:     end for
9: end procedure
```

Consider the highly constrained signal detection problem where two known discrete time signals $s_0, s_1 \in \mathbb{C}^n$ can be inferred by the noisy signal:

$$x = s + \epsilon, \tag{2.17}$$

where $\epsilon \sim \mathbb{N}(0, \sigma^2 I)$ is Additive White Gaussian Noise (AWGN). If the signals s_0, s_1 and noise σ^2 are known, this coherent problem (known signals, known noise) can be modeled optimally with a Neyman-Pearson decision rule [94], derived by maximizing P_D for a given

$P_{fa} = \alpha$. Given the NP criterion decision rule [94]:

$$\rho^{NP} = \arg\max_{\rho \in D} P_D(\rho), \tag{2.18}$$

the alternate hypothesis for this signal detection example is decided as:

$$\rho^{NP}\left(L(x) = \frac{P(x; s_1)}{P(x; s_0)} < \gamma\right) = 1, \tag{2.19}$$

where the decision threshold γ is solved for given:

$$P_{fa} = \int_{x:L(x)>\gamma} P(x; s_0)dx = \alpha. \tag{2.20}$$

Alternatively, a Bayes decision rule $\delta^{B\pi}(L(x)) \in 0, 1$ can be derived if non-equal priors exist (the prior is the set of probabilities $\pi = \{P(s_0), P(s_1)\}$, and $P(s_0) \neq P(s_1)$), by deciding [94]:

$$\rho^{B\pi}\left(\frac{P(x; s_1)}{P(x; s_0)} > \frac{P(s_1)}{P(s_0)} = \gamma\right) = 1. \tag{2.21}$$

If the noise variance (σ^2) or signal hypothesis' (s_0, s_1) are unknown, the problem becomes parametric, meaning one or more parameters are unknown. This use of the word parametric differs from the data scientist's definition of parametric and non-parametric, which describe the presence or absence of weights in an estimator model. In the case of a parametric problem, statisticians generally take one of three approaches [94]. The first approach is to apply the Neyman-Pearson rule with some significance level α to identify a uniformly most powerful or locally most powerful decision rule. The second approach is to determine a Bayesian decision rule if priors and cost assignments exist. The third approach is to use the generalized likelihood ratio test, which rules s_1 if [94]:

$$\frac{\int_{x \in X/X_0} P(s_1|x)dx}{\int_{x \in X_0} P(s_0|x)dx} > v, \tag{2.22}$$

where:

$$P_{fa} = \int_{x:L(x)>v} P(x; s_0)dx = \alpha, \tag{2.23}$$

which can be implemented if the MLE for all unknown parameter(s) can be identified. All of these solutions require a Probability Density Function (PDF) describing the likelihood of observations.

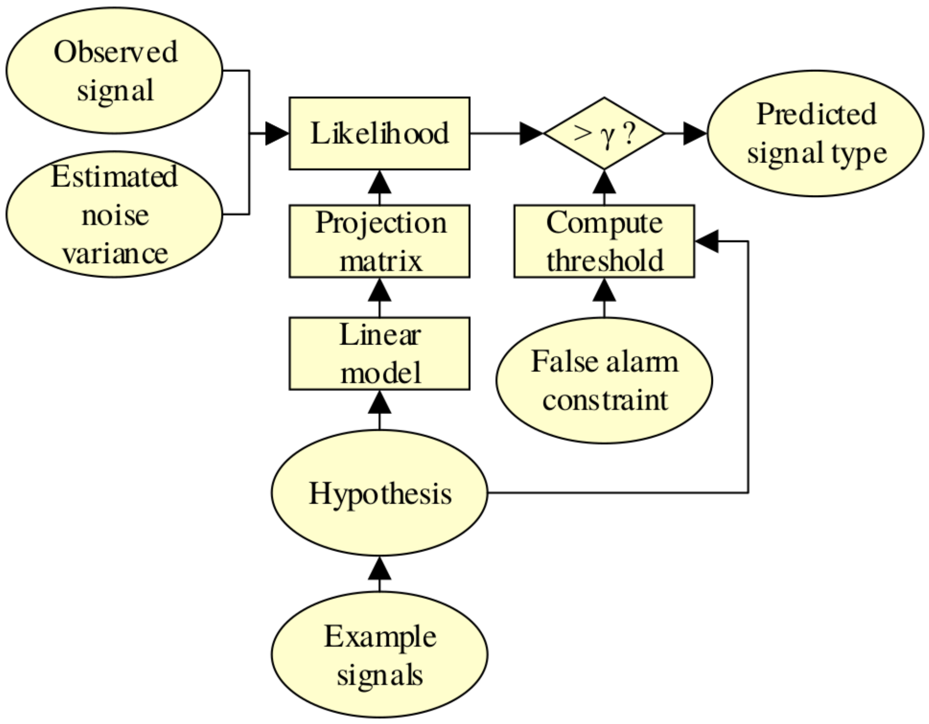
In application, the noise and signals can have unknown, non-linear, and time-varying PDFs due to both intentional alterations (*e.g.*, ACM, pulse shaping) and unintentional alterations (*e.g.*, amplifier non-linearities). Given these complications, highly structured solutions become difficult or impossible to use (Figure 2.6).

ML algorithms that utilize NNs provide powerful solutions when data is available, but their PDF is difficult to model, as a single layer NN has been proven to be a universal function approximator [91]. This non-constructive proof asserts and proves that any bounded function $f(x)$ can be approximated as $\hat{f}(x)$ within a finite tolerance δ, or $|f(\hat{x}) - f(x)| < \delta$.

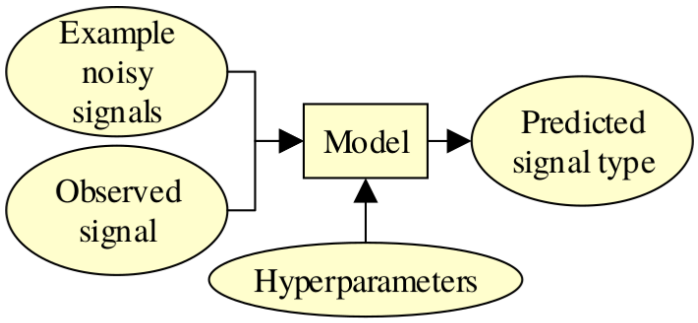
For example, if the inputs $x = -2, y = 5, z = -4$ are stored by the state $q = x + y$ and compute an output $f = z \times q$ (Figure 2.7), The forward pass values can be computed as $q = 3$ and $f = -12$. The output's gradient with respect to itself would be given as $\frac{\delta f}{\delta f} = 1$. Since f is a function of q and z, its gradient is given by $\nabla f = \left[\frac{\delta f}{\delta q}, \frac{\delta f}{\delta z}\right] = [z, q] = [-4, 3]$. The state q is dependent on x, y and has a gradient given by $\nabla q = \left[\frac{\delta q}{\delta x}, \frac{\delta q}{\delta y}\right] = [1, 1]$. The resulting gradient for each input is then $x = \frac{\delta f}{\delta f} \times \frac{\delta f}{\delta q} \times \frac{\delta q}{\delta x} = -4$, $y = \frac{\delta f}{\delta f} \times \frac{\delta f}{\delta q} \times \frac{\delta q}{\delta y} = -4$, and $z = \frac{\delta f}{\delta f} \times \frac{\delta f}{\delta z} = 3$.

2.1.3 CNN

While logistic/softmax regression models allow for higher order predictions than LSR and a lower chance of under/over fitting than polynomial regression, their computational cost does not scale well with a large number of inputs and many layers of neurons. To exemplify this, consider a data set of spectrograms where each of 100 time-steps is computed as a 512-bin FFT $x \in \mathbb{R}^{512,100}$. A single layer softmax regression model with 25 neurons would require 1,280,000 weights and 25 bias terms. The computation of the first layer's linear logit $z = x^T w + b$ would require 2.1×10^{18} computations given an $\mathcal{O}(n^3)$ for matrix multiplication and $\mathcal{O}(n)$ for element-wise addition ($n = 1.28 \times 10^6$). The CNN was designed to address this computational load by constraining the softmax regression model to use the same set (filter) of weights for each input feature. Consider the same data and model again, where instead of 25 neurons, a set of 25 filter weights $w \in \mathbb{R}^{f_h, f_w}$ are used to compute linear logits, where filter height $f_h = 5$ and width $f_w = 5$ span 5 frequency bins and 5 time steps. The weights are convolved over the input data; the r^{th} row c^{th} column of the linear logit is now computed as:



(a) Linear Signal Classifier - MFLRT



(b) Nonlinear Signal Classifier - SL

Figure 2.6: Multifamily Likelihood Ratio Test (MFLRT) [1] (a) and SL (b) algorithms for multiple signal classification. MFLRT is a linear model, and is consequently suboptimal for nonlinear signals. Additionally, MFLRT requires difficult analytical derivations for decision thresholds γ that are a function of signal PDFs and classification objectives (hypothesis), and requires knowledge of AWGN variance (variance can be estimated as $\hat{\sigma}^2$ at a penalty to performance) to calculate likelihood. SL algorithms require no analytical derivations to use, and have been shown to be a robust, scalable, and dependable classification model when the model structure and hyper parameters are chosen carefully through a combination of trial-and-error and domain knowledge.

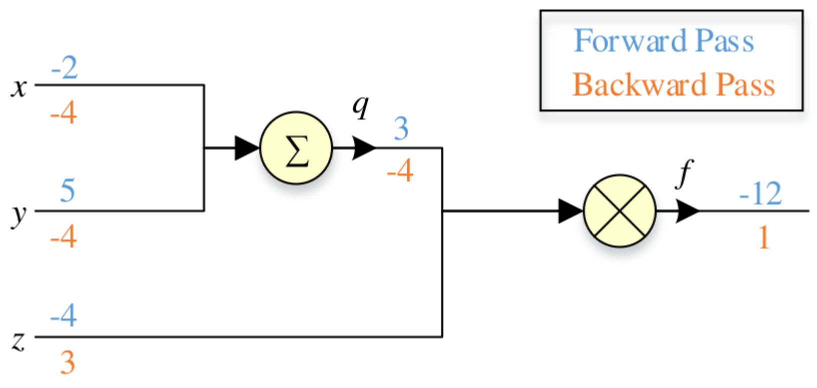


Figure 2.7: A mathematical circuit model showing forward pass values being computed using gates ($\sum, \times$) and example inputs. Backward pass values are then computed by applying the chain rule recursively.

$$z_{r,c} = \sum_{i=-f_h/w}^{f_h/2} \sum_{j=-f_w/2}^{f_w/2} x_{r+i,c+j} w_{i,j}, \tag{2.24}$$

and the dimensions of the linear logit $z \in \mathbb{R}^{z_h, z_w}$ are:

$$\begin{aligned} z_h &= (x_h - f_h) + 1, \\ z_w &= (x_w - f_w) + 1. \end{aligned} \tag{2.25}$$

The resulting number of matrix multiplications (Figure 2.8) between the filter and input is $z_h \times z_w = 48768$, or 7.62×10^8 computations ($n = 1.56 \times 10^4$) this time, which is 10 orders of magnitude fewer computations.

By using a filter constraint on linear logit computations, we have introduced an assumption to the predictive model, called equivariance [95]. Equivariance is the concept that given some transform $f(x)$ on the data x and the convolution $z(x)$, then $f(z(x)) = z(f(x))$ (Figure 2.9). If equivariance is not held, a CNN will perform worse than a softmax regression model.

CNNs use many different techniques to enforce equivariance. A simple approach is to use multiple filters per convoluational layer, similar to how a neuron layer has multiple neurons computed in parallel. In practice this allows each filter to be trained to recognize different representations of a signal (*i.e.*, a signal observed at different frequency bands in a frequency selective channel). A mandatory form of equivariance enforcement is pooling (Figure 2.10), which enforces data equivariance to data scaling and shifting using a window

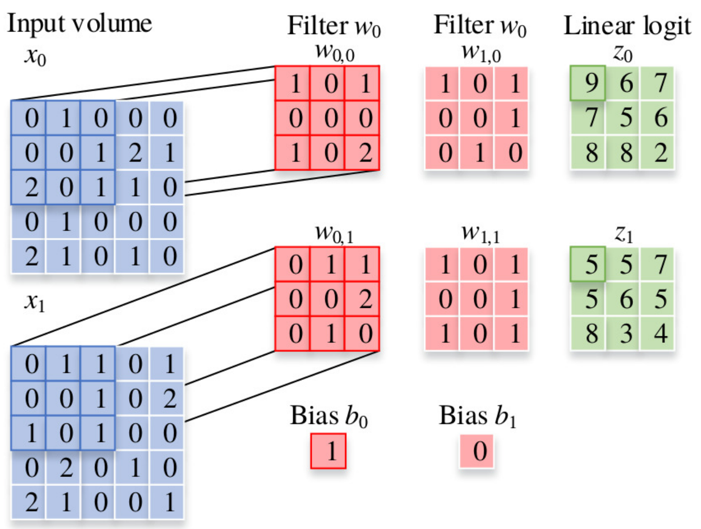


Figure 2.8: A convolutional layer computing outputs $z_{r,c}$ across two input dimensions for some arbitrary data. Representing data with more than one dimension in wireless communications applications often is used to correlate mathematical transforms on data to their labels (*i.e.*, real/imaginary components, absolute value, difference between features).

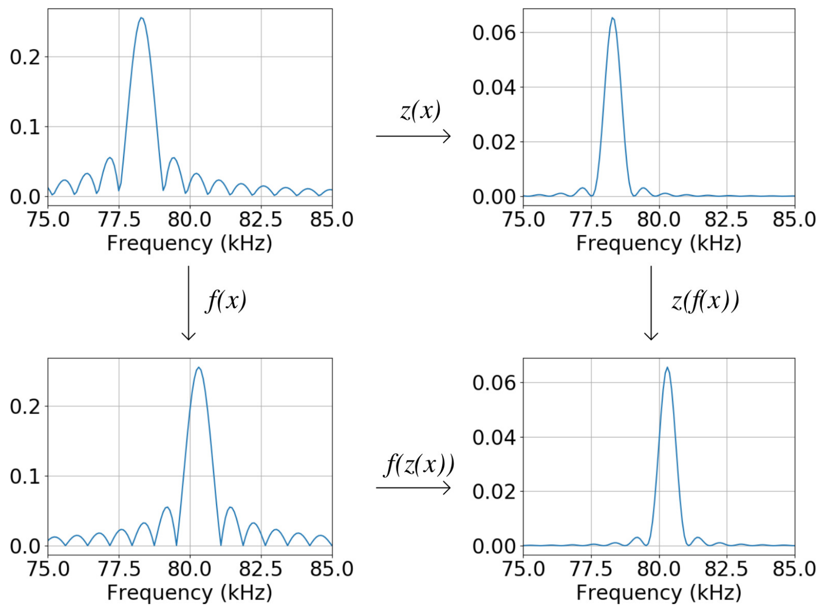
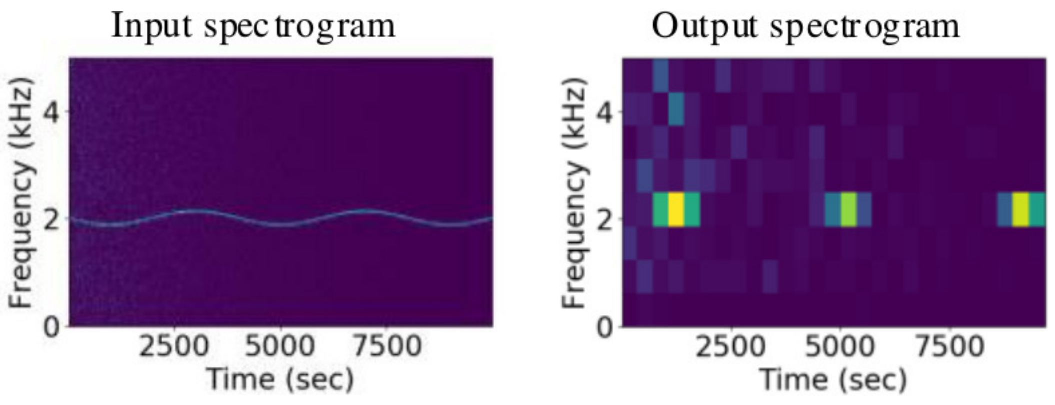


Figure 2.9: A signal x is shifted in frequency in a flat frequency channel (coherence bandwidth of the channel is larger than the bandwidth of the signal) by $f(x)$ and convolved with a convolutional layer's weights by $z(x)$. In a frequency selective channel, the equivariance assumption would no longer hold, as different frequency bins would be attenuated by different amounts.

of size $p_{r,c} \in \mathbb{R}^{p_h, p_w}$:

$$p_{r,c} = \max\{x_{r-p_h/2, c-p_w/2}, ..., x_{r+p_h/2, c+p_h/2}\}, \tag{2.26}$$

meaning $p_{r,c}$ does not change if a signal x (Figure 2.9) is shifted by a small amount in amplitude, phase, frequency, or time, depending on the domain of the data. The size of the pooling window affects how large scaling and shifting can be while still holding equivariance, larger shifts can be made equivariant with larger pooling windows $p_h, p_w \to \infty$. However, this comes at the cost of lost information, as more elements are discarded with larger pooling windows. Consequently, it is a common practice to design CNNs with repeated layers of small convolution and pooling windows so that filters can have a redundant set of equivariance and receptive field.



(a) signal (b) Pooled signal

Figure 2.10: Max pooling of a 512 bin spectrogram with pooling window of size $p_h = p_w = 8$. Spectrogram displays a sin-sweeping sinusoid. Notice that some key attributes of the signal are maintained, such as the frequency-sweep period. Such features may be highly correlated to model outputs.

The second assumption a CNN model assumes is a sufficient receptive field. An element of the activation $z_{l,r,c}, l = 1, ..., L$ at layer l of a CNN is said to have a receptive field on layer k of size $RF_{l,k}(z_{l,r,c}) \in \mathbb{R}^{rf_h, rf_w}$ defined as the set:

$$\begin{aligned} RF_{l,k}(z_{l,r,c}) =& \{z_{k,r-rf_h/2, c-rf_w/2}, ..., \\ & z_{k,r+rf_h/2, c+rf_w/2}\}, \end{aligned} \tag{2.27}$$

or less formally, all the values influencing that convolution, extending back to the k^{th} layer.

Most commonly, the receptive field is used to describe how much of the input sample ($k = 1$) influences a particular filter or filter element (*i.e.*, the highlighted fields of Figure 2.11 would be the receptive field of a neuron in the $k = 4$ layer). For a CNN to have a sufficient receptive field, filters must be sufficiently large as to learn patterns relevant towards classification. For instance, if a CNN classifies time domain signals with some periodic behavior of period T, and there exist no filters with receptive field $rf_w \geq T$, it may be difficult or impossible to accurately train the CNN.

Convolution is a linear function, so if a CNN is to have the same non-linear capabilities as softmax regression, a similar activation function must be used after computing the linear convolutional logit z. In CNN, Rectified Linear Unit (ReLU) activations [96] can compute each activation $a \in \mathbb{R}^{z_h, z_w}$ from input element x as:

$$a(z(x)) = \max(0, z(x)). \tag{2.28}$$

In models with more than two neuron layers, the weight update $\nabla_w f_{CE} \to 0$ as the number of layers $L \to \infty$ due to the product of many small gradients when computing the weight update by chain rule. The ReLU function was created to mitigate these numerically unstable gradients. If this occurs for a weight with a large receptive field, training can be significantly slowed or even stopped. This phenomenon is called a neuron blackout [92]. Leaky ReLU [97] is an activation function which was developed to mitigate the neuron blackout issue by giving a small slope α for values $x < 0$:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}. \tag{2.29}$$

To aid in making receptive fields sufficient, CNNs are typically terminated with a few dense layers of neurons, without the convolution filter constraint. Flattening layers are required to allow the 1D dense layers to compute their linear logits, where flattening of a pooled 3D layer P with n_f filters to a 1D vector of flattened features $p \in \mathbb{R}^{n_f \times p_h \times p_w}$ is computed as:

$$p = \text{flatten}(P). \tag{2.30}$$

A final consideration for computing convolutional linear logits is zero-padding [92]. Zero-padding is a method of increasing the dimension of features in a CNN after dimension reduction by pooling or convolution, without adding information or noise. The stride of convolution is defined as how many rows or columns the filter is shifted by after each convolution is computed, $z_{r,c}$. These two operations affect the linear convolutional logit dimensions (2.25). After padding with a window $\in \mathbb{R}^{P_h,P_w}$, convolving with some stride $\in \mathbb{R}^{S_h,S_w}$, and pooling, convolutional logit dimensions are computed as:

$$\begin{aligned} z_h &= ((x_h - f_h + 2P_w)/S_h + 1)/p_h, \\ z_w &= ((x_w - f_w + 2P_w)/S_w + 1)/p_w. \end{aligned} \tag{2.31}$$

Consider the CNN architecture in Figure 2.11 as an example of a forward pass through convolutional, pooling, flattening, fully connected, and softmax layers.

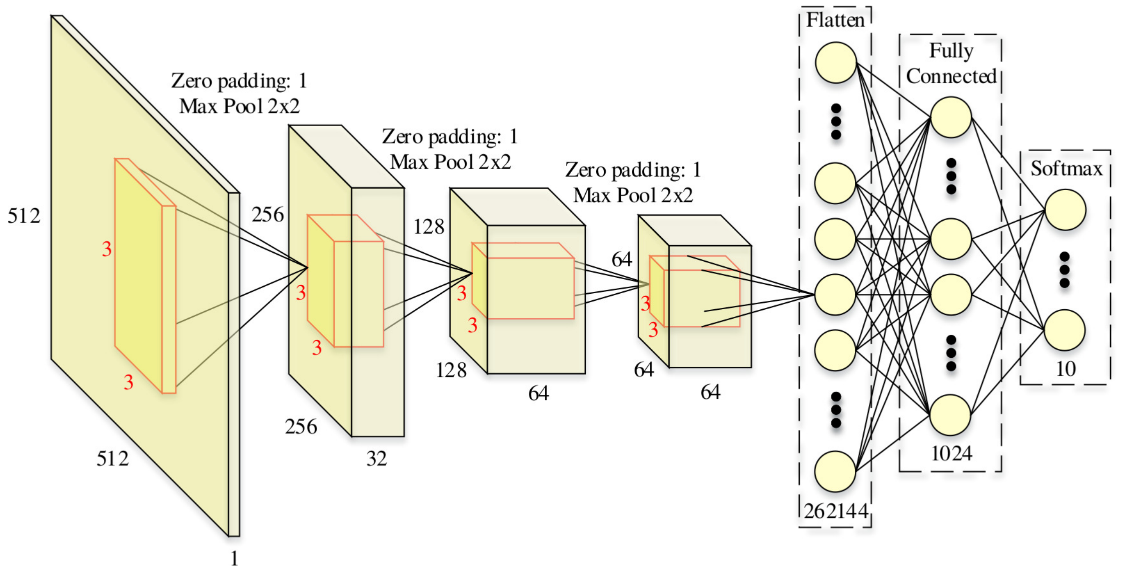


Figure 2.11: An arbitrary CNN with repeated convolutions, padding, and down sampling via max pooling of a 512 by 512 data sample. Layer heights and widths are computed by (2.31), and depths are chosen. In wireless communications, such a sample may represent a 512-bin spectrogram. The signal is classified via a softmax layer to be one of ten categories. Receptive fields of a neuron in the flattening layer on the input sample are highlighted.

Training a model (Algorithm 8) is done by calculating the weights most likely to predict the correct label in the presence of noise. Similar to softmax regression, this is accomplished by minimizing categorical cross entropy loss (2.12). SGD is performed, where gradients are calculated via the chain rule as in logistic/softmax regression. The CNNs testing protocol is described in Algorithm 4.

Algorithm 3 CNN model training protocol [92]

```
1: procedure Given training data $X$, labels $Y$, learning rate $\eta$, training iterations $n_e$
2:     initialize model parameters $w, b$
3:     for $n_e$ do
4:         for each $x$ in $X$, $y$ in $Y$ do
5:             for layer convolution layer $l$ in $L$ do
6:                 for filter $w$ in layer $l$ do
7:                     apply zero-padding if used
8:                     compute convolution $z_l(x, w, b)$
9:                     apply ReLU $a = \max(0, z)$
10:                    compute max pooling $P = \text{pool}(a)$
11:                end for
12:            end for
13:            flatten down sampled features, $p = \text{flatten}(P)$
14:            for each dense layer do
15:                compute linear logit, $z(p, w, b)$
16:                compute activation function $a = \max(0, z)$
17:            end for
18:            compute softmax $\hat{y} = \arg\max(a(z))$
19:            compute loss $f_{CE}(y, \hat{y})$
20:            compute gradients $\frac{\delta}{\delta w} f_{CE}$, $\frac{\delta}{\delta b} f_{CE}$
21:            update model parameters $w, b$
22:        end for
23:    end for
24: end procedure
```

2.2 SL Guidelines

Implementing an SL algorithm requires the consideration of several design factors, including the depth and width of the CNN and RNN, the ordering, type, and parameters of the layers in a CNN, the SVM kernel type and parameter choice, and the SGD parameters. This section covers the basic concepts and practical considerations for an SL implementation.

2.2.1 Weight Updates

The task of using SGD to compute the global loss minimum is non-trivial. Several versions of SGD have been developed [92, 98–101], each placing different emphases on how to decrease the magnitude of weight errors, perform fewer calculations, and lessen the

Algorithm 4 CNN classification prediction protocol [92]

```
1: procedure GIVEN TRAINED MODEL PARAMETERS $w, b$, TEST DATA $X$
2:     for each $x$ in $X$ do
3:         for layer convolution layer $l$ in $L$ do
4:             for filter $w$ in layer $l$ do
5:                 apply zero-padding if used
6:                 compute convolution $z_l(x, w, b)$
7:                 apply ReLU $a = \max(0, z)$
8:                 compute max pooling $P = \text{pool}(a)$
9:             end for
10:        end for
11:        flatten down sampled features, $p = \text{flatten}(P)$
12:        for each dense layer do
13:            compute linear logit, $z(p, w, b)$
14:            compute activation function $a = \max(0, z)$
15:        end for
16:        compute softmax $\hat{y} = \arg\max(a(z))$
17:    end for
18: end procedure
```

sensitivity to initial parameter choices. In general, standard weight updates are defined as the weight matrix w changing after each gradient calculation:

$$w_{(i+1)} = w_{(i)} - \eta \nabla_{w_{(i)}}, \tag{2.32}$$

where the hyper-parameter η is the learning rate, and $\nabla_{w_{(i)}}$ is the current weight gradient.

Typically, weight updates are halted after training over a pre-determined number of epochs, or after the gradient becomes smaller than a convergence threshold. Additional stop conditions can be implemented, such as convergence patience, where the gradients must be below a convergence threshold for a sequential number of epochs before stopping. Weight values converging to local loss minima is avoided by computing gradients from small batch sizes or by using robust weight update techniques (*i.e.*, equation (2.33)). The state-of-the-art theory for understanding loss curves was described by Nakkiran *et al.* [102], who explain and expand on the "classic" bias-variance curve (Figure 2.12). The reference describes how previous loss curve behavior theories expect SL models to be initially "classically under-trained" as the model is trained over epochs. During this period, the loss curves indicate that the models under-fit (*i.e.*, training and validation error are relatively high) until an critical epoch is reached. After this critical point, all future training epochs will show loss curves that indicate the model is increasingly over-fitting, demonstrated by a growing difference between a small training error and relatively large validation error.

However, the authors found that, in practice, if model complexity and duration are large enough to achieve near zero training error, increasing the model complexity or training duration further will eventually decrease testing loss after reaching the "interpolation threshold". This phenomenon is called model-size double descent, or epoch-wise double descent, because a "double descent" shaped test error curve is observed. Finally, the authors show that, by varying the amount of training data available, the location of the interpolation threshold will shift, such that test error can increase if training is performed for the same number of epochs.

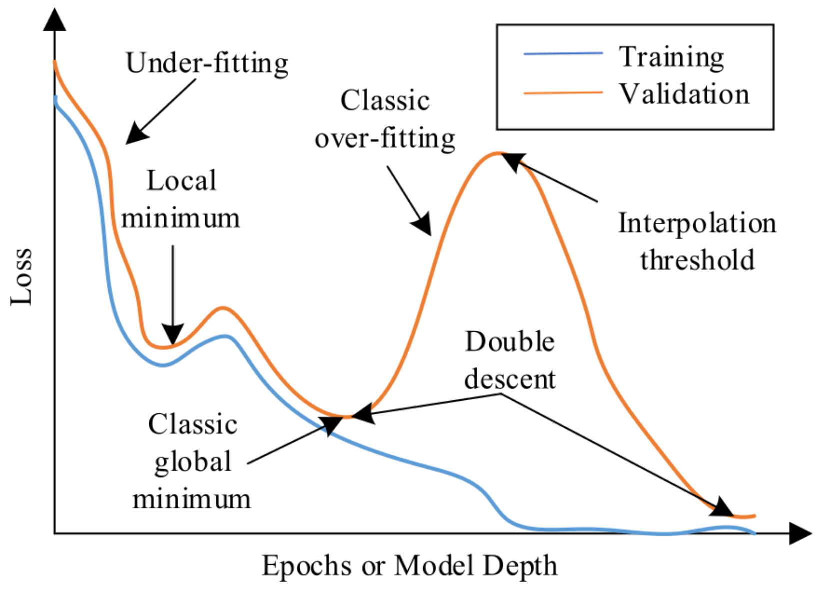


Figure 2.12: When model complexity is small compared to the number of training samples, the test loss follows a U-shaped, "classic" bias-variance tradeoff. A model is classically under-trained when both training and validation loss are high, which can be mitigated by training for more epochs, gathering more data, adding more weights to the model, or validating SGD hyper-parameters (Section 2.2.7). Typically, during SGD a model will under-fit until it begins to over-fit, which is when training loss is low but validation loss is high. Over-fitting can be mitigated by training for fewer epochs, decreasing the number of weights in a model, voting via ensemble learning (Section 2.2.8), or constraining the model with regularization (Section 2.2.3).

One innovation applied to the standard SGD is the development of "momentum" updates that attempt to achieve faster convergence [103]. The term momentum is used as a physical analogy for the optimization problem that is SGD, where the current weights of a model can be thought of as a ball rolling around in a mountain range. Standard SGD updates have been found to be slow to converge for non-spherical gradients, and unable to leave local loss minima "valleys" if the learning rate is too small. Momentum updates, which were inspired by position, velocity, and acceleration models from physics, were developed to solve both of these issues. See Table 2.1 for a survey of momentum-based SGD updates. In such a weight update scheme, the weights w are updated proportionally to their "velocity" as:

$$v_{(i)} = \mu v_{(i-1)} - \eta \nabla_{w_{(i)}}, \tag{2.33a}$$

$$w_{(i+1)} = w_{(i)} + v_{(i)}, \tag{2.33b}$$

where the velocity is initialized as $v_{-1} = 0$, and the momentum μ is a tunable hyper-parameter. The current velocity value is defined as $v_{(i)}$, and $v_{(i-1)}$ is the velocity of the weights from the previous epoch of training over epochs $i = 1, ...$

At the start of training, it is common for the "ball" to descend a steep and tall "mountain", such that its velocity is so high that the weights "overshoot" the loss minima, and ascend the gradient of another loss "mountain" or circle the "valley". The Nesterov momentum [98] is one of many SGD momentum schemes developed to mitigate these issues. It does so by computing velocity that is proportional to the difference in the velocity of the i^{th} and $(i-1)^{th}$ training iteration, making it much harder for weight updates to become unstable (*i.e.*, $\mu v_{(i-1)} >> \eta \nabla_{w_{(i)}}$). For the i^{th} SGD backward pass, the Nesterov momentum update on $w_{(i)}$ is defined as:

$$v_{(i)} = \mu v_{(i-1)} - \eta \nabla_{w_{(i)}}, \tag{2.34a}$$

$$w_{(i+1)} = w_{(i)} + (1+\mu) v_{(i)} - \mu v_{(i-1)}. \tag{2.34b}$$

Another category of SGD algorithms use higher order statistics to enhance the depth and speed of the convergence at the cost of computational efficiency. Newton's method, which is defined as:

$$w_{(i+1)} = w_{(i)} - \frac{\nabla_{w_{(i)}}}{\nabla^2_{w_{(i)}}}, \tag{2.35}$$

utilizes the Hessian matrix $\nabla^2_{w_{(i)}}$, derived from the 2nd order Taylor series expression for $w_{(i)}$, to find the root of the gradient. This second order approach is very powerful because it solves directly for the vertex or loss minima, which allows the weights to converge to the nearest local loss minimum in a single update. Thus, this method requires no learning rate η, nor a learning rate annealing function.

However, there are several shortcomings to Newton's method. If the nearest loss minimia is not the global loss minima, the weights can easily converge to the wrong values without the aid of momentum approaches. Additionally, several "Quasi-Newton" SGD algorithms (see Table 2.2) have been developed using the approximation $\nabla^2_{w_{(i)}} \approx (\nabla_{w_{(i)}})^2$, since the Hessian is too computationally expensive proportional to the number of weights m, *i.e.*, $\mathcal{O}(m^3)$ versus $\mathcal{O}(m^2)$. The approximation $\nabla^2_{w_{(i)}} \approx (\nabla_{w_{(i)}})^2$ only holds true for full batch SGD, which unfortunately requires the entire training data set to be held in temporary memory which is sometimes impossible. As the batch size decreases, the approximation is decreasingly accurate, and the use of Quasi-Newton SGD methods may become inappropriate. Adam [101] is one such Quasi-Newton algorithm, which updates weights as:

$$m_{(i)} = \beta_1 m_{(i)} + (1-\beta_1)\nabla_{w_{(i)}}, \tag{2.36a}$$
$$m_{(i)} = \frac{m_{(i)}}{1-\beta_1^i}, \tag{2.36b}$$
$$v_{(i)} = \beta_2 v_{(i)} + (1-\beta_2)(\nabla_{w_{(i)}})^2, \tag{2.36c}$$
$$v_{(i)} = \frac{v_{(i)}}{1-\beta_2^i}, \tag{2.36d}$$
$$w_{(i+1)} = w_{(i)} + \frac{-\eta \times m_{(i)}}{\sqrt{v_{(i)}} + h}. \tag{2.36e}$$

Adam combines the acceleration/de-acceleration of the learning rate present in RMSprop and the second order statistics of Newtons method. β_1 and β_2 serve as momentum hyperparameters similar to Nesterov's μ. The term h is used to avoid division by zero errors. Finally, $(\nabla_{w_{(i)}})^2$ is used to solve for the nearest vertex. As long as the Hessian approximation holds, Adam can be thought of as the best of both RMSprop and Newton's method. Consequently, Adam has seen significant use in all ML applications.

For large data sets, deep models, and slow-converging SGD algorithms, the interested reader is directed towards parallel and distributed SGD frameworks such as "Hogwild!" [104], which performs well when data is sparse, or when most elements of the data are near-zero valued. Downpour SGD [105] performs a variety of boosting, where many copies of the model are trained on different subsets of data. However, since the models do not communicate with each other, divergence is a constant concern with this framework. Delay-tolerant Algorithms for SGD [106] have been shown to work well by adapting SGD to past gradients and by updating delays between devices. Elastic Averaging SGD [107] links asynchronous model updates to a central "elastic force" variable to allow for more exploration of the parameter space. Finally, Tensorflow [108] splits a computation graph into sub-graphs for distributed SGD.

While some of the algorithms discussed so far autonomously anneal the learning rate, the learning rate η can additionally be annealed in any SGD algorithm by an "annealing function" to increase weight convergence depth and speed. Step decay [92] is one such annealing function, which reduces the learning rate every s weight update steps by a factor α:

$$\eta_{(i)} = \begin{cases} \eta_{(i-1)} - \alpha, & \text{if } i\%s \\ n_{(i-1)}, & \text{otherwise} \end{cases}, \tag{2.37}$$

where $\alpha \in [0, 1]$ and integer s are hyper-parameters. An aggressive annealing function is exponential decay [92]:

$$\eta_{(i)} = \eta_{(i-1)} - \alpha_0 e^{-ki}, \tag{2.38}$$

where the values α_0 and k are hyper-parameters. This annealing function is more dependent on its hyper-parameters and number of update steps than (2.37), as the learning rate can quickly become too small to move, even in steep and convex regions of the gradient.

Inverse decay [92] is another annealing function which was developed as a middle-of-the-road option between (2.38) and (2.37):

$$\eta_{(i)} = \eta_{(i-1)} - \alpha_0/(1 + ki). \tag{2.39}$$

Finally, an annealed gradient noise [116] can be introduced into SGD updates in order to make weight updates more robust to poor initialization:

$$w_{(i)} = w_{(i-1)} + \mathcal{N}\left(0, \frac{\eta}{(1+i)^{\gamma}}\right), \tag{2.40}$$

where near the end of training the added noise will increase training error if not properly annealed by decay term γ, and near the start of training the added noise will not be of benefit if annealed too quickly.

Table 2.1: A summary of momentum-based SGD methods and their improvements upon each other.

Name	**Reference**	**Update**	**Hyper-parameters & Terms**	**Advantages**	**Disadvantages**
Vanilla	[109]	$w_{(i+1)} = w_{(i)} - \eta\nabla_{w_{(i)}}$	$\eta \approx 0.01$: learning rate	Simple, stable	Slow
Momentum	[103]	$v_{(i)} = \mu v_{(i-1)} + \eta\nabla_{w_{(i)}}$ $w_{(i+1)} = w_{(i)} - v_{(i)}$	v: velocity, $\eta \approx 0.01$: learning rate, $\mu \approx 0.9$: momentum	Faster convergence than vanilla	Unstable
Nesterov Momentum	[98]	$v_{(i)} = \mu v_{(i-1)} + \eta\nabla_{w_{(i)}}$ $w_{(i)} + (1+\mu)v_{(i)} - \mu v_{(i-1)}$	v: velocity, $\eta \approx 0.01$: learning rate, $\mu \approx 0.9$: momentum	Stability via "look ahead"	Not adaptive
AggMo	[110]	$v^t_{(i)} = \beta^t v^t_{(i-1)} - \nabla_{w_{(i-1)}}$ $w_{(i+1)} =$ $w_{(i)} + \frac{\eta}{K}\sum_{t=1}^{K} v^t_{(i)}$	$\beta^t = 1 - a^{t-1}, i = 1, ..., K$: damping function, $a \approx 0.1$: damping base, $\eta \approx 0.01$: learning rate, $K \approx 3$: count	Stability via redundant damping	Many hyper-parameters, not adaptive

Table 2.2: A summary of Quasi-Newton SGD methods and their improvements upon each other, some of which may also incorporate momentum-analogous terms.

Name	Reference	Update	Hyper-parameters & Terms	Advantages	Disadvantages
Adagrad	[100]	$w_{(i+1)} = w_{(i)} - \frac{\eta \cdot \nabla_{w_{(i)}}}{\sqrt{(\nabla_{w_{(i)}})^2 + h}}$	$h \approx 0^+$: smoothing term, $\eta \approx 0.001$: learning rate	Adaptive learning rate	Stops early
Adadelta	[111]	$\Delta w_{(i)} = -\frac{RMS[\Delta w_{(i-1)}]}{RMS[\nabla_{w_{(i)}}]} \nabla_{w_{(i)}}$ $w_{(i+1)} = w_{(i)} + \Delta w_{(i)}$	$RMS[\cdot]$: RMS of exponentially decaying squared parameter updates, Δw: weight update, $\gamma \approx 0.95$: decay	Fixes Adagrad's early stopping, no learning rate	Biased updates too small at first
RMSprop	[99]	$E[(\nabla_{w_{(i)}})^2] = \gamma E[(\nabla_{w_{(i-1)}})^2] + (1 - \gamma)(\nabla_{w_{(i)}})^2$ $w_{(i+1)} = w_{(i)} - \frac{\eta \cdot \nabla_{w_{(i)}}}{\sqrt{E[(\nabla_{w_{(i)}})^2] + h}}$	$E[(\nabla_{w_{(i)}})^2]$: exponentially decaying squared parameter updates, $\gamma \approx 0.9$: decay, $\eta \approx 0.001$: learning rate	Fixes Adagrad's early stopping, strong RNN performance	Biased updates too small at first
Adam	[101]	$\hat{m}_{(i)} = \frac{\beta_1 m_{(i-1)} + (1-\beta_1)\nabla_{w_{(i)}}}{1-\beta_1^i}$ $\hat{v}_{(i)} = \frac{\beta_2 v_{(i-1)} + (1-\beta_2)(\nabla_{w_{(i)}})^2}{1-\beta_2^i}$ $w_{(i+1)} = w_{(i)} - \frac{\eta \hat{m}_{(i)}}{\sqrt{\hat{v}_{(i)}} + h}$	m: first moment, v: second moment, $\beta_1 \approx 0.9$: first order decay, $\beta_2 \approx 0.999$: second order decay, $h \approx 0^+$: smoothing term, $\eta \approx 0.001$: learning rate	Adds momentum and bias correction to RM-Sprop/Adadelta	Exponential moving average has poor generalization, unstable momentum
Adamax	[101]	$\hat{m}_{(i)} = \frac{\beta_1 m_{(i-1)} + (1-\beta_1)\nabla_{w_{(i)}}}{1-\beta_1^i}$ $\hat{v}_{(i)} = \frac{\beta_2^\infty v_{(i-1)} + (1-\beta_2^\infty)(\nabla_{w_{(i)}})^\infty}{1-\beta_2^\infty}$ $w_{(i+1)} = w_{(i)} - \frac{\eta \hat{m}_{(i)}}{\sqrt{\hat{v}_{(i)}} + h}$	m: first moment, v: second moment, $\beta_1 \approx 0.9$: first order decay, $\beta_2 \approx 0.999$: second order decay, $h \approx 0^+$: smoothing term, $\eta \approx 0.002$: learning rate	Stabilizes Adam via ℓ_∞ normalization of second moment	Exponential moving average has poor generalization
Nadam	[112]	$\hat{m}_{(i)} = \frac{\beta_1 m_{(i-1)} + (1-\beta_1)\nabla_{w_{(i)}}}{1-\beta_1^i}$ $\hat{v}_{(i)} = \frac{\beta_2 v_{(i-1)} + (1-\beta_2)(\nabla_{w_{(i)}})^2}{1-\beta_2^i}$ $w_{(i+1)} = w_{(i)} - \frac{\eta}{\sqrt{\hat{v}_{(i)}} + h}(\beta_1 \hat{m}_{i)} + \frac{(1-\beta_1)\nabla_{w_{(i)}}}{1-\beta_1^i})$	m: first moment, v: second moment, $\beta_1 \approx 0.9$: first order decay, $\beta_2 \approx 0.999$: second order decay, $h \approx 0^+$: smoothing term, $\eta \approx 0.001$: learning rate	Stabilizes Adam via Nesterov	Exponential moving average has poor generalization
AMSGrad	[113]	$\hat{m}_{(i)} = \frac{\beta_1 m_{(i-1)} + (1-\beta_1)\nabla_{w_{(i)}}}{1-\beta_1^i}$ $\hat{v}_{(i)} = \max(\hat{v}_{(i-1)}, \beta_2 v_{(i-1)} + (1 - \beta_2)(\nabla_{w_{(i)}})^2)$ $w_{(i+1)} = w_{(i)} - \frac{\eta \hat{m}_{(i)}}{\sqrt{\hat{v}_{(i)}} + h}$	m: first moment, v: second moment, $\beta_1 \approx 0.9$: first order decay, $\beta_2 \approx 0.999$: second order decay, $h \approx 0^+$: smoothing term, $\eta \approx 0.001$: learning rate	Fixes Adam generalization via max function in second moment	Inconsistent improvement over different data sets and models
AdamW	[114]	$\hat{m}_{(i)} = \frac{\beta_1 m_{(i-1)} + (1-\beta_1)\nabla_{w_{(i)}}}{1-\beta_1^i}$ $\hat{v}_{(i)} = \frac{\beta_2 v_{(i-1)} + (1-\beta_2)(\nabla_{w_{(i)}})^2}{1-\beta_2^i}$ $w_{(i+1)} = w_{(i)} - \eta\big(\frac{\alpha \hat{m}_{(i)}}{\sqrt{\hat{v}_{(i)} + h}} + \nabla_{w_{(i)}}\big)$	m: first moment, v: second moment, $\beta_1 \approx 0.9$: first order decay, $\beta_2 \approx 0.999$: second order decay, $h \approx 0^+$: smoothing term, $\eta \approx 0.01$: learning rate, $\alpha \approx 0.001$: ℓ_2 regularization weight	Improves ℓ_2 regularization of Adam via removing from second moment	Additional hyper-parameters
QHAdam	[115]	$\hat{m}_{(i)} = \frac{\beta_1 m_{(i-1)} + (1-\beta_1)\nabla_{w_{(i)}}}{1-\beta_1^i}$ $\hat{v}_{(i)} = \frac{\beta_2 v_{(i-1)} + (1-\beta_2)(\nabla_{w_{(i)}})^2}{1-\beta_2^i}$ $w(i+1) = w_{(i)} - \eta \left[\frac{(1-v_1) \cdot \nabla_{w_{(i)}} + v_1 \cdot \hat{m}_{(i)}}{\sqrt{(1-v_2)(\nabla_{w_{(i)}})^2 + v_2 \cdot \hat{v}_{(i)}} + h} \right]$	m: first moment, v: second moment, $\beta_1 \approx 0.9$: first order decay, $\beta_2 \approx 0.999$: second order decay, $h \approx 0^+$: smoothing term, $\eta \approx 0.001$: learning rate, $v_1 \approx 0.7$: first order immediate discount factor, $v_2 \approx 1$: second order immediate discount factor	Deeper convergence via weighted average updates from multiple algorithms	Additional hyper-parameters

2.2.2 Weight Initialization

If each layer of a L-layer deep NN model scales inputs by k, the final scaling will be k^L, such that for large L, we get $\nabla_{w_{(i)}} = 0,\ k < 1$ (the gradients "vanish"), or $\nabla_{w_{(i)}} = \infty,\ k > 1$ (the gradients "explode") [117]. This means that weights in deep learning, or any non-convex optimization problem, must be initialized to avoid prohibitively long training times. However, weights cannot be initialized at the same value. For example, a one input, four neuron NN would compute linear logits as $z = xw_0 + xw_1 + xw_2 + xw_3 + b_0 + b_1 + b_2 + b_3$. If all weights and biases are equal, then the gradient for each weight will be equal (*i.e.*, $\frac{\delta z}{\delta x} = w$), such that each neuron learns the same information (Figure 2.13).

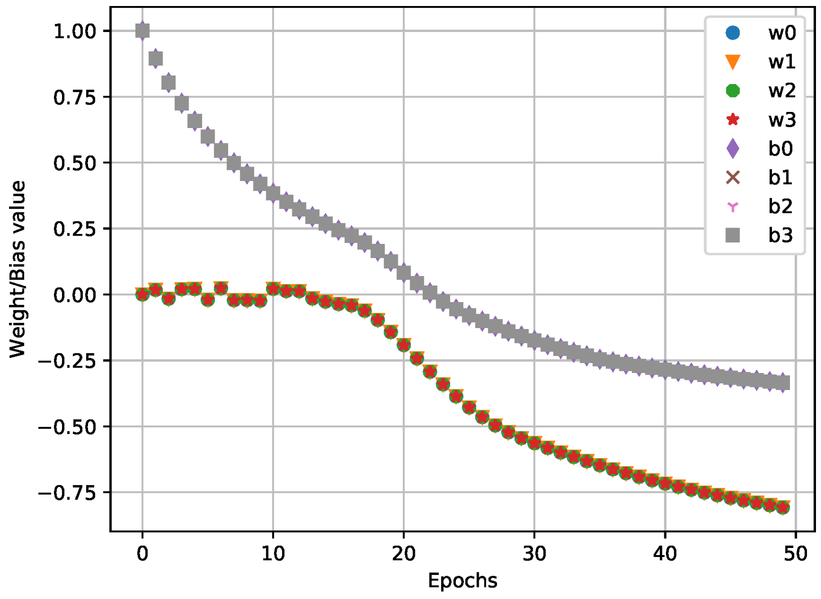


Figure 2.13: Bias and weight values of a four neuron NN over 50 epochs of SGD, initialized as $w = 0$ and $b = 1$. Consequently, all weights and biases compute the same gradient update for each epoch $\frac{\delta z}{\delta x} = w$, making weight convergence to values that correspond to the global loss minimum impossible.

In order to avoid exploding or vanishing gradients during training, weight initialization must follow two rules concerning the activations a or outputs of each layer of the NN:

$$E[a^l] = E[a^{l-1}], \forall l, \tag{2.41}$$

$$Var(a^l) = Var(a^{l-1}), \forall l. \tag{2.42}$$

Kumar [118] shows that any initialization scheme that follows these two rules in a NN with linear-near-zero activations such as tanh or sigmoid will express the equality:

$$Var(a^l) = n^{l-1}Var(w^l)Var(a^{l-1}), \forall l. \tag{2.43}$$

This equality serves as the justification for Xavier (also called Glorot) initialization [118], in which all weights of an n-neuron layer are initialized according to samples from the distribution:

$$w^l \sim \mathcal{N}(0, \frac{1}{n^{l-1}}), \tag{2.44}$$

$$b^l = 0, \tag{2.45}$$

where $Var(w^l) = \frac{1}{n^{l-1}}$ such that equation (2.43) satisfies constraint (2.42). The results of this can be seen in Figure 2.14, where each gradient is unique and each neuron learns different weight updates with no issues concerning vanishing or exploding gradients. Zero-mean normal or uniformly distributed initialization schemes without the proper variance scaling, as seen in Xavier intitialization, is sufficient for shallow models. However, they will experience vanishing or exploding gradients with deeper models because they fail to satisfy the constraint (2.42).

Many deep NN models do not use linear-near-zero activations, such as deep ReLU CNNs. The authors of Kaiming initialization [97] re-derived equation (2.43) with ReLU instead of tanh activations to determine that the simple modification:

$$w^l \sim \mathcal{N}(0, \sqrt{\frac{2}{n^{l-1}}}), \tag{2.46}$$

$$b^l = 0, \tag{2.47}$$

must be made to the initialization scheme such that the constraint (2.42) holds.

There also exists a truncated version of Kaiming initialization (and any Gaussian-based initialization), where the weight initializations are bounded by $-1 < w^l < 1$ to avoid sampling large weights, which becomes more common in larger NNs as $n \to \infty$ and $L \to \infty$ and the normal distrubtion is sampled from more times.

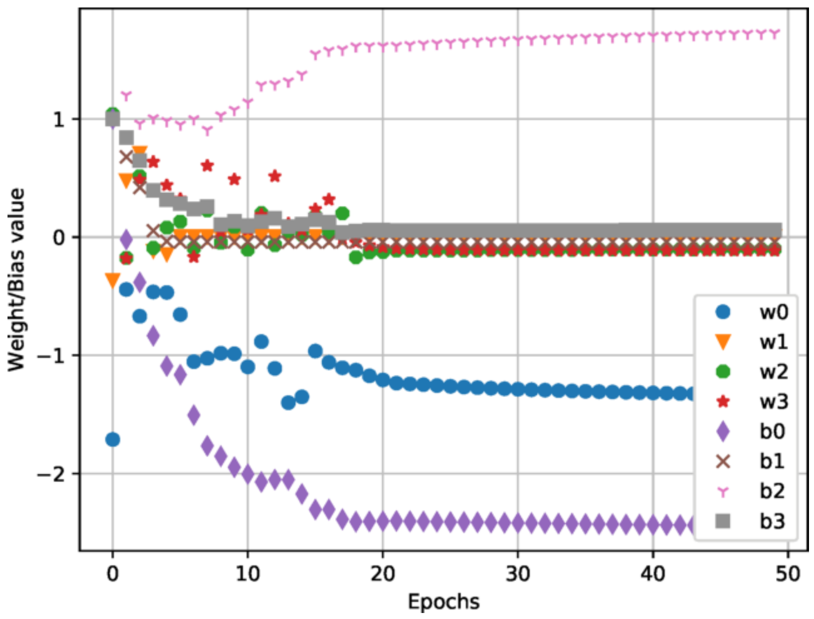


Figure 2.14: Bias and weight values for a single input, four neuron NN over 50 epochs of training, initialized via the Xavier initialization. Gradient computations are now able to take non-zero, unequal values and achieve convergence to values that correspond to a global loss minimum.

2.2.3 Model Regularization

Regularization in SL is to make an assumption and to constrain a model to that assumption. Examples include the assumption of a continuous range of optimal weights and the constraint of the tie-breaking term $\frac{1}{2nC}||w||^2$ in the non-kernelized dual SVM loss function, the assumption of input equivariance and constraint of filters in CNNs, and the assumption of varying input dimensions and constraint of hidden nodes in RNNs.

The given CNN and RNN regularization examples are specifically called regularization by weight sharing. With all else equal, a CNN or RNN model will have fewer weights than a logistic or softmax regression model due to its filters and hidden memory units. To summarize those sections, CNN filters compute logits via small convolutional windows instead of fully connecting every input from the previous layer. RNNs, on the other hand, maintain a small number of hidden memory states whose logits and activations are a superposition of all previous input values to that layer. These models with relatively fewer weights have decreased computational costs as they have smaller matrices to compute. They also have reduced risks of over-fitting, because fewer weights compute lower-order estimation and classifications of unknown parameters.

The given SVM regularization example is specifically called a regularization loss term. Loss terms can be included in the loss function (*i.e.*, categorical cross entropy or Mean Squared Error (MSE) loss) of any model by summation. Common loss terms are a function of the p-norm of the weights $||w||_p = (\sum w^p)^{1/p}$, and are implemented with the goal of forcing the model to learn certain weight values. Examples of loss terms include L1-norm regularization [119]:

$$\lambda||w||, \tag{2.48}$$

which forces weights to take on smaller values, and is weighted against the rest of the loss function by hyper-parameter λ. Interestingly, this behavior can equivalently be produced by adding Laplacian-distributed random samples to all training inputs, as outlined by Li *et al.* [120]. Another example is the L2-norm or Tikhonov regularization [119]:

$$\frac{\lambda}{2}||w||_2, \tag{2.49}$$

which also forces weights to be smaller but also forces there to be fewer outliers as $p \to \infty$. This behavior may alternatively be enforced by adding Gaussian-distributed random samples to all training inputs, as outlined by Li *et al.* [120]. By increasing the prediction error via loss terms, models trained with L1 and L2 regularization do not learn larger weight values unless their reduction of the prediction loss is larger than their increase of the regularization loss. This constraint mitigates arbitrary learning and creates models that generalize better to testing data.

Dropout [121] is another regularization technique, implemented when a model is assumed to be over-fitting because it has too many weights. Rather than reducing the number of weights in a model, constraining a model by using Dropout allows a subset of weights to be randomly selected and have their values set to zero for a single forward pass. In addition to mitigating over-fitting, Dropout has been found to also mitigate neuron co-adaptation as a tie-breaking constraint [121]. Neuron co-adaptation is the ambiguity of optimal weights caused when the linear logit of a neuron computed during training is equal to zero. This creates ambiguity because the inputs could either all be zero, or their weighted sum could be equal to zero. By randomly dropping weights, the weighted sum can be shifted to a non-zero value, which clarifies the ambiguity.

In SGD, it has been found that the weights converge the fastest when the distribution of weights at each layer in a NN possess zero mean with an identity covariance matrix [122]. Consider an NN with inputs x, hidden layer h, the input weight matrix g_1 and the output weight matrix g_2, and predictions $\hat{y} = g_2(g_1(x))$. During training, backpropagation dictates that $P(h)$ and $P(\hat{y})$ will be updated from the gradient $\frac{\delta f}{\delta g_1}$. It is at this point that "internal covariate shift" occurs: when $P(g_2)$ is updated according to $\frac{\delta f}{\delta g_2}$, $P(h)$ is no longer the same value as it was when $\frac{\delta f}{\delta g_2}$ was calculated. Batch normalization is a form of model regularization in which the i^{th} activation of layer l is controlled [123]:

$$a^*_{i,l} = \frac{a_{i,l} - \mu_l}{\sqrt{\sigma_l^2 + h}}, \tag{2.50}$$

where $h \approx 0^+$ to avoid divide by zero errors. By using batch normalization in our example, the distribution $P(h)$ would be held constant, such that all gradients are accurate, and internal covariate shift is reduced.

2.2.4 Data Representation

In wireless communications, the same analog signal can be represented by a significant number of axes or spaces [124], including amplitude-phase, IQ or complex space, Gram-Schmidt basis vectors, Fourier transforms, Laplace transforms, spectral correlation functions, cyclic autocorrelation functions, wavelet transforms, and spectrograms or "waterfalls". If sampled, the same signal can be represented as digital bits, code-words, symbols, Z-transforms, or any number of categorical attributes such as signal detected/signal not detected, signal transmitted by user 1/user 2, and duration since last transmission has been short/medium/long. This number of data representation choices becomes even more daunting when common arithmetic signal processing operations, such as auto and cross correlation, dot and cross products, energy and power calculations, spectral coherence, and phase, amplitude, or frequency differences, are applied in order to compare two or more of the signal representations mentioned thus far.

Luckily, the ideal data representation to choose for SL model inputs is usually given by existing unknown parameter estimation and classification methods. For instance, control coding schemes classify bits, and demodulators classify sequences of IQ data. There are two reasons behind these choices that should serve as a guide when no non-SL estimator or classifier is available: inputs should have low correlation and low multicollinearity. In other words, each input should be independent, and each input should have unique indicators that tie it to the ground truth value of its unknown parameter [124].

Input independence is important because models that train on many subsequent inputs of the same class generalize poorly. When many of the initial weight updates during SGD are computed with respect to highly dependent inputs, the model will consequently test successfully on that class, and unsuccessfully on all others. This input dependence is most commonly mitigated by shuffling the training data set and its corresponding labels. The correlation between two continuous signals $f(t)$ and $g(t)$ can be quantified as [125]:

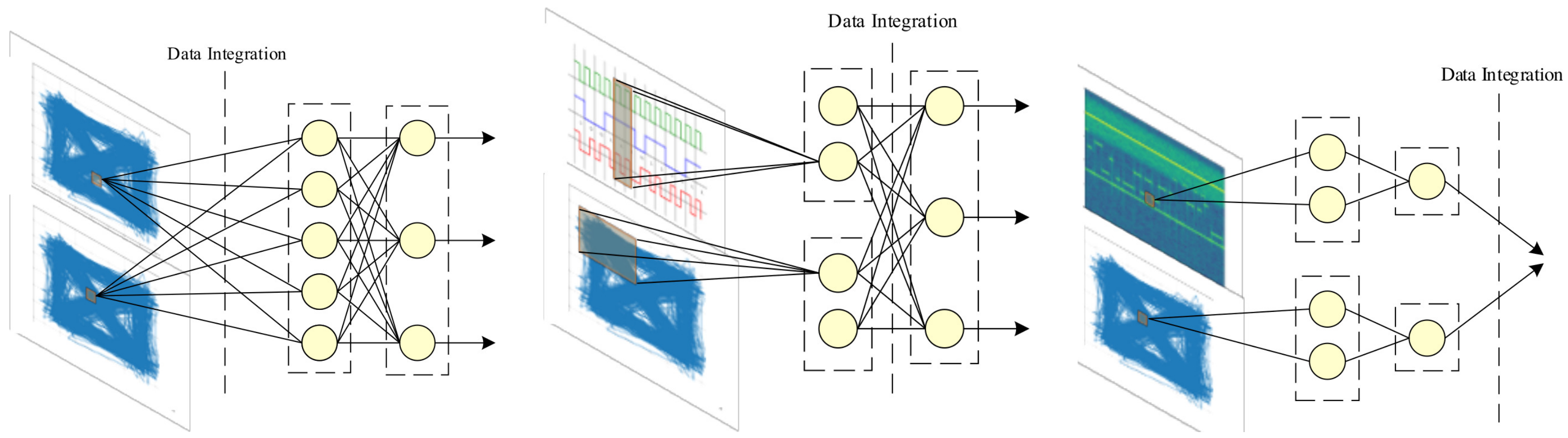
$$(f * g)(\tau) = \int_{-\infty}^{\infty} f^*(t) g(t+\tau) \delta t. \tag{2.51}$$

Low multicollinearity is important because highly similar inputs with different unknown parameters cannot be correctly estimated or classified. Multiple data representations may be incorporated into a SL model using early, middle, or late integration (Figure 2.15) in an effort to decrease multicollinearity. Early integration involves combining the data before the SL model, either via the concatenation of the data, or the addition of another dimension to the data space. Middle integration can only be performed with NN models, where each data representation has an independent branch of neuron layers that are concatenated partway into the NNs architecture. Late integration involves training independent SL models for each data representation, and interprets the set of unknown parameter estimates or classifications as a single result via averaging or voting functions. A popular metric used to quantify multicollinearity is the variance inflation factor, which describes how much the variance of a trained weight changes when predictions are correlated. It is computed as [126]:

$$VIF = \frac{1}{1 - R^2}, \tag{2.52}$$

where R^2 is the coefficient of determination, computed as:

$$R^2 = 1 - \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_i (y_i - f(x_i))^2}. \tag{2.53}$$



(a) Early Integration (b) Middle Integration (c) Late Integration

Figure 2.15: Different data representations can be integrated, via multiple approaches, in order to achieve an additional prediction performance gain by reducing multicollinearity. Data from the same domain is typically concatenated and integrated early as shown in (a). Data domains with high correlation between each other are typically integrated in the middle of a model as shown in (b), and highly dissimilar data is integrated at the end of an implementation by some voting scheme (c). Alternatively, late integration of identical inputs can be used as a form of ensemble learning in order to reduce the variance of model predictions introduced by the randomness of SGD (see Section 2.2.8).

2.2.5 Data Pre-Processing

While many SL models achieve universal approximator characteristics, that does not mean that data can be recklessly thrown into training without any considerations for null values, standardization, categorical variables, one-hot encoding, sample dependence, or multicollinearity.

Null values can appear in signal data sets for any number of reasons, such as poor alignment of signals or varying length samples. These null values will carry through to model predictions and gradients during training if not changed. Imputation, or the process of substituting null values, is one solution, where the null value's true value can be estimated via the mean of that feature across all signals in the data set, or via the mean of that signal across all features in that signal. If the researcher wishes to keep the null value as an indicator that something that caused a null value has occurred, imputation can instead take the form of substituting a categorical value that is outside the range of values taken by all other features (*i.e.*, a value of -1 in an otherwise all positive valued data set). Finally, if all null values occur at the start or end of signals in the data set due to signals of varying length, the signals' null values can be clipped or replaced by zeros in order to make all signals in the data set the same length. This method is typically a last resort, however, as clipping removes information, and adding zeros adds fabricated information.

Standardization of data is to force a zero mean and unit variance distribution on inputs. Symmetric distributions allow for faster convergence during SGD when using momentum, or schemes such as those discussed in Section 2.2.1. Non-symmetric gradient "landscapes" cause momentum updates to "circle the drain", overshoot loss minima by building up speed on steep declines, and contain "rougher" landscapes with more local loss minima. Standarization is implemented as:

$$x_i = \frac{x_i - \mu}{\sigma^2}, \tag{2.54}$$

where a data sample $x_i, i = 1, ..., N$ is zero centered by subtracting the data set mean μ and given unit variance by dividing by the data set variance σ^2.

During classification tasks, the unknown parameter of each input takes the form of a discrete value from a set of possibilities. Inputs can also be discrete values. Both discrete or categorical inputs and outputs of SL models are either ordinal or nominal. Ordinal values are related. If a pulse amplitude modulated signal encodes signals as having amplitudes $A \in -3A, -A, A, 3A$, then signals of $3A$ are more similar to A signals than $-A$ or $-3A$

signals. Nominal categorical values have no relationship: no two values are more or less similar to each other than any other pair of values. While ordinal values can be expressed using decimal values $1, 2, 3...$, nominal values cannot, or a false relationship between values will be taught to the SL model. Nominal values must instead be one hot encoded, such that if m possible categories exist and the value of that category is n, then the value is expressed as a vector of m zero-valued elements with the n^{th} element set to a value of one.

Multicollinearity can be mitigated by pre-processing. Principal Component Analysis (PCA) is an algorithm [127] that removes input features from all signals of a feature-rich data set. The goal is to remove features with high multicollinearity, such that the low multicollinearity features can dominantly teach the SL model. In PCA, user-defined p low multicollinearity input features are identified by standardizing input data, computing the covariance matrix of the standardized input data, and computing the eigenvalue decomposition of the standardized covariance matrix. Principle components describe the dimension-reduced form of the data set, where each feature is a linear combination of several other features. The singular value decomposition factorization of the standardized covariance matrix $\mathrm{cov}(X_{ZC})$ gives the principal components X_{PCA} as [128]:

$$x_{i_{ZC}} = x_i - \frac{\sum x_i}{k \times D}, x_i \in X, x_{i_{ZC}} \in X_{ZC}, \tag{2.55a}$$

$$\mathrm{cov}(X_{ZC}) = \frac{X_{ZC}^T \cdot X_{ZC}}{N}, \tag{2.55b}$$

$$USV^* = \mathrm{cov}(X_{ZC}), \tag{2.55c}$$

$$X_{PCA} = X_{ZC} \cdot U_p, U_p \in [N, p], \tag{2.55d}$$

where eigenvalues are S, conjugate transpose of the unitary matrix V^*, k samples per signal, number of features $D \geq p$, number of samples N, matrix transpose X^T, and the element-wise dot product between two matrices $\cdot$

$$X \cdot Y = x_1 \times y_1 + x_2 \times y_2 + ... + x_{n-1} \times y_{n-1}, \tag{2.56}$$

for elements $x \in X, y \in Y$ of length n.

2.2.6 Data Augmentation

Small training data sets do not train SL models well because they do not fully describe the testing data's underlying probability distribution. If more data cannot be collected, data augmentation can instead increase the diversity of data in a small training set by estimating the underlying distribution and sampling additional signals to combat data scarcity.

A common way of estimating the underlying distribution is to train an encoding NN that maps inputs to a lower dimension $z(x)$, and a decoding NN that maps inputs back to their original values $h(z(x)) = x$. This is done by forcing the encoder weights via regularization to achieve a Gaussian behavior, $z(x) \sim \mathcal{N}(0, 1)$. If the combined encoder-decoder weights are well-trained to reconstruct images, and the encoder weights are effectively regularized, random Gaussian samples can be input to the decoder $h(\mathcal{N}(0, 1))$ to output simulated training data. This process outlines how Generative Adversarial Networks (GANs) [129] and VAEs [130] are used for data augmentation. Less popular and more constrainted distribution estimators are produced through Bayesian Metropolis-Hastings sampling [131], Gibbs sampling [132], importance sampling [133], and rejection sampling [134].

Training data can also be simulated by applying plausible transforms to existing data sets. This form of data augmentation is performed by introducing simulated variations to the training data under the assumption that the test data will also be affected by those variations. Examples of plausible transforms include frequency, amplitude, phase shifts in the data, wireless channel effects such as AWGN, multi-path, scattering, diffraction, reflection, and Doppler, and hardware non-idealities such as clock drift, power amplification characteristics, and other RF front end non-linearities.

2.2.7 Hyper-Parameter Validation

When designing a SL model, there are often several aspects of the architecture or SGD (*i.e.*, learning rate, number of neurons in a layer) that provide no clear choice towards maximizing the testing accuracy. The process of experimenting with different operating parameters and other implementation values is referred to as hyper-parameter validation [92].

One of the most straightforward methods of hyper-parameter validation is a grid search, through which a SL model is trained with the same training data that uses each combination of hyper-parameters h from the set H. In order to evaluate which hyper-parameter subset is optimal, the training data set must be partitioned into a smaller training data set and a validation set. The hyper-parameters that result in a trained model with the highest validation set accuracy are used at the testing stage of SL model deployment.

The gap between the validation and the test prediction performance is minimized when the number of samples in both partitions are large and equal in count, and when both sets are sampled from the same underlying distribution. Common partition ratios between training, validation, and testing sets include 80:10:10, 60:20:20, and 50:25:25. Ratios with smaller training sets are chosen when the data set size is large, which affects the minimization of the variance in test data performance. Smaller validation/testing ratios are chosen when the data set size is small, in order to ensure the proper training of the model. As the number of hyper-parameters used in grid search increases, computational costs become prohibitive (Algorithm 7).

Algorithm 5 Grid-search validation method [135]

```
1: procedure GIVEN TRAINING DATA $X_{tr}$, VALIDATION DATA $X_{val}$, SET OF HYPER-PARAMETERS $H$
2:     Best weights $w^* = None$
3:     Best accuracy $acc^* = 0$
4:     for Set $h$ in $H$ do
5:         Train model on $X_{tr}, Y_{tr}$ to obtain $w$ using $h$
6:         Test model on $X_{val}$ using $w$ to obtain $\hat{Y}_{val}$
7:         Compute test accuracy $acc$ using $\hat{Y}_{val}$
8:         if $acc > acc^*$ then
9:             $acc^* = acc$
10:            $w^* = w$
11:        end if
12:    end for
13: end procedure
```

If the amount of data is limited, validation accuracy can vary significantly, and sub-optimal hyper-parameters can be selected. Cross-fold hyper parameter validation, where the training data is split up into k equal-sized "folds", or partitions (instead of just two, as in grid search), can mitigate this issue. For each of the k folds, one fold is chosen as the validation set, and the other $k - 1$ folds are used for training. The data split this way are grid searched. Then, a new fold is chosen as the single validation fold, and the old validation fold becomes part of the new $k - 1$ training partition. This process repeats until each fold has served as the validation fold, then the final hyper-parameters are chosen based on which hyper-parameters had the highest average validation accuracy across each of the k folds. The weights trained with these hyper-parameters are used for final test data evaluation (Algorithm 6). The larger k is, the higher the correlation between each learned model; the highest correlation exists for $k = N$, or leave-one-out validation. The smaller k is, the higher the variance of predictions, as the training set gets smaller. Common compromises include $k = 3, 5, 10$. Cross-fold validation makes up for the high variance that would be experienced in grid search of small training sets by averaging over all folds in order to obtain hyper-parameter choices that are not biased to any one chosen fold.

Algorithm 6 Cross-fold validation method [136]

```
1: procedure GIVEN TRAINING DATA $X$, SET OF HYPER-PARAMETER SETS $H$, $K$ FOLDS
2:     Partition $X$ into $K$ folds $X_k, k = 1, ..., K$
3:     Best weights $w^* = None$
4:     Best accuracy $acc^* = 0$
5:     for Set $h$ in $H$ do
6:         for $k = 1, ..., K$ do
7:             Train model on $X \not\subset X_k$ to obtain $w$ using $h$
8:             Test model on $X_k$ using $w$ to obtain $\hat{Y}_k$
9:             Compute test accuracy $acc_k$ using $\hat{Y}_k$
10:        end for
11:        Compute average $acc = \frac{1}{K} \sum_k acc_k$
12:        if $acc > acc^*$ then
13:            $acc^* = acc$
14:            $w^* = w$
15:        end if
16:    end for
17: end procedure
```

This process can be repeated using nested cross-fold validation, where multiple SL models (*i.e.*, SVM, RNN, DT) are tuned to complete the same unknown parameter estimation or classification task.

2.2.8 Ensemble Learning

A form of late integration (Section 2.2.4) known as ensemble learning can significantly improve the performance of trained SL models. An ensemble of SL models is a collection of n models independently trained on the same data set. Ensembles perform classification or regression by averaging their predictions on test data. The benefit of this can be shown by minimizng the expected squared error of predictions, defined as:

$$E\left[((\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i) - y)^2\right] = \frac{v}{n} + c\frac{n-1}{n}, \tag{2.57}$$

where the variance $v = E[(\hat{y}_i - y_i)^2]$, and the covariance $c = E[(\hat{y}_i - y_j)^2], i \neq j$. If the covariance is zero, or each model makes different mistakes, then the gain of using an ensemble on the squared error of predictions is $\frac{1}{n}$. If $v = c$, then the ensemble brings no gains, and the prediction MSE remains at v.

While any SL model can be trained in an ensemble, there exist a few ensemble training algorithms specific to a certain family of SL algorithms or that are designed to mitigate training issues. Bootstrapping is a technique used to reduce prediction covariance c by training each model in the ensemble with a different set of examples from the training set, sampled with replacement. The random forest [137] algorithm is an example of bagging, which trains n DT on bootstrapped data. Furthermore, random forest ensemble splits the attributes of each node of each version of the DT randomly, selecting m attributes from p each time. Attribute selection is usually done by computing the entropy of each attribute and selecting the highest entropy feature. Finally, ensembles that perform classification by bagging pick the class with the highest number of votes.

Adaboost [138] is another ensemble algorithm, where SVM or DT models are boosted rather than bagged. The difference between the two is that the classifications of boosted ensembles are weighted, and bagged ensembles have equal weight voting. The weights are determined using error computations, such that weak learners, or models with high prediction variance v, have less influence on classifications.

2.3 UL and Mixture Models

Gaussian Mixture Models (GMMs) can serve as a probabilistic model to assign the locations estimates $\hat{P}_i$ to K sub-populations, where each sub-population represents a wireless beacon located at $\hat{\mu_k}$ with location estimates that vary according to the covariance matrix $\hat{\Sigma_k}$. In application, multi-object tracking models cannot assume the number of objects k. Non-parametric mixture models side step this issue by assuming an infinite number of sup-populations. The GMM generative model produces observations:

$$p(y_i|\pi_1, ..., \pi_k, \mu_1, ..., \mu_k, s_1, ..., s_k) = \sum_{j=1}^{k} \pi_j \mathcal{N}(\mu_j, s_j^{-1}), \tag{2.58}$$

where the mixing proportions $\sum_{j=1}^{k} \pi_j = 1$ and observations from Gaussians with larger mixing proportional are generated more often. Since computing $p(\pi_1, ..., \pi_k|y_1, ..., y_N) \propto p(y_1, ..., y_N|\pi_1, ..., \pi_N)p(\pi_1, ..., \pi_N)0$ is nontractable, inference of GMMs is be performed by markov chain monte carlo sampling [139], Variational Inference (VI) [140], or Expectation Maximization (EM) [141]. If data samples are large with high dimensions and many non-Gaussian latent variables, variational inference be used. For strictly GMM observations, the authors prefer EM as a more hands-off approach, which risks convergence to a local minima, over VI, which risks mode default and poor mixing [142].

EM optimization of DPGMM inference of parameters $\theta = \pi_1, ..., \pi_k, \mu_1, ..., \mu_k, s_1, ..., s_k$ is then given (Algorithm 7) by the expectation step [141]:

$$p(c_i = k|y_i, \theta) = \frac{\mathcal{N}(\mu_k, s_k^{-1})\pi_k}{\sum_{j=1}^{k} \mathcal{N}(\mu_j, s_j^{-1})\pi_j}, \tag{2.59}$$

where observation y_i belongs to mixture k if $p(c_i = k|y_i, \theta) > p(c_i \neq k|y_i, \theta)$.

The maximization step is given as the computation of the new parameters θ as:

$$\pi_k = \frac{1}{N} \sum_{i=1}^{N} p(c_i = k|y_i, \theta), \tag{2.60}$$

$$\mu_k = \frac{\sum_{i=1}^{N} p(c_i = k|y_i, \theta) y_i}{\sum_{i=1}^{N} p(c_i = k|y_i, \theta)}, \tag{2.61}$$

$$s_k = \frac{\sum_{i=1}^{N} p(c_i = k|y_i, \theta)}{\sum_{i=1}^{N} (y_i - \mu_k)^2 p(c_i = k|y_i, \theta)}. \tag{2.62}$$

2.4 Adversarial Perturbations

An adversarial perturbation is defined as a jamming signal that is added to a signal that is given to a ML model during either training or testing with the intent of causing incorrect estimation or classification during the testing phase of the ML model. This interaction can be generally described as [5]:

$$x^* = x + \epsilon\eta, \tag{2.63}$$

Algorithm 7 Dirichlet Process Gaussian Mixture Model inference by EM [141]

```
1: procedure GIVEN OBSERVATIONS $y_i$
2:     Initialize $\theta = \pi_1, ..., \pi_k, \mu_1, ..., \mu_k, s_1, ..., s_k$
3:     while not converged do
4:         for $i \in \{1, ..., N\}$ do
5:             compute $p(c_i = k|y_i, \theta)$
6:         end for
7:         for $j \in \{1, ..., k\}$ do
8:             compute $\pi_j|p(c_i = k|y_i, \theta)$
9:             compute $\mu_j|y_i, p(c_i = k|y_i, \theta)$
10:            compute $s_j|y_i, \mu_k, p(c_i = k|y_i, \theta)$
11:        end for
12:    end while
13: end procedure
```

where the perturbation η is scaled by ϵ and added to the original signal x to form a perturbed sum of signal x^*. If the trained ML model's predictions are described as $f(x) = \hat{y}$, then the perturbations must be crafted such that the prediction loss increases, or, formally:

$$\mathcal{L}(f(x^*), y) > \mathcal{L}(f(x), y). \tag{2.64}$$

The reason for the scalar or mask ϵ was originally to craft computer vision perturbations that are imperceptible to the human eye, but broadly is used to minimize the perturbation according to the metric and scheme of choice. Broadly speaking, ϵ schemes can be either constrained to a fixed value or optimized to achieve a minimum or maximum result for a specified constraint function. Typically ϵ minimizing is defined by the vector norm metric for some chosen p, most commonly $p \in \{0, 1, 2, \infty\}$:

$$||\epsilon||_p = \left(\sum_{i=1}^{n} ||\epsilon_i||^p \right)^{1/p} = \delta \tag{2.65}$$

Finally, ϵ can take the form of a scalar that effects all elements of the perturbation signal equally, or it can take the form of a mask or signal that scales some elements more or less than others.

The perturbation itself can be described by a number of attributes, including adversarial falsification, adversary knowledge, adversary specificity, and attack frequency.

- Adversarial falsification: The perturbation is crafted to achieve false positive or false negative predictions.

- Adversary knowledge: The perturbation is crafted with full knowledge of the trained ML model (white box) or only knowing a history of queries, or observed inputs and outputs of the ML model (black box).

- Attack frequency: Perturbations may be crafted after one query (black) or gradient computation (white), or improved over a number of repeated queries or gradient computations.

- Adversary specificity: The perturbation is crafted to achieve a targeted false prediction or any false, non-targeted prediction.

For surveys of adversarial attacks and countermeasures, the reader is directed towards other works [6, 60–62]. Since gradient-based white box attacks leverage their knowledge to take small steps in directions of steep or unstable loss, the theory behind most countermeasures is to increase the "flatness" of trained models loss function. This serves the dual purpose of generalizing the model well to unexpected statistical behavior in the testing data partition. Black box countermeasures, on the other hand, leverage the transferability attribute, which holds as long as non-linear activation functions produce outputs that are distributed primarily around their linear region of operation. Consequently, countermeasures for these attacks carefully reduce the linear behavior of classifiers, which makes learning more difficult.

Adversarial perturbations in the wireless communications domain are ubiquitously generated via the RML2016.10a simulated dataset [143]. The data set was the first of two [144] modulation classification data sets, classifying 128-sample IQ signals to one of 11 modulation classes. The data experiences a noisy channel simulated by GNU Radio's dynamic channel model, which applies a truncated Gaussian-distributed random walk to symbol rate and carrier frequency offset, a Rician-distributed frequency-selective fading multipath model, and a Gaussian-distributed AWGN model. The adversarial papers also use the same supervised learning model to compute perturbations, the VT-CNN2 [2] model. This has made the comparison of various papers a simple task, while amplifying the issues present in the data set and model due to a lack of diversity.

These perturbation papers all assume one of two different three-player scenarios: a friendly transmitter and receiver, where an intelligent, jamming adversary synchronously adds perturbations to the transmitter to fool the receiving radio, which classifies the observed sum of signals [76, 77]. In the other, a friendly transmitter adds pre-channel perturbations to fool an adversarial receiver which classifies the transmissions, while maximizing communication capabilities between the transmitter and receiver [78–80].

2.5 Chapter Summary

In this chapter we have provided the reader with a statistical foundation in neural networks, SL, and UL and surveyed the implementation challenges that can arise when employing them. Through examples, visualization, and derivation, we have shown the power of these algorithms while also drawing attention to the systematic vulnerabilities their implementation can present to attackers. We now present the contribution chapters of this dissertation, wherein we contribute to the state-of-the-art by securing physical layer passband signals using ML.

Chapter 3

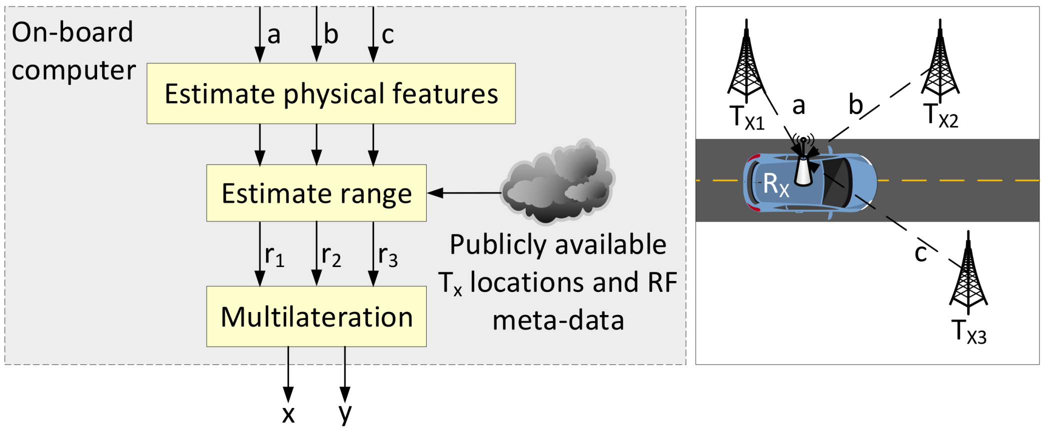
Machine Learning-Based Parameterized Fingerprinting for Unknown Number of Transmitters

3.1 Introduction

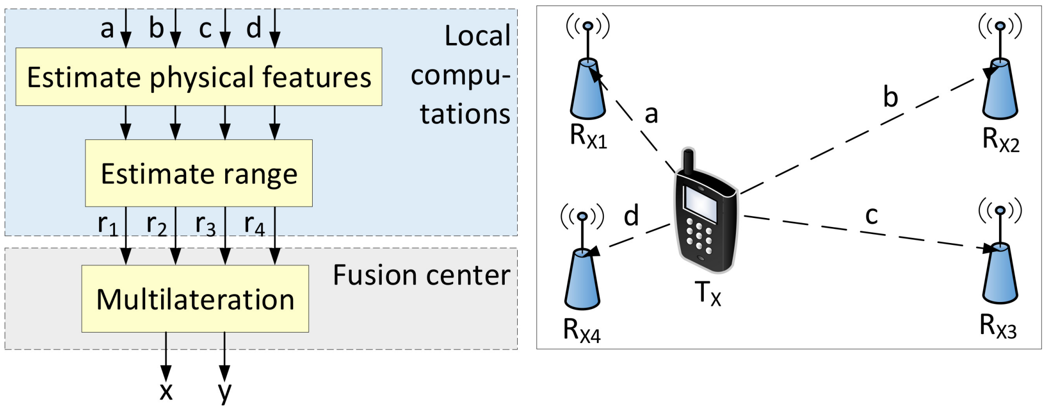
In this chapter, we present a novel algorithmic framework distinguished from other state-of-the-art for low assumption multilateration networks in the following ways:

1. The framework provides a stationary alternative to MHT that is relatively computationally inexpensive by utilizing Bayesian clustering instead of hypothesis test trees.
2. It introduces an opportunistic alternative to current WSNs that use multilateration data to classify signal source IDs, allowing for localization without packet sniffing.
3. The framework requires fewer assumptions compared to current RF signature-based WSNs that use phase and frequency meta-data, allowing for the localization of transmitters when little is known *a priori* by the WSN about RF behavior.

The rest of this chapter is organized as follows. Section 3.2 presents a detailed overview of the SOP concept, the value of meta-data, and the goals of our approach. Section 3.3 describes the framework as well as details of our methodology and experimentation. Section 3.4 evaluates the performance of our CNN-based multilateration and UL-based clustering of spatial multilateration data. This analysis includes an Anderson Darling (AD)



(a) Outdoor receiver localization



(b) Indoor transmitter localization

Figure 3.1: A typical SOP navigation scenario (a) and its inverse problem, SOP beacon localization (b). Physical feature and range estimates are independent of other readings in the SOP beacon localization problem, such that they may be computed locally at each receiving sensor or at the fusion center.

normality test of spatial data to confirm the Gaussian assumptions of the UL algorithm, as well as a Kruskal-Wallis (KW) non-parametric test of Received Signal Strength (RSS) estimates. Final thoughts are discussed in Section 3.5, including drawbacks and open challenges of our methodology.

3.2 Overview of Localization Model

A SOP is a wireless signal that is both freely available but not designed to carry information for receiving localization systems [150]. SOP-based navigation systems perform

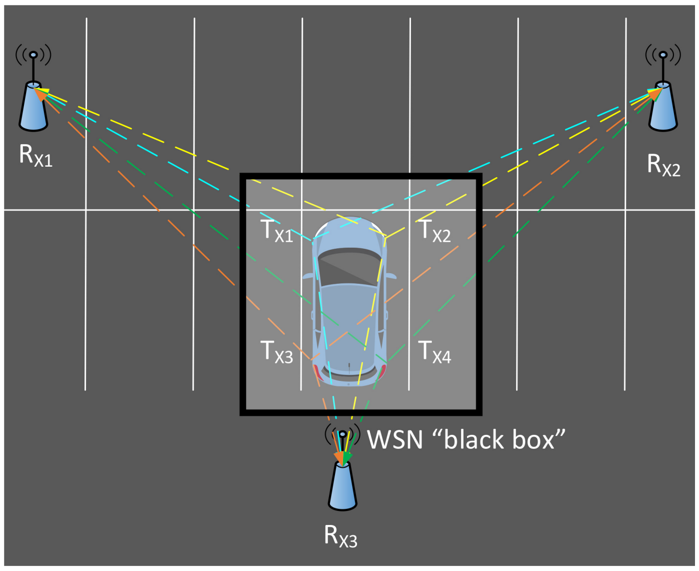


Figure 3.2: An example of the low meta data scenario is a subset of the transmitter localization problem, where the number of beacons is hidden from the localization network of sensors, and must be inferred. As in the standard localization problem, the the transmitter locations are unknown. Additionally, the origin of the four signals detected in this scenario is hidden, such that the receiving network must infer which transmitter sent which signal via the physical layer pass-band characteristic inference of signal strength, angle, or timing estimates.

multilateration given a user's receiver with an unknown location. Several local transmitters with known locations are used with enough wireless information known about their carrier frequency and bandwidth to individually detect each signal and estimate physical properties from them. These systems (Figure 3.1a) typically use high altitude, high transmit power SOPs such as Digital Television (DTV), Global System for Mobile Communications (GSM), and Code Division Multiple Access (CDMA) signals in outdoor multilateration experiments with a small number of SOP transmitters [151–153]. System performance for these approaches varies greatly depending on the availability and orientation of the transmitters, and their advantages and disadvantages have been sufficiently covered in [149,150].

SOP-based navigation researchers have also explored the inverse approach focusing on the beacon localization problem. In this approach, several receivers with known locations are used together to estimate the positions of one or more SOP-emitting transmitters with unknown locations (Figure 3.1b). They usually report Supervised Learning (SL)-based, RSS-based indoor multilateration [154] or parameterized fingerprinting experiments to estimate the location of a single Zigbee, Wi-Fi, Bluetooth, or Ultra Wide-Band (UWB) SOP transmitter via multiple receivers [155–160]. System performance for these works varies greatly depending on the SL algorithm used, training data collected, and the number, placement, and design of the receivers.

In this work, we investigated a variant of the transmitter localization scenario, where a WSN "black box" had to solve the problem without meta data or packet sniffing capabilities (Figure 3.2). Metaphorically speaking, the "black box" represents the WSN operator's intent to localize any detectable signal without knowing their protocol type (*e.g.* Zigbee, Wi-Fi, Bluetooth, UWB, TPMS). Without *a priori* knowledge of how the channel activity corresponds to transmitter activity or packet-layer data such as reported transmitter location [53–58], our WSN was forced to use only locally-computed physical signal characteristics (*e.g.*, Time Difference of Arrival (TDoA), RSS) to determine the source of each detected signals while also locating and counting those source radios. Given that pre-multilateration physical features in this complex WSN varied by protocol, time, frequency, bandwidth, and several other physical characteristics, a post-multilateration clustering approach was needed to leverage the sum-of-Gaussians characteristic of all CNN multilateration estimates despite their front-end variations.

3.3 Proposed Multilateration Framework

In this section, we describe our framework (see Figure 3.3). The method begins with the signal detection and pre-processing of the TPMS signals received by our WSN setup, their processing by a central computer to produce several input features, generation of spatial transmitter location estimates from those features via CNN-based multilateration, and UL-based inference of unknown parameters leveraging that spatial data.

3.3.1 Signal Detection and Pre-Processing

Data files in the form of two-dimensional arrays were processed to generate RSS estimates from each receiver for each TPMS frame over time before localization could occur (block (a) in Figure 3.3). The process of extracting RSS estimates from a given data file includes: down-sampling, frame detection, preamble frequency and timing correction, and frame correlation for estimating the RSS per frame.

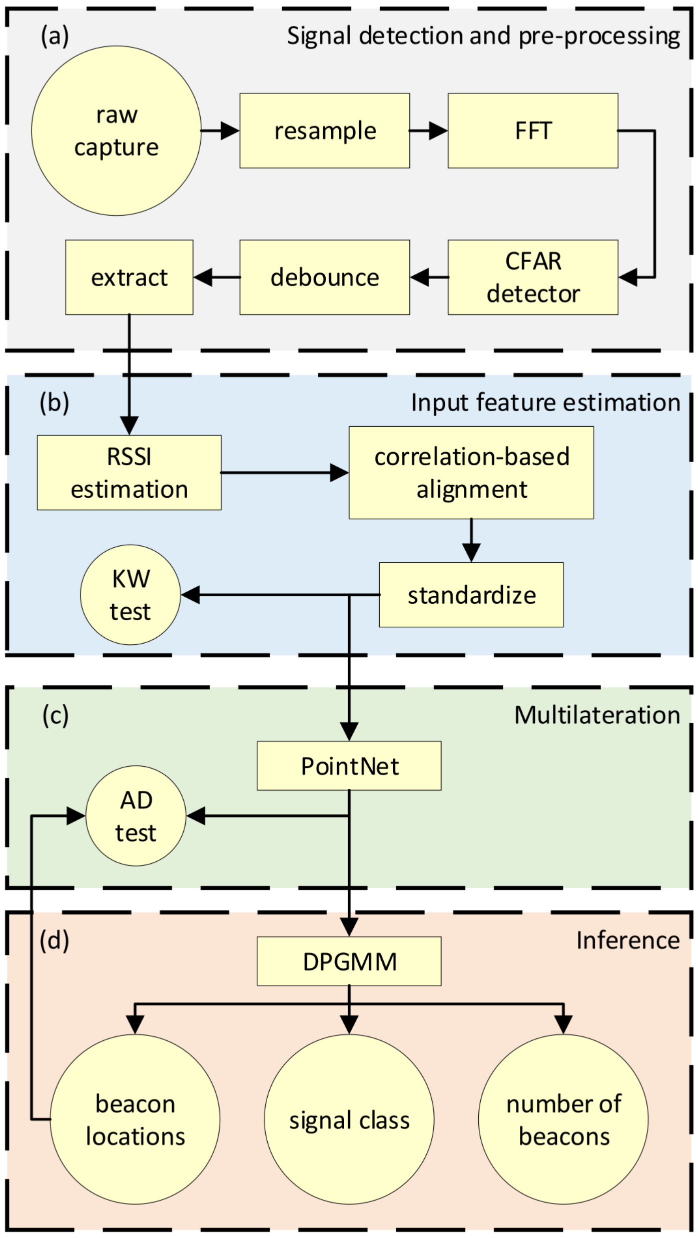


Figure 3.3: An overview of the interaction between of all estimators, classifiers, and digital signal processing tasks of the proposed approach presented in this work. Inputs and outputs are represented by circles, while intermediate steps are represented by rectangles. kW and AD tests were performed to determine the uniqueness of each beacon's true RSS population and normality of estimated beacon location subtracted multilateration estimates, respectively. Normality is an assumption of the DPGMM inference, and by determining the number of unique input data populations, we identify maximum performance bounds for the DPGMM inference.

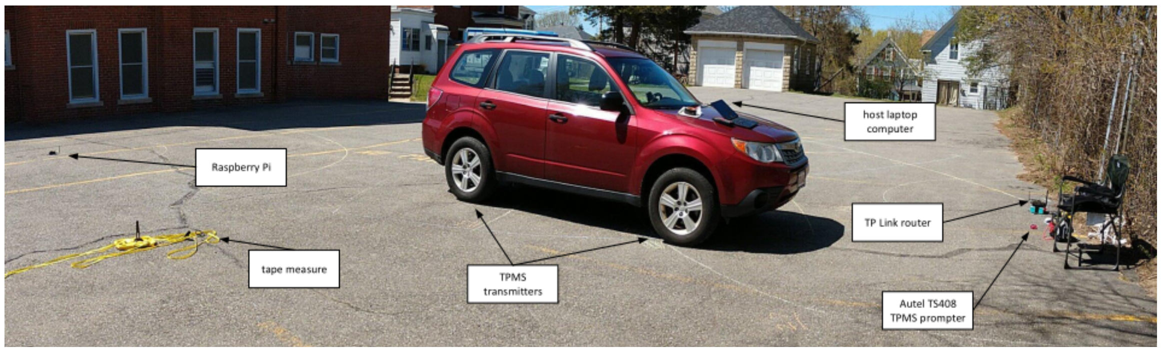


Figure 3.4: The experiment site, located at 358 Pleasant St, Gardner MA, USA (longitude -71^{o}59'41.32"W, latitude -42^{o}34'9.19"N). The hardware used in the experiment is also pictured.

The first step required band-pass filtering and down-sampling to reduce the computational load as well as increase the effective bit depth resulting from oversampling.

After re-sampling, detection was performed. This step began by identifying the start of each individual frame. For this application we used a simple power detector with a Constant False Alarm Rate (CFAR) threshold. We started by decomposing the down-sampled file into $k = 1, ..., K$ bins based on the detector's integration period T. The energy level of each bin in the decomposed signal $x(t)$ was computed as [161]:

$$E[k] = \frac{1}{T}\int_{Tk}^{T(k-1)} \|x(t)\|^2 dt. \tag{3.1}$$

For this type of CFAR detector, the threshold level is computed using [162]:

$$\alpha = N(P_{\text{fa}}^{-1/N} - 1), \tag{3.2}$$

given false alarm probability P_{fa} and number of samples N.

3.3.2 Input Feature Estimation

After a frame was detected, the frequency and timing errors had to be corrected before the RSS can be estimated using correlation (see block (b) in Figure 3.3). The frequency and timing offsets form a continuous ambiguity function (CAF). To estimate the frequency and timing offset parameters, the known preamble was frequency shifted and stretched as needed, then correlated against the detected frame. The peak of the correlations represented the best fit. This technique allowed for varying degrees of accuracy but at the expense of computational load. For this experiment, we post-processed the data on a desktop such that computational resources were not a limiting factor. The correlation between the CAF search preamble and the received signal also acted to estimate the RSS of the received frame [163]:

$$\text{RSS} = \frac{1}{N}\sum_{t=1}^{N} x_t ||\hat{x}_t||, \tag{3.3}$$

where x_t sampled from $t = 1, ..., N$ is the CFAR detected signal and $\hat{x}_t$ is the unit power CAF preamble after correcting for frequency and timing offsets. Through this process, we employed three receivers to produce a dataset where input features were represented as $x = [\text{RSS}_1, \text{RSS}_2, \text{RSS}_3]$.

3.3.3 Multilateration

Multilateration (see block (c) of Figure 3.3) is the process of using physical signal characteristics to estimate the distance between receiver-transmitter pairs, and subsequently computing the transmitter's most likely location. The CNN algorithm is a popular choice for replacing both stages of multilateration with a set of trainable weights [155–160]. This algorithmic substitution is useful for replacing fingerprinting methods because it allows for continuous, rather than discrete-space, unknown parameter estimation through the use of trainable weights. This attribute is important since physical space is a continuous variable, and traditional fingerprinting typically predicts physical transmitter location via nearest-neighbor techniques that introduce inherent errors due to their discrete-space nature [155–160]. Alternatively, traditional continuous-space, physical characteristic-based transmitter localization techniques first estimate the range by assuming an electromagnetic signal propagation path loss model, then solve for the geometric problem of intersecting circles (or parabolas) to identify the most likely region of the transmitter [164]. State-of-the-art CNN localization works [155–160] use the universal approximator characteristic [165] of Neural Networks (NNs) to provide flexible, accurate transmitter location estimates that improve traditional fingerprinting and provide an alternative to multilateration if the training data is available. Although range and geometry solving multilateration methods do not require training data, they do require meta data in the form of path loss information [166].

Although CNN architectures can take many forms, the Stochastic Gradient Descent

(SGD) training methodology is frequently employed (see Algorithm 8) [167]. A NN can be described as a union of small non-linear regression models arranged in both parallel and serial connections. In our case, these models were iteratively optimized via SGD to minimize the estimation error of each transmitter's location. The use of filters in the CNN architecture can be described as enforcing a constraint on weights that results in model regularization, lower computational costs, and the model assumption of input equivariance. Input equivariance is the idea that data patterns correlated to the ground truth are highly similar despite shifts, rotations, and other common transforms.

Algorithm 8 CNN model training protocol [167]

```
 1: procedure GIVEN TRAINING DATA $X$, LABELS $Y$, LEARNING RATE $\eta$, TRAINING ITERATIONS $n_e$
 2:     initialize model parameters $w, b$
 3:     for $n_e$ do
 4:         for each $x$ in $X$, $y$ in $Y$ do
 5:             for layer convolution layer $l$ in $L$ do
 6:                 for filter $w$ in layer $l$ do
 7:                     apply zero-padding if used
 8:                     compute convolution $z_l(x, w, b)$
 9:                     apply ReLU $a = \max(0, z)$
10:                     compute max pooling $P = \text{pool}(a)$
11:                 end for
12:             end for
13:             flatten down sampled features, $p = \text{flatten}(P)$
14:             for each dense layer do
15:                 compute linear logit, $z(p, w, b)$
16:                 compute activation function $a = \max(0, z)$
17:             end for
18:             compute softmax $\hat{y} = \arg\max(a(z))$
19:             compute loss $f_{CE}(y, \hat{y})$
20:             compute gradients $\frac{\delta}{\delta w} f_{CE}$, $\frac{\delta}{\delta b} f_{CE}$
21:             update model parameters $w, b$
22:         end for
23:     end for
24: end procedure
```

3.3.4 Inference

Using clustering algorithms as the final step in our proposed methodology, we proceeded to estimate or classify the unknown parameters described in Section 3.1 (see block (d) in

Figure 3.3). This was achieved by self-organizing the spatial transmitter location estimates provided by the CNN. Membership in a cluster represents the probability the signal that produced that location estimate was transmitted by the same radio as all other location estimates in that group. The number of active clusters represents the probability of how many radios transmitted the set of all signals detected by the WSN. Finally, the estimated mean of each cluster represents the most probable location of a transmitter.

The Dirichlet Process Gaussian Mixture Model (DPGMM) assumes a set of samples was generated by the following generative conditional distribution [141]:

$$\begin{aligned} p(P_i|\pi_1, ..., \pi_k, \mu_1, ..., \mu_k, s_1, ..., s_k) = \\ \sum_{j=1}^{k} \pi_j \mathcal{N}(\mu_j, s_j^{-1}), \end{aligned} \tag{3.4}$$

where the mixing proportions $\sum_{j=1}^{k} \pi_j = 1$, and observations from Gaussians of mean μ and covariance s^{-1} with larger mixing proportions π are generated more often, and mixing proportions that are small indicate inactive mixtures. Since integrating to evaluate the distributions $p(\pi_1, ..., \pi_k|P_1, ..., P_N) \propto p(P_1, ..., P_N|\pi_1, ..., \pi_N)p(\pi_1, ..., \pi_N)$ are not tractable, inference of DPGMMs is always performed either by Markov Chain Monte Carlo (MCMC) sampling [139], Variational Inference (VI) [140], or Expectation Maximization (EM) [141]. In this work, we employed the optimization-based EM as a relatively hands-off approach when compared to VI or MCMC, which only risks convergence to a local minima [142]. MCMC and VI, on the other hand, risk mode default and poor mixing due to their utilization of Gibbs sampling [142]. Additionally, EM provides faster and more accurate results for when point estimates are sufficient, as in this study [142].

EM optimization of the DPGMM conditional probabilities $\theta = \pi_1, ..., \pi_k, \mu_1, ..., \mu_k, s_1, ..., s_k$ is then provided first by the expectation step, which computes the cluster indicator vector c of each sample given current estimated distribution parameters [141]:

$$p(c_i = z|P_i, \theta) = \frac{\mathcal{N}(\mu_z, s_z^{-1})\pi_z}{\sum_{j=1}^{k} \mathcal{N}(\mu_j, s_j^{-1})\pi_j}, \tag{3.5}$$

where observation P_i belongs to mixture z if $p(c_i = z|P_i, \theta) > p(c_i \neq z|P_i, \theta)$.

Then, inversely, the maximization step is given as the computation of the new Bayesian posterior distributions $\theta = [\mu, s^{-1}, \pi]$ by using the current cluster assignments as:

$$\pi_j = \frac{1}{N}\sum_{i=1}^{N} p(c_i = j|P_i, \theta), \tag{3.6}$$

$$\mu_j = \frac{\sum_{i=1}^{N} p(c_i = j|P_i, \theta)P_i}{\sum_{i=1}^{N} p(c_i = j|P_i, \theta)}, \tag{3.7}$$

$$s_j = \frac{\sum_{i=1}^{N} p(c_i = j|P_i, \theta)}{\sum_{i=1}^{N}(P_i - \mu_j)^2 p(c_i = j|P_i, \theta)}. \tag{3.8}$$

Algorithm 9 shows an overview of the DPGMM EM optimization, including the combination of the expectation and maximization steps.

Algorithm 9 Dirichlet Process Gaussian Mixture Model (DPGMM) inference by EM [141]

```
1: procedure Given observations $y_i$
2:     Initialize $\theta = \pi_1, ..., \pi_k, \mu_1, ..., \mu_k, s_1, ..., s_k$
3:     while not converged do
4:         Expectation step
5:         for $i \in \{1, ..., N\}$ do
6:             for $j \in \{1, ..., k\}$ do
7:                 compute $p(c_i = j|P_i, \theta)$
8:             end for
9:         end for
10:        Maximization step
11:        for $j \in \{1, ..., k\}$ do
12:            for $i \in \{1, ..., N\}$ do
13:                compute $\pi_j|p(c_i = j|P_i, \theta)$
14:                compute $\mu_j|P_i, p(c_i = j|P_i, \theta)$
15:                compute $s_j|P_i, \mu_j, p(c_i = j|P_i, \theta)$
16:            end for
17:        end for
18:    end while
19: end procedure
```

Current wireless localization works that make use of unsupervised learning algorithms [168–171] do so for self-supervised applications, wherein clustering is used to generate labels for unlabeled data such that the costs of training a supervised learning model are reduced. These methods have been shown to reduce the bias and variance of CNN-based fingerprinting estimates.

In previous work, we applied a Gaussian Mixture Model (GMM) to a known number of mobile, simulated wireless beacons [172]. In the following sections, we describe how we applied a modified version of that proof of concept simulation to a practical experiment. There were several differences between the two efforts.

- Mutliateration estimates were produced from real measurements, rather than generated directly from a Gaussian distribution.
- While the Markov process used generated Gaussian samples at each time step with a stationary mean, real-life data sets have a non-stationary mean. This would make the use of mixture models inappropriate, which assume the distributions are stationary.
- We used a Dirichlet prior, which allowed for the inference of an unknown number of clusters instead of a fixed number specified by the user utilizing *a priori* meta data.

Next, we present the implementation details of our novel framework (see Figure 3.3), as well as evaluate its performance.

3.4 Experimental Results

In this section, we describe the implementation details of the proposed CNN-based multilateration, including the method used to generate dataset ground truths, the architecture used, and the training protocol. We subsequently evaluate our entire methodology (Figure 3.3), supplemented by the use of assumption-checking hypothesis testing.

3.4.1 Experiment Implementation Details

Three custom-built Raspberry Pi 3 Model B+ receivers employing NooElec NESDR smart v4 RTL SDR units and isotropic antennas were used to digitize and store the RF signals at 315 MHz emanating from the TPMS sensors of a Subaru Forester 2016 (Figure 3.4). For each TPMS sensor, a three minute RF capture was made per location at 2.5 MSPS complex IQ. The experiment consisted of four TPMS sensors, three custom receivers, nine training vehicle positions, and two vehicle positions used for evaluation, which resulted in 132 data files. The samples were loaded into MATLAB and the built-in resample function was used with a re-sampling factor of 32. This resulted in an output bandwidth of 78.125 kHz which was above the silicon on insulator (SOI) bandwidth and was about 18 samples per symbol for the TPMS SOI. Finally, a time constant of $T = 5$ milliseconds was used in our CFAR detector, which used both training cells and guard cells, but only on one side of the cell under test. We used three guard cells and 10 training cells.

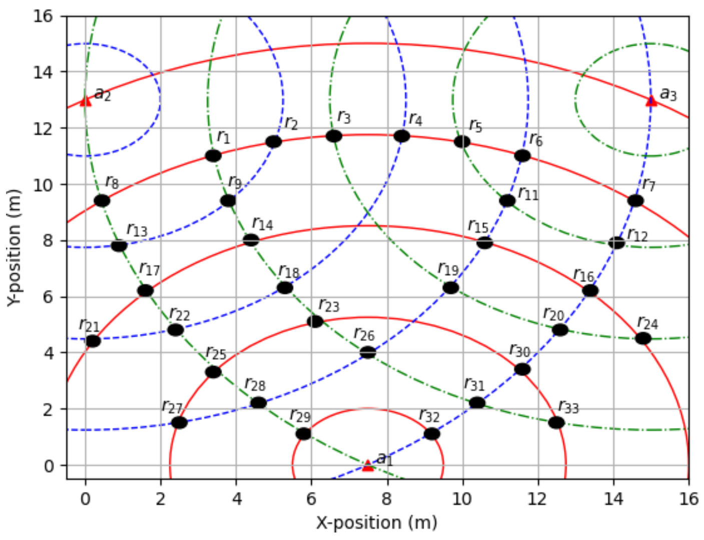
3.4.2 Label Generation Via Surveying

To generate the ground truth labels for the training and testing of our CNN, we surveyed the parking lot located at 358 Pleasant St, Massachusetts, USA (longitude -71^{o}59'41.32"W, latitude -42^{o}34'9.19"N). String was used to draw circles with tied sticks of chalk, anchored down to the center of the circles on each of our three Raspberry Pi units, which were used as the sensing localization network (Figure 3.5). Since each chalk intersection location was known, the grid allowed physical objects to be located in the experimental region by trilateration via hand-measuring three distances to the three nearest chalk intersects (not to be confused with the wireless multilateration performed by our experiment). Measuring distances to more than three reference points would have increased the accuracy of label generation at the cost of more time spent measuring at each point.

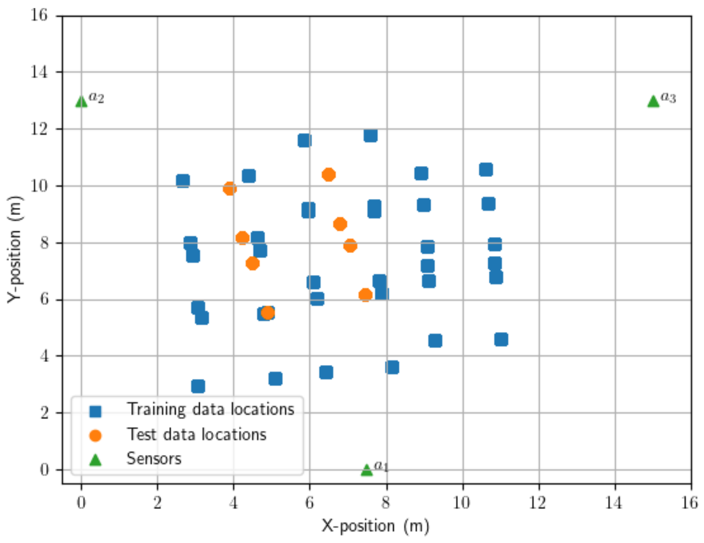
The error for the hand-made range measurements of the 32 grid locations given by Figure 3.5 was found, in an experiment of 32 measurements, to have a mean error of 2.00 cm and an error standard deviation of 1.68 cm. These moments were determined by comparing theoretical geometry (see Figure 3.5) to the actual measurements. This error combines human tape-measure error, which is about 0.5 cm or half a tick mark, and chalk circle intersect misplacement errors caused by slack in the string and its measurement error. Given this statistical description of range measurements, we used the second order Taylor series approximation of the Cramer Rao Lower Bound (CRLB) for multilateration measurements m [173]:

$$\begin{aligned}\underline{CRLB}(\underline{\theta}|\underline{m}) \geq \Bigg[&\frac{d\underline{m}(\underline{\theta})}{d\underline{\theta}}^T \underline{N}(\underline{\theta})^{-1}\frac{d\underline{m}(\underline{\theta})}{d\underline{\theta}} + \\ &\frac{1}{2}tr\left(\underline{N}(\underline{\theta})^{-1}\frac{d\underline{N}(\underline{\theta})}{d\underline{\theta}}\underline{N}(\underline{\theta})^{-1}\frac{d\underline{N}(\underline{\theta})}{d\underline{\theta}}\right)\Bigg]^{-1}.\end{aligned} \tag{3.9}$$

This computation is performed given the trace function $tr(\cdot)$, anchor-beacon distance matrix $\underline{\theta}$, the gradient of measurement covariance with respect to beacon location $\underline{N}(\underline{\theta})$, and the gradient of measurements with respect to beacon location $\frac{d\underline{m}(\underline{\theta})}{d\underline{\theta}}$ to compute the CRLB of training labels gathered for the fingerprint data. This yielded a standard deviation of 16.8 cm on the x axis and 17.3 cm on the y axis for our ground truth labels.



(a) Surveyed reference locations



(b) Anchor and beacon placements

Figure 3.5: The true location of beacons was measured by adding the measured distance of the beacon from these known intersect points (a) and the anchor and beacon locations for the experiment (b).

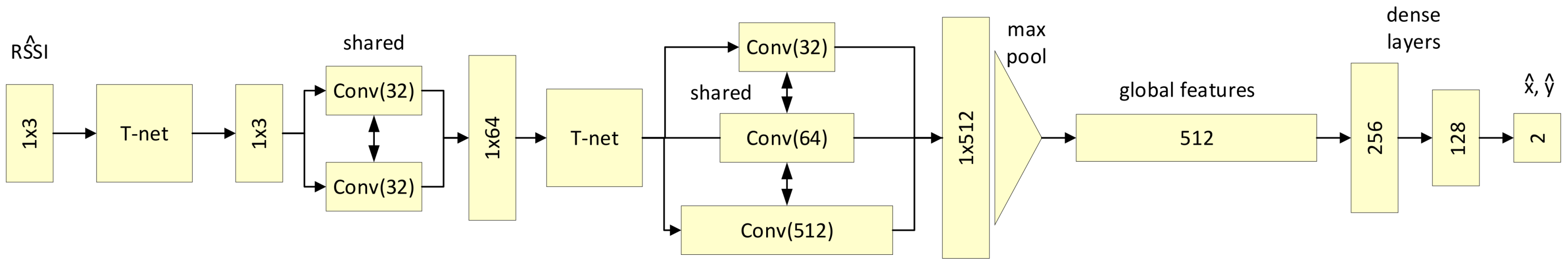


Figure 3.6: An overview of the PointNet CNN architecture, which we modify only by changing the 10-class softmax output to a two-dimensional linear output. A dropout with $p = 0.3$ is used after each dense layer. Each T-net is a mini-network that aims to learn a affine transformation matrix. PointNets are designed for low-dimensional data and physical space. We employed a bagging ensemble of three PointNets for this experiment.

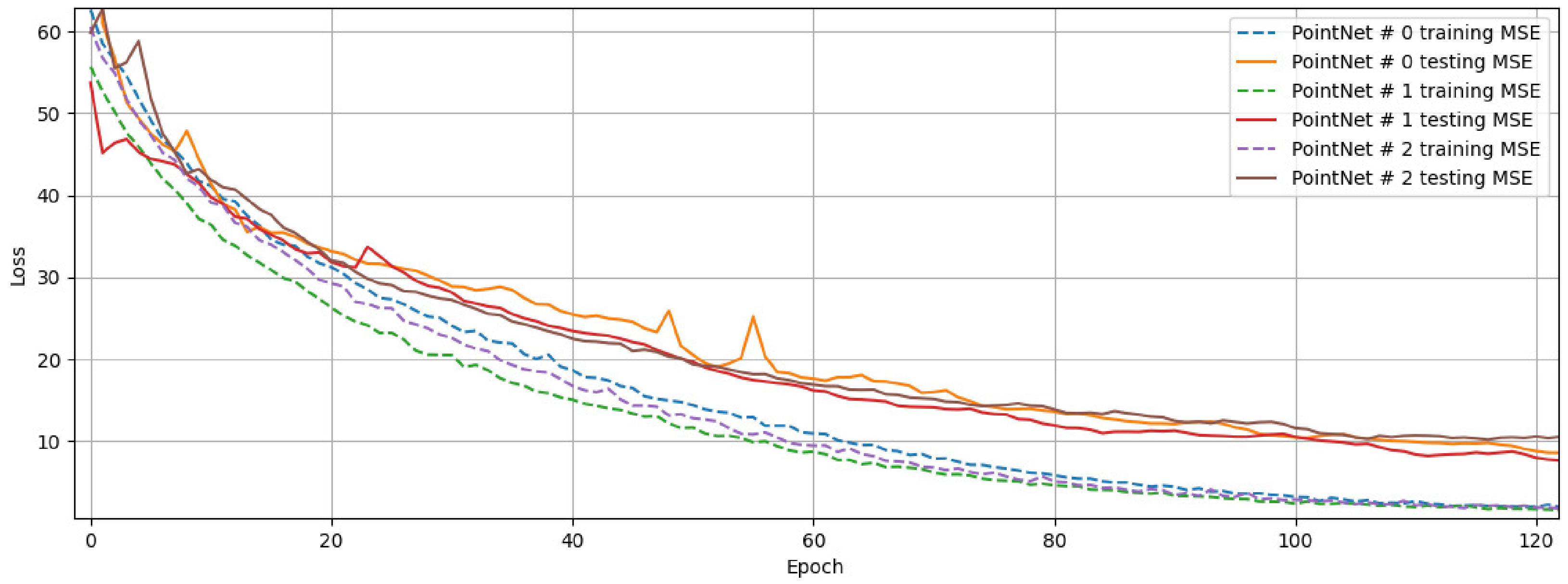


Figure 3.7: The transient epochs of MSE training and validation loss of the CNN. Final training MSE for the ensemble was 0.89 meters, and a final test set MSE of 4.87 meters. CRLB results found that training and test labels have a minimum error of 0.168 meter and 0.173 meter along the x and y axis, respectively.

3.4.3 CNN Training

The CNN architecture developed for this work (Figure 3.6) was designed using a modified PointNet [174] architecture and a host of over-fitting countermeasures to combat the high noise, low number, and large shift in statistics between training and testing data and labels. The PointNet was terminated with a two neuron linear layer, whose outputs represented the estimated two-dimensional location of the transmitter whose range estimates were input. The model's three-dimensional input represents a set of RSS estimates from the perspective of each sensor for a single detected signal of unknown origin. The PointNet was comprised of two T-nets, each of which is a mini-network comprised of a 32, 64, and 512 filter convolutional layer, a global max pooling [175] layer, a 256 and 128 neuron dense layer, and final dense layer with a number of neurons equal to the squared number of features. We did not change the architecture's depth or width because our inputs are of the same dimensionality as the three-dimensional spatial data that this state-of-the-art architecture was optimized for. These T-nets utilized L2 [119] regularizers ($\lambda = 0.001$) to learn an affine transformation matrix. Each convolutional layer has a kernel size of one, valid zero padding [176], a Rectified Linear Unit (ReLU) activation function, and Kaiming [177] kernel initialization. Additionally, the PointNet was comprised of two sets of convolutional layers with shared weights, a global max pooling layer, and two dense layers with 256 and 128 neurons.

We trained three PointNets in a bagging ensemble [178] learning scheme, which can significantly improve the performance of the trained SL models. A bagging ensemble of SL models is a collection of n models independently trained on the same data set, not to be confused with boosting or stacking ensembles. Bagging ensembles perform regression by averaging their predictions on test data. The benefit of this can be shown by minimizing the expected squared error of predictions, defined as [179]:

$$E\left[\left(\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\right) - y\right)^2\right] = \frac{v}{n} + c\frac{n-1}{n}, \tag{3.10}$$

where the variance $v = E[(\hat{y}_i - y_i)^2]$, and the covariance $c = E[(\hat{y}_i - y_j)^2], i \neq j$. If the covariance is zero, or each model makes different mistakes, then the gain of using an ensemble on the squared error of predictions is $\frac{1}{n}$. If $v = c$, then the ensemble brings no gains, and the prediction MSE remains at v.

The weights were optimized in TensorFlow [108] by minimizing training data MSE loss via the Adam [180] quasi-Newton SGD method (Figure 3.7). The relatively large difference between the ensemble's loss and each weak learner's loss indicated a low covariance c as described by Eq. (3.10), or that our use of a bagging ensemble was appropriate and beneficial towards making accurate multilateration estimates. Data was trained in batches of the whole training set, which was of size 601 detected TPMS packets over all 9 locations and 4 transmitters, and batch normalization [123] was used. No early stopping was implemented over the 500 epochs of training, at which point the model loss converged. An Intel Core i7-10750H CPU and 16GB of RAM were sufficient to train this relatively shallow network and low-dimensional data.

3.4.4 CNN Testing & UL Inference

The deployment stage of the proposed SOP beacon localization system used the hardware testbed and algorithms described in this chapter. Since physical signal features were estimated, range estimates and subsequently multilateration-based location estimates were random variables [173]. Specifically, location estimates are known to be distributed as multivariate Gaussian variables (even if observations are non-Gaussian [181–184]) such that $\hat{P}_i \sim \mathcal{N}(\mu_j, \Sigma_j)$ for a number of multilateration estimates $i = 1, ..., N$ and a number of SOP beacons $j = 1, ..., k$ [181–184]. Consequently, we utilized a DPGMM as a probabilistic inference model with which to assign the locations estimates $\hat{P}_i$ to k sub-populations. Each sub-population or cluster of estimates represents the spatially-inferred probability that the j^{th} wireless beacon transmitted the i^{th} multilateration estimate (Figure 3.3). This inference and assignment method differs from the heuristic methods [53, 55] discussed in Section 3.1 since it is automated, leverages the underlying statistics of the data, and is not overly sensitive to hyper-parameter choices.

The evaluation of the experiment can be seen in Figure 3.8, in which $N = 229$ detected TPMS packets were localized using the trained CNN. The Autel prompting radio [185] was used for about three minutes at each TPMS beacon location, and the vehicle was moved once for a total of eight beacons, on which we superimposed the multiateration estimates in post-processing. We implemented the DPGMM with $k = N$ such that every data point may become a cluster if its mixing proportion $\pi_j, j = 1, ..., k$ is sufficiently large.

We found the multilateration estimates possessed a high error relative to the distance

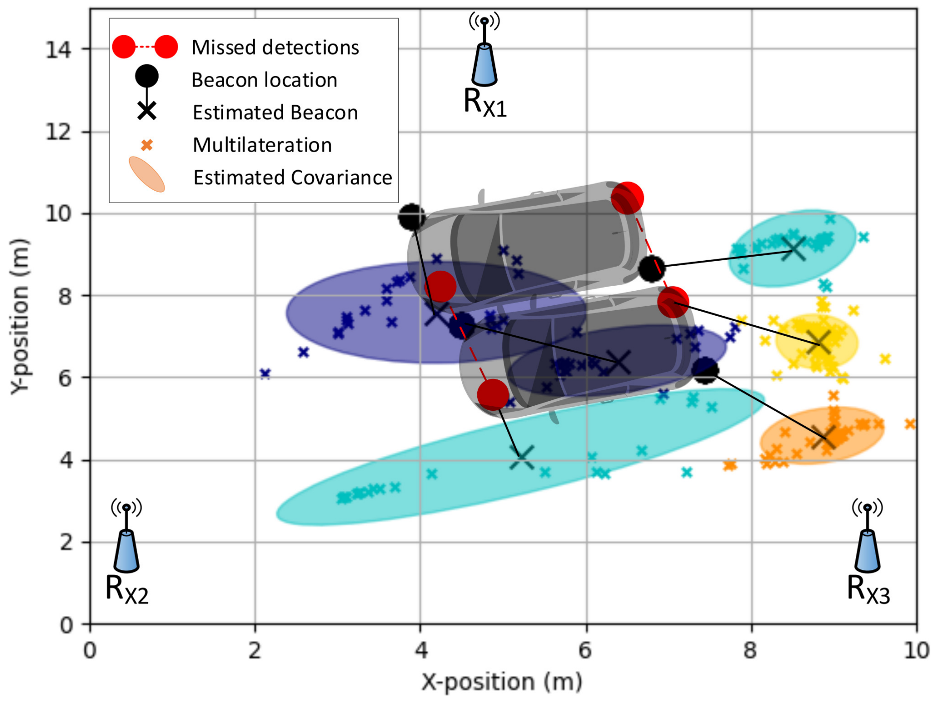
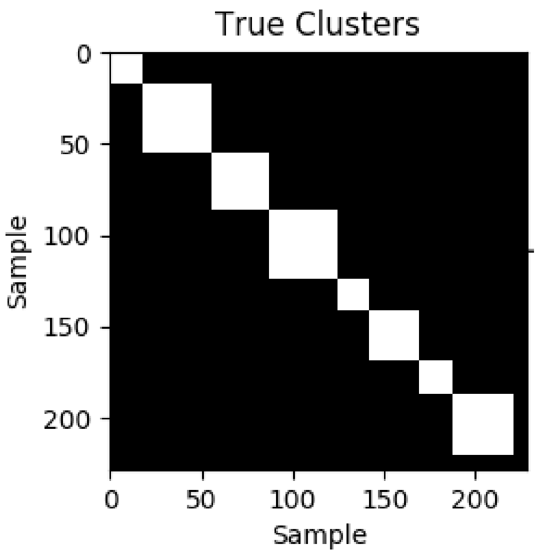


Figure 3.8: A summary of the $N = 229$ spatial multilateration estimates and the estimated cluster means and co-variances of the DPGMM. "Missed detections" represent pairs of transmitters whose spatial and RSS estimates could not be statistically distinguished.

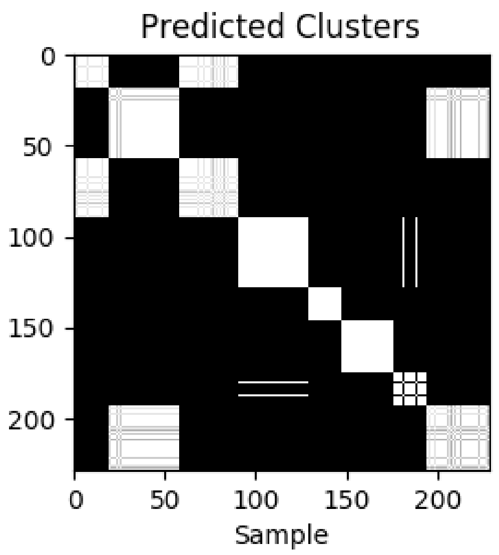
between beacons. This could potentially be the result of the significant variation in the RSS estimates, the 22.5 dBi directional TPMS antenna [186], or our use of only three receivers, the minimum number required for two-dimensional localization.

The two pairs of RSS populations that could not be differentiated by the EM-based DPGMM clustering were not neighboring locations, but the front-left tire's transmitter and the back-right tire's transmitter at two locations. This indicated a relatively higher correlation between the transmit power and received power than the transmitter location and received power. While only six of the eight RSS populations could be distinguished by the DPGMM, this is actually a typical performance given the small number of sensors, the small size and high variance of the RSS estimates, and the low-power, highly directional design of the transmitters.

We performed a Kruskal-Wallis (kW) [187] test, a non-normal method for determining if two sets of samples originate from the same distribution. The test statistic is defined as:



(a) True cluster



(b) Predicted cluster

Figure 3.9: A summary of the test set's $N = 229$ detected TPMS packet multilateration estimates true indicator variables $c_i = j, i = 1, ..., N, j = 1, ..., k$ (a) and our predictions (b). Any sample from one axis with the same indicator variable as a sample from the other axis is colored white.

$$H = (N-1)\frac{\sum_{i=1}^{g} n_i(\bar{r}_i - \bar{r})^2}{\sum_{i=1}^{g}\sum_{j=1}^{n_i}(r_{ij} - \bar{r})^2}, \tag{3.11}$$

where N is the number of samples across all groups g, n_i is the number of samples in group i, r denotes rank, and $\bar{\cdot}$ denotes mean. For a critical statistic H^* computed by table lookup for $\alpha = 0.05$, we found $H < H^*$ on the RSS populations of each pair of beacons that made up a missed detection, or that we could not reject the null hypothesis. We could not find evidence the estimates were statistically different. On average, the six beacon estimates were evaluated with ground truth values to have a 1.72 meter variance ($E[(\hat{P} - P)^2]$) and 1.19 meter bias ($E[|\hat{P} - P|]$).

Insight into the performance of the DPGMM is provided by the confusion matrix which quantifies the accuracy of the expectation step, *i.e.* classification of which transmitter produced which signals (see Figure 3.9). The confusion matrix shows the true mixing proportions of the data (Figure 3.9a) and that some transmitter captures produced more detected signals from the localization sensors than others. That number, from top left to bottom right of the confusion matrix, is $[18, 38, 31, 38, 17, 27, 18, 34]$ detected TPMS packets

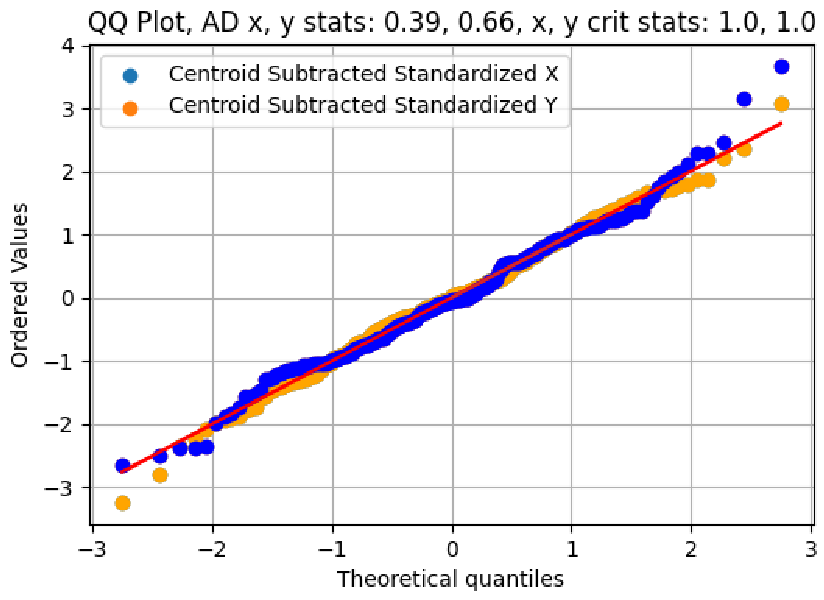


Figure 3.10: QQ plot of the cluster mean subtracted, standardized PointNet multilateration estimates. Additionally, the axis-specific AD statistics and critical statistics are provided.

for each of the three-minute captures at the eight test locations. Alternatively, the confusion matrix also shows that only six of the eight clusters were detected (Figure 3.9b), as, starting from the top left, if squares are named one through eight, it can be seen the first and third clusters, as well as sixth and eighth clusters are combined. Finally, we can see that two additional multilateration estimates from the seventh beacon are misclassified, producing an indicator variable classification precision or Positive Predictive Value (PPV) of 76.4%. When incorrect cluster assignments caused by missed cluster detections are removed, the PPV is 93.7%. PPV is defined as the number of true positives divided by the sum of true and false positives [188]:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{3.12}$$

To gather evidence that our PointNet's location estimates were Gaussian distributed and our use of a Gaussian inference model was suitable, we utilized the AD [189] test for determining if a set of samples are drawn from a particular distribution. To do so, we first subtracted the estimated cluster mean from each population of estimates, then standardized them:

$$Y_i = \frac{(\hat{P}_i - \hat{\mu}_j) - \frac{1}{g}\sum_{i=1}^{g}(\hat{P}_i - \hat{\mu}_j)}{\frac{1}{g-1}\sum_{i=1}^{g}(\hat{P}_i - \frac{1}{g}\sum_{i=1}^{g}(\hat{P}_i - \hat{\mu}_j))^2}. \tag{3.13}$$

This calculation produced a set of samples that, when tested, whether the estimates were Gaussian with respect to their cluster center. The AD critical value for normal tests can be computed as [190]:

$$A^{*2} = A^2\left(1 + \frac{0.75}{N} + \frac{2.25}{N^2}\right), \tag{3.14}$$

and the AD statistic is computed as [190]:

$$\begin{aligned} A^2 = & -N - \frac{1}{N}\sum_{i=1}^{N}(2i-1)(\ln\Phi(Y_i)+ \\ & \ln(1-\Phi(Y_{N+1-i}))), \end{aligned} \tag{3.15}$$

where Φ is the standard normal Cumulative Distribution Function (CDF). Our AD results are presented in the Quantile-Quantile (QQ) plot in Figure 3.10, which visually and quantitatively show that PointNet multilateration estimates are Gaussian distributed due to the diagonal trend line and small AD test statistics.

3.5 Chapter Summary

In this chapter, we described our “black box” localization problem statement in Section 3.2. Then, we presented the motivation and methodology details for utilizing SL and UL in transmitter localization in Section 3.3. Finally, we discussed the implementation details of our methodology, and evaluated the training and testing of our the proposed framework in Section 3.4.

The opportunistic, self-organizing, stationary transmitter, jointly localizing and classifying methodology demonstrated in chapter 3 that:

- Locating transmitters in low meta data scenarios with high accuracy.
- The $N = 229$ spatial test set was able to recognize every statistically different RSS population as verified by a Kruskal-Wallis (kW) test.
- That test classified the source radio of transmissions with a 76.4% indicator variable PPV (93.7% when accounting for missed cluster detections).
- The detected transmitters were localized with an average of 1.71 meter variance and 1.19 meter bias within a roughly 15 meter square whose perimeter is made up of receivers.

- We confirmed the Gaussian assumption of our clustering algorithm by failing to deny that the PointNet CNN's location predictions were Gaussian distributed. We accomplished this verification via an AD test with critical statistics $A^{*2} = 1.0$ and observed statistics $A^2 = 0.39(0.66)$ for x-axis (y-axis) samples.

Our methodology allows for the post-multilateration clustering of location estimates such that unknown wireless devices present in driverless vehicles may be opportunistically leveraged for localization tasks, as long as they broadcast detectable emissions from which physical passband characteristics may be estimated from. This lowers reliance on GNSS systems and SOP localization networks that require meta data.

Chapter 4

Physical Eavesdropper Evasion: Signal Dependent Perturbation Design and Adversarial Training

4.1 Introduction

In this chapter, we present a number of novel contributions to the state-of-the-art. These contributions confirm as well as deny state-of-the-art best practices from the adversarial computer vision domain, as well as establish new ones for wireless communication scenarios through both simulated and experimental demonstrations. This chapter is organized as follows:

- In Section 4.2, we introduce the real-world physical scenario that motivates adversarial attack of a signal classifier and define metrics of success for such an adversary. We proceed to describe the method of generating offline training data, capturing of online physical testing data, and modulation classification architectures used. Finally, we briefly survey adversarial threat models and three popular perturbation types for readers with a understanding of wireless communications but without an understanding of perturbations.
- In Section 4.3, we present two novel contributions to wireless adversarial perturbation design that have been over looked by previous works; constraining perturbations to add constructively with transmitted signals, and constraining perturbations to the

same bandwidth as transmitted signals. Both constraints are beneficial due to the dual-objective nature of the transmitter in the wireless scenario described by Section 4.2, which are to minimize the Bit Error Rate (BER) of the intended receiver while also minimizing the eavesdropper's classification precision, which is also called Positive Predictive Value (PPV). In other data domains, the use of adversarial perturbations is typically a single-objective scenario in which minimizing the PPV of the classifier is the only concern. However, the single-objective scenarios is not valid in our domain and we define additional metrics in this work.

- In Section 4.4, we present several novel studies of state-of-the-art computer vision adversarial training schemes applied to the problem of wireless adversarial perturbation defense. We validate these schemes experimentally using NI Ettus Research's USRP N210 Software Defined Radios (SDRs). We implement the state-of-the-art adversarial training scheme, compare state-of-the-art parallel and cascade variations of that training scheme, investigate the effectiveness of state-of-the-art iterative attacks, analyze the effect of adversarial training as a regularizer by varying model capacity, and investigate the presence of label leaking in ground truth based adversarial training schemes.
- In Section 4.5, we review our contributions and offer up several open challenges to the wireless security community.

4.1.1 Assumptions

In this work, we explore defense approaches against adversarial perturbations in a current step, white box attack regime, and consider adversarial training as the most effective and appropriate defense. By "current step", we mean the adversary may always refresh their information of the eavesdropper's system, and it is always knowledgeable of the classification networks current parameters. For example, if adversarial training is implemented, we evaluate the model against perturbations crafted from the adversarial trained model, not the non-adversarial trained model. We reason that if an adversary can observe the eavesdropper's radio and ML systems once, then it can be observed again in the future. We note that very few adversarial training works operate using this challenging but realistic assumption. In non-current step approaches, the lack of the adversary's ability to see the final network can potentially yield especially misleading results in gradient masking [62] works, where adversarial detectors are unaware of supplemental classification schemes being used.

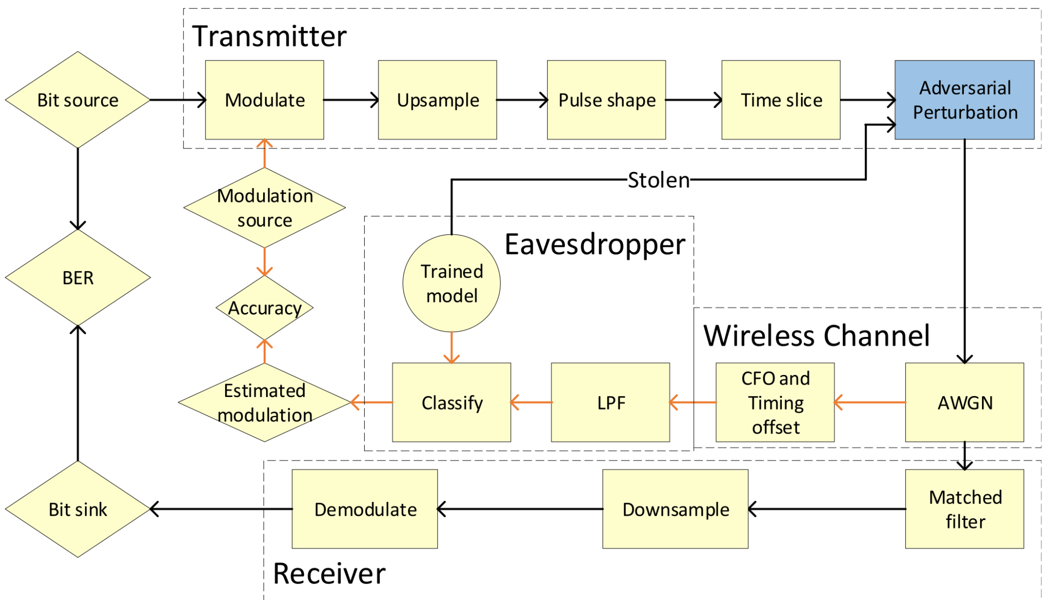


Figure 4.1: The transmitter, given both a signal and a perturbation power constraint, strategically amplifies certain samples of signals such that an adversarial eavesdropper cannot correctly classify the modulation scheme of the observed signals. When successful, the transmitter avoids being demodulated correctly and its bits estimated by the eavesdropper are random and lack any information. We measure the success of perturbations by how low of a BER they achieve with the intended receiver and by how low of a classification accuracy (PPV) the eavesdropper achieves in this dual-objective scenario. Conversely, we measure the success of the eavesdropper by how high of a classification PPV it can achieve on observed signals and how many bit errors it can force the transmitter to make in order to avoid correct demodulation.

Traditionally, the term “white box” is used in adversarial perturbation scenarios to describe the weights and architecture of the target ML classifier as fully observable by the agent who is crafting perturbations. In our white box scenario, the adversary can observe not only the trained model and its weights, but all aspects of the eavesdropper’s radio and ML systems, such as perturbation detection networks or ensemble classification schemes. Consequently, we do not investigate the use of semi-supervised perturbation detection algorithms [64] because it has been shown that they increase the attack surface of the classifier when the adversary is aware of them [61]. We do not investigate the use of gradient masking [62] or defensive distillation [66] because the process of fooling these networks is well understood [83, 84, 191, 192]. Finally, we do not investigate the use of network verification [193–196] as these methods are still prohibitively computationally expensive for all but the smallest datasets and models. While our classification architectures are relatively small (see Section 4.2.1 and Section 4.2.2), we show in Section 4.4.5 that making them any smaller such that network verification would be possible, will make them more vulnerable to adversarial perturbations.

Adversarial training has been described as a powerful regularization method [5] that performs a similar function to L_1 regularization on the activations of linear classifiers [59]. When a model is overfitting, adversarial training, defensive distillation, and gradient masking schemes have all doubled as defenses against adversarial perturbations, as well as regularizers that increase the classification PPV of non-adversarial test data.

4.2 System Model

The state-of-the-art perturbation approaches assume one of two different three-player scenarios. In the first scenario, a transmitter and receiver communicate while a reactive adversary eavesdrops, computes the proper perturbation, then synchronously transmits those perturbations in order to fool the receiving radio, which classifies the observed sum of signals [76, 77]. In the other, a transmitter adds pre-channel perturbations to fool an adversarial eavesdropper, which classifies the transmissions while maximizing communication capabilities between the transmitter and receiver [78–80]. In this work, we investigate the latter scenario (Figure 4.1), which we consider the only realizable white box scenario of the two as it is a significant challenge for an eavesdropper to receive a signal, process it, and transmit a perturbation which adds together with the original transmission at the intended receiver with no delay [76]. Due to the disjoint research resulting in the contributions presented in this work, we present in the following subsections two different datasets and modulation classification architectures.

4.2.1 Signal Dependent Waveforms and Classifier

Adversarial perturbations in the wireless communications domain are ubiquitously generated to fool classifiers trained on the RML2016.10A [143] simulated dataset. The dataset is the first of two [144] modulation classification datasets, classifying 128-sample In-phase Quadrature (IQ) signals (256 input features/example) to one of 11 modulation classes. The signals are altered by a noisy channel model simulated by the GNU Radio (GR) dynamic channel model, which applies a truncated Gaussian-distributed random walk to the symbol rate and carrier frequency offset, a Rician distributed frequency-selective fading multipath model, and a Gaussian-distributed AWGN model. Previous papers that apply adversarial attacks to this dataset also used the same supervised learning model to compute perturbations, the VT-CNN2 [2] model. This has simplified the comparison of various papers.

In this work, we create variations inspired by the RML2016.10A [143] and RML2018.01A [3] datasets, for use in Section 4.3 and Section 4.4, respectively. The RML2018.01A dataset expands upon its predecessor by classifying on a higher number of modulation classes, generating data via a wireless channel model with a greater number of noise sources, and more complex feature engineering that allow for modulation classification using not just IQ data, but also moments and cumulants. In the related work [86], the RML2016.10A [143] dataset experiences significant multicollinearity, uses a pulse shaping filter without zero-crossings, and does not properly compute Signal-to-Noise-Ratio (SNR) ratios:

$$\mathrm{SNR} = 10\log_{10}\left(\frac{I \times \sum |x|^2}{\sum |n|^2}\right). \tag{4.1}$$

Consequently, we chose to create a new dataset similar to RML2016.10A in which we lower the Samples Per Symbol (SPS), I, from 8 to 2 to generate more symbols per signal time slice, and increase the number of samples per signal capture from 128 to 1024 to reduce multicollinearity. We compute AWGN as a circularly symmetric Gaussian random variable:

$$n \sim \mathcal{N}(0, \sigma^2) + 1j \times \mathcal{N}(0, \sigma^2), \tag{4.2}$$

where the standard deviation $\sigma = \sqrt{\frac{10^{-SNR/10}}{2}}$ is used instead of $\sigma = 10^{-SNR/10}$ [3] to properly compute the SNR. Finally, we implement a slightly different Finite Impulse Response (FIR) Root Raised Cosine [197] (RRC) with a rolloff $\alpha = 0.35$ and 12 taps since the RML2016.10A does not possess zero crossings in its RRC filter, as the dataset is not designed for bit estimation.

Our version of the dataset also assumes no multipath instead of a three-tap scenario with delays $\tau = [0.0, 0.9, 1.7]$ and magnitudes $\rho = [1, 0.8, 0.3]$ since we believe their multipath model to be overly specific and our test data generated via physical experimentation will utilize a coaxial cable to connect radios, thus experiencing little-to-no multipath effects. For additional ease of bit estimation, we deviate from the original dataset by applying all channel effects without the use of GR channel models, since GR generates streams of data that are difficult to align and are dynamically changing in size to optimize computational

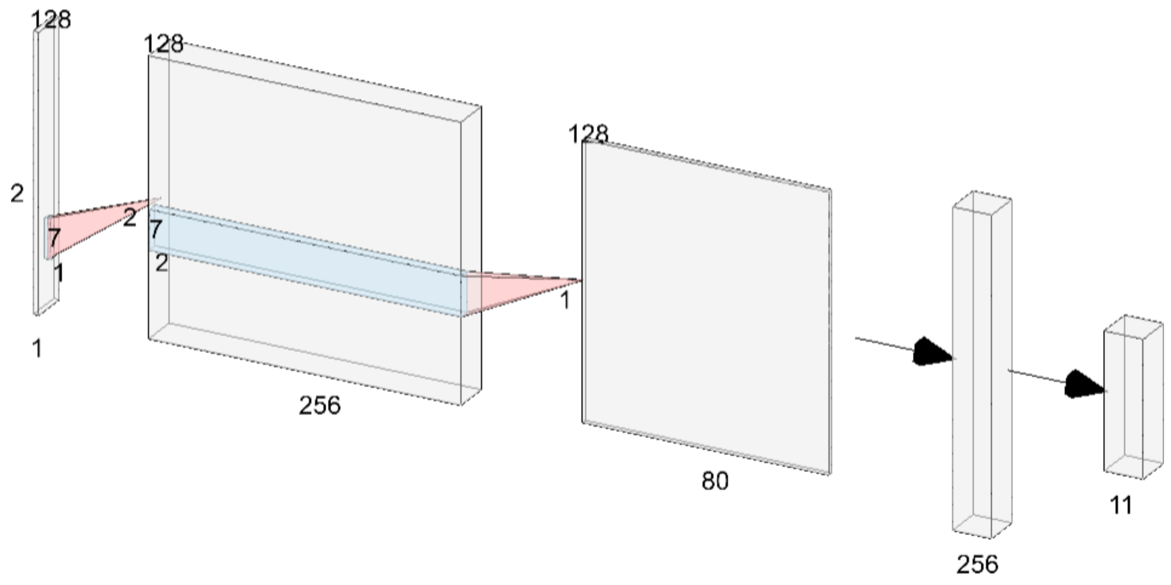


Figure 4.2: The VT-CNN2 [2] modulation classifier model used in Section 4.3.

efficiency. Specifically, we implement the cumulative random walk of truncated Gaussian samples for Symbol Rate Offset (SRO) as:

$$\begin{aligned} x_{\text{SRO}}[n] &= \mathcal{F}^{-1}\Bigg\{\mathcal{F}\{x\}\Bigg(\cos\left(\frac{2\pi n}{N}\sum_{i=1}^{n}\text{SRO}_i\right) - 1j\sin\left(\frac{2\pi n}{N}\sum_{i=1}^{n}\text{SRO}_i\right)\Bigg)\Bigg\}, \\ \text{SRO} &\sim \text{Clip}_{0,-50,50}\{\mathcal{N}(0,0.01)\}. \end{aligned} \tag{4.3}$$

We define the clip function $\text{Clip}_{x,-\delta,\delta}\{A\}$ as the element wise re-sampling of $A_{i,j}$ to the range $[x_{i,j}-\delta, x_{i,j}+\delta]$. The cumulative random walk of truncated Gaussian samples for CFO is defined as:

$$\begin{aligned} x_{\text{CFO}}[n] &= x_{\text{SRO}}[n] \times \exp\Bigg(\frac{-2j\pi n}{f_s}\sum_{i=1}^{n}\text{CFO}_i\Bigg), \\ \text{CFO} &\sim \text{Clip}_{0,-500,500}\{\mathcal{N}(0,0.01)\}, \end{aligned} \tag{4.4}$$

for sample index $n = 1, ..., N$, vector length $N = 128$, and sample rate $f_s = 200$ MHz, which are all equal to the values used in [143].

Finally, we implemented an 11th order FIR Butterworth [198] filter with a normalized frequency cutoff of 0.65 to isolate the signals from OOB noise. Both filters are applied via the convolution operator.

Wireless adversarial perturbation works ubiquitously craft their signals from the open-sourced modulation classification dataset RML2016.10A [143] and the CNN model named VT-CNN2 [2] (Figure 4.5) produced in [199]. Consequently, we will use this architecture and dataset in our perturbation crafting in Section 4.3. The shallow VT-CNN2 network is constructed using Keras [200], which is comprised of just two convolutional layers with 256 and 80 filters, respectively, as well as kernel sizes (7,1) and (7,2), respectively. The model is terminated with a dense layer with 256 neurons and a softmax classifier layer consisting of 11 outputs. Each convolutional layer utilizes valid padding, dropout [201] with $p = 0.5$, Glorot (also known as Xavier) uniform weight initialization [202], and Rectified Linear Unit (ReLU) activations [203]. The first convolutional layer has no bias terms, and each dense layer is "He" initialized, totaling in 2.9×10^6 parameters. We did not observe that weight regularization improved classification performance.

The model is optimized using Tensorflow [204] by minimizing categorical cross entropy via the Adam [205] quasi-Newton Stochastic Gradient Descent (SGD) method over 100 epochs in mini-batches of size 1024. The training is stopped early when the validation loss has not decreased for 5 epochs, saving the lowest validation loss weights. The following Adam [205] parameters are used: $\alpha = 0.001$, $\beta = (0.9, 0.999)$, $\epsilon = 1 \times 10^{-7}$, $\gamma = 0.0$ with no warm up steps.

4.2.2 Adversarial Training Waveforms and Classifier

During the development of this work, we found the open-sourced generation code for RML2018.01A [3] dataset also experiences multicollinearity and incorrectly labels some generated signals. To address the issue of multicollinearity, we used time slices of 4096 complex IQ samples instead of 1024. By trial-and-error, we found by using more samples per signal, we did not need to generate as many signals to achieve the same test PPV, such that our training dataset contains 1.4 million signals instead of 2 million [3]. We additionally implemented the dataset with the following differences from the GR channel model presented in [3]: $\alpha \sim \mathcal{U}(0.35, 0.45)$ instead of $\alpha \sim \mathcal{U}(0.1, 0.4)$, $\sigma_{\text{clk}} = 0.005$ instead of $\sigma_{\text{clk}} = 0.01$, $\tau = [0.0]$ instead of $\tau = [0.0, 0.5, 1.0, 2.0]$ and SNR in the range $E_s/N_0 \in [0, 30]$ dB instead of $E_s/N_0 \in [-20, 30]$ dB. This smaller range of channel models were chosen because of the use of a coaxial cable in physical data generation rather than wireless connection between radios.

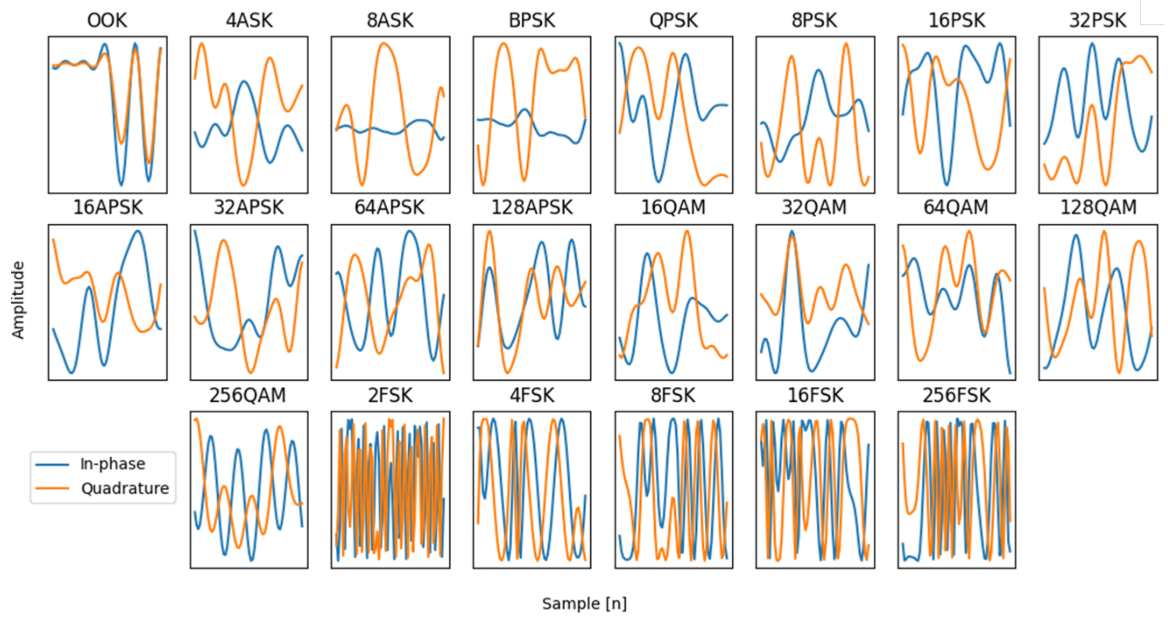


Figure 4.3: One captured signal for each modulation class from the connected USRP N210 SDRs using our RML2018.01A [3] inspired dataset. These visualizations are oversampled by 8 SPS and display only the first 32 samples of the 4096 sample signals for visualization purposes.

Since the training of models to be robust to perturbations is an adjacent task to training models that generalize well to test data that differs statistically from training data [5], we generated several physical test sets (Figure 4.3) to see how well our adversarial training schemes perform as regularizers. These test sets are comprised of 1408 signals, also called examples, where each of the 88 GR channel models generate 16 signals, as opposed to the training sets wherein 2728 GR channel models generate 512 signals each. Each GR channel model has a fixed modulation class, SNR, and SPS. To generate the data, two USRP N210 SDRs (Figure 4.4) are connected via coaxial cable. No digital gain or digital attenuation is used, the radios sample captures using a 1 MHz bandwidth and 20 MHz carrier frequency. We add perturbations to these streams of data via a synchronous Out-Of-Tree (OOT) block in GR implemented before the USRP sink block, which loads the trained Pytorch model, predicts and computes the gradient using the time slice of data, and adds the perturbation to the output. For consistency, we enforce a unit energy constraint on all datasets before classification and perturbation crafting as:

$$x_{norm} = \frac{x\sqrt{n}}{||x||_1}, \tag{4.5}$$

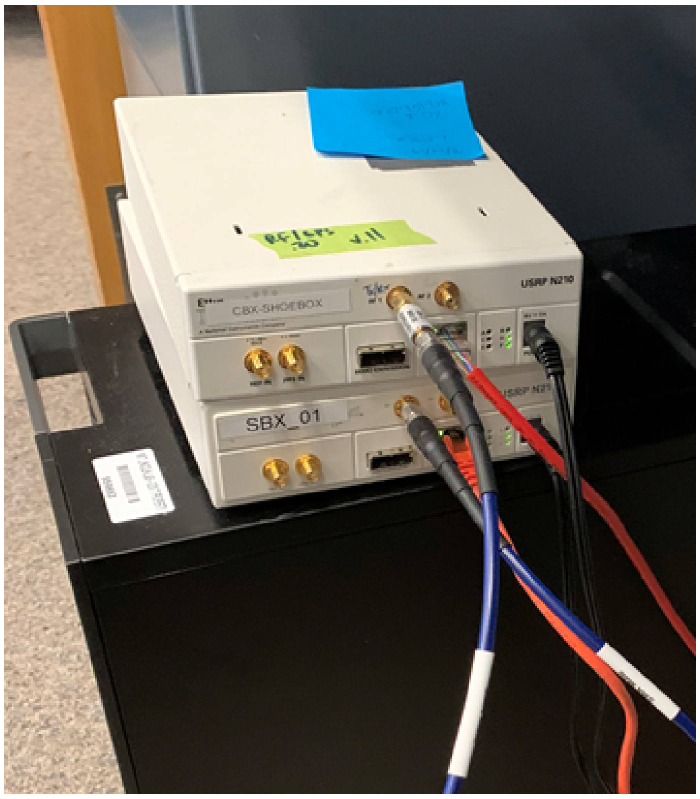


Figure 4.4: USRP N210 SDRs, their coaxial connection, and host computer. The connection employs a 10 dB attenuator.

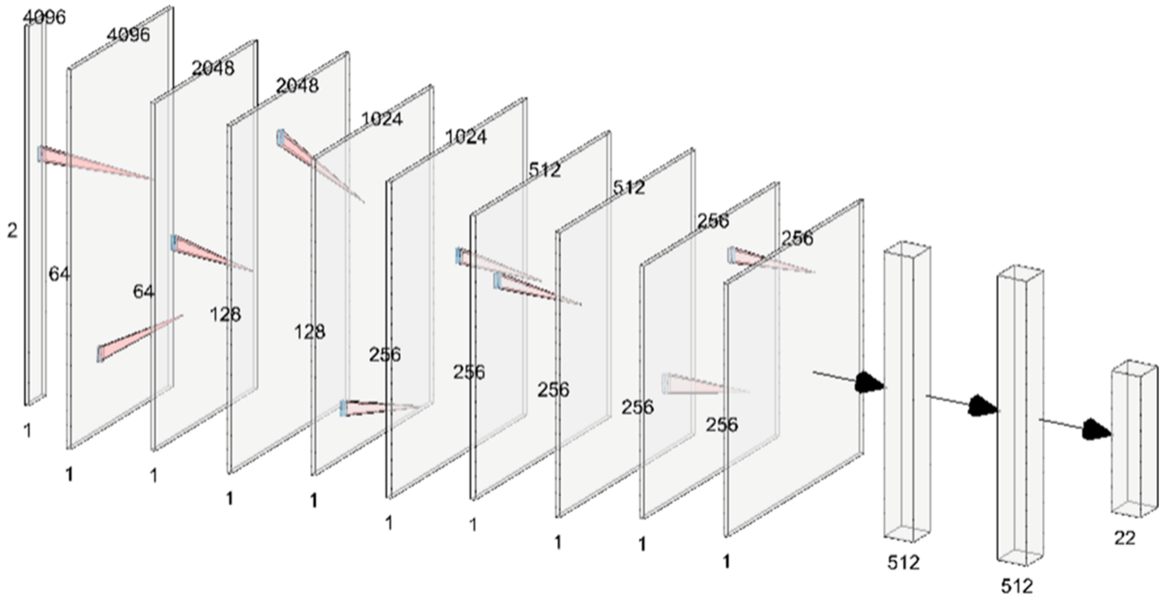


Figure 4.5: The VGG10 [4] modulation classifier model used in Section 4.4.

where n is the length of x. The received test signals do not have an observable DC offset, such that mean subtraction is not implemented in simulated data.

For our adversarial training presented in Section 4.4, we implemented a deeper model in Pytorch [206], which possesses a higher learning capacity necessary for this work (see Section 4.4.5) when performing adversarial training and training using a dataset with a higher number of classes. We used, as found by trial-and-error, a deep model inspired by the Visual Geometry Group (VGG) 10 [4] CNN model (Figure 4.2) comprised of 9 convolutional layers with ReLU activations [203] and the following number of filters per layer: $[64, 64, 128, 128, 256, 256, 256, 256, 256]$. The model is terminated with two dense layers with 512 neurons and 22 outputs from the second dense layer. Max pooling is implemented every two layers with stride 2 and size 2. All convolutional layers used have stride 1 and kernel size 3, totaling 18.2×10^6 parameters. All weights are initialized using Kaiming initialization [177]. We did not find that dropout [201] and weight regularization improve classification performance.

The model is optimized in Pytorch [206] by minimizing log softmax plus categorical cross entropy loss via the Adam [205] quasi-Newton method over 20 epochs in mini-batches of size 256. The following Adam [205] parameters are used: $\alpha = 0.0442$, $\beta = (0.9, 0.98)$, $\epsilon = 1 \times 10^{-9}$, $\gamma = 0.1$ with 4,000 warm up steps. No early stopping is implemented, and batch normalization [123] is used. Currently, it was observed the strongest perturbations are those crafted exploiting Neural Networks (NNs) with skip connections, also known as ResNets [207], as a new attack surface [208]. Consequently, we opt to forgo skip connections, despite their advantage in training deep models that are robust to vanishing gradient problems.

4.2.3 Adversary Goals and Description

An adversarial perturbation is defined as a signal is added to another signal which is given to a ML model during either training or testing with the intent of causing incorrect estimation or classification during the testing phase of the ML model's deployment. This interaction can be generally described as:

$$x^* = x + \epsilon\eta, \tag{4.6}$$

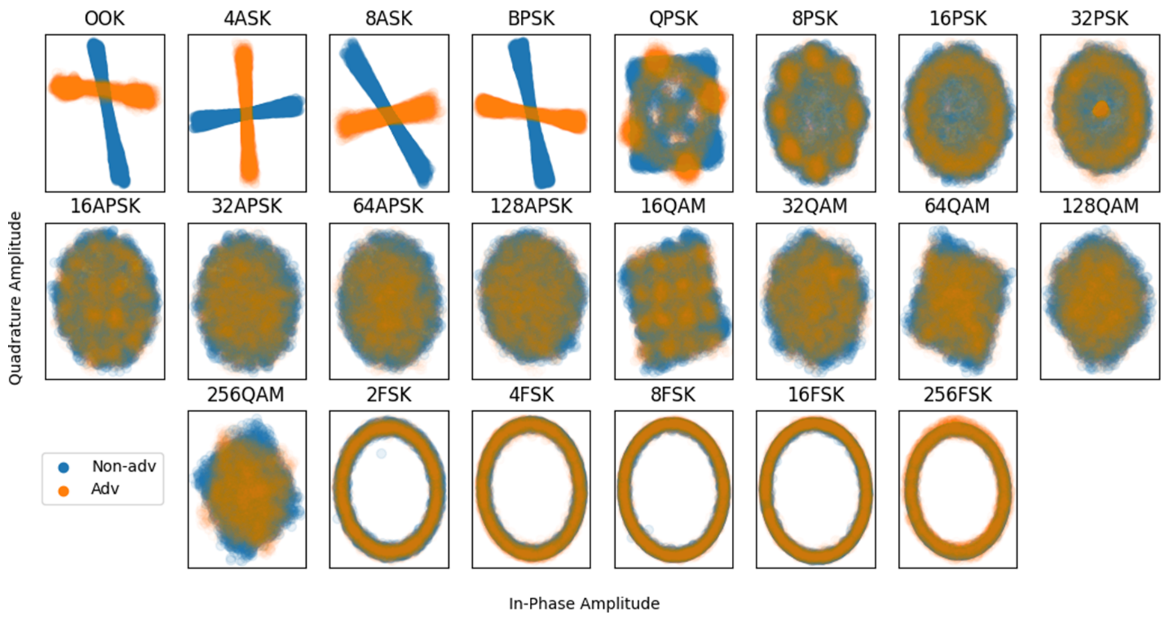


Figure 4.6: One *i.i.d.* captured perturbation and non-adversarial signal for each modulation class from the connected USRP N210 SDRs from our implementation of the RML2018.01A [3] dataset. These visualizations are over sampled by 8 SPS.

where the perturbation η is scaled by ϵ and added to the original signal x to form a perturbed sum of the signal and scaled perturbation, x^*. If the trained ML model's predictions are described as $f(x) = \hat{y}$, then the perturbations are crafted using the observed or estimated prediction loss function of the model given a signal:

$$\mathcal{L}(f(x^*), y) > \mathcal{L}(f(x), y), \tag{4.7}$$

with the expectation that increasing the loss will decrease the performance metrics of the deployed model (*i.e.,* F1-score, precision, recall, AUC, ROC, IoU, mAP). The reason for the scalar or mask ϵ was originally to craft computer vision perturbations that are imperceptible to the human eye, but generally used to minimize the perturbation according to the metric and scheme of choice. Broadly speaking, ϵ can be either constrained, $||\epsilon||_p = \delta$, to a fixed value or optimized, $\arg\min_\epsilon f(||\epsilon||_p)$, to achieve a minimum or maximum result for a specified constraint function. Typically, ϵ minimizing is defined by the vector norm metric for some chosen p, most commonly $p \in \{0, 1, 2, \infty\}$:

$$||\epsilon||_p = \left(\sum_{i=1}^{n} ||\eta_i||^p \right)^{1/p}, \tag{4.8}$$

which can be interpreted as designing perturbations to have a maximum number of changes, sum of absolute value of change, sum of squared change, and magnitude of largest change, respectively.

Finally, ϵ can take the form of a scalar that effects all elements of the perturbation signal equally, or it can take the form of a mask or signal that scales some elements more or less than others.

The adversarial threat model, or philosophy that guides the crafting of perturbations, can be described by a number of attributes including adversarial falsification, adversary knowledge, adversary specificity, and attack frequency:

- **Adversarial falsification**: The perturbation is crafted to achieve false positive or false negative predictions.
- **Adversary knowledge**: The perturbation is crafted with full knowledge of the trained ML model (white box) or only knowing a history of queries, or observed inputs and outputs of the ML model (black box).
- **Attack frequency**: Perturbations may be crafted after one query (black) or gradient computation (white), or improved over a number of repeated queries or gradient computations.
- **Adversary specificity**: The perturbation is crafted to achieve a targeted false prediction or any false non-targeted prediction.

Surveys of adversarial attacks and countermeasures are available on these topics from references [6, 60–62]. Since gradient-based white box attacks leverage their knowledge to take small steps in directions of steep or unstable loss, the theory behind most countermeasures is to increase the "flatness" of trained models loss function, or otherwise require perturbations to be larger in magnitude to achieve the same increase in loss. This serves the dual purpose of generalizing the model well to unexpected statistical behavior in the testing data partition. On the other hand, black box perturbation countermeasures leverage the transferability attribute, which holds as long as non-linear activation functions produce outputs that are distributed primarily around their linear region of operation. Consequently, countermeasures for these attacks carefully reduce or mask the linear behavior of classifiers.

In this work, we implemented several different attacks to confirm, deny, or establish best practices presented in leading ML data domains. Due to the variation of perturbations and non-adversarial signals, we define a generalized, adaptive scaling factor based on perturbation energy E_p for all attacks:

$$\epsilon = \frac{\sqrt{10^{\frac{E_s}{E_p}/10} \sum_{i=1}^{n} |x_i|^2}}{||\eta||_1}, \tag{4.9}$$

where x is the information signal and η is the perturbation signal, which achieves the desired signal (E_s) to perturbation energy ratio:

$$\frac{E_s}{E_p} = 10 \log_{10} \left(\frac{\sum_{i=1}^{n} |x_i|^2}{\sum_{i=1}^{n} |\eta_i|^2} \right). \tag{4.10}$$

The choice of $\frac{E_s}{E_p}$ represents the importance placed by the transmitter on each of the two objectives, being receiver BER and eavesdropper classification PPV:

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{4.11}$$

Given a finite power constraint, it is intuitive that amplifying all samples equally would result in the lowest BER. Yielding some of that power to strategically amplify some samples more than others grants the transmission a measure of obfuscation from fragile ML-based classifiers, at a cost to BER proportional to the power given up. If the transmitter has an objective eavesdropper PPV, the optimal choice for the $\frac{E_s}{E_p}$ ratio cannot be determined without a PPV feedback loop (see [79, 80]) from the eavesdropper to the transmitter, even in a white box scenario where all ML weights and classification rules are known. However, if the wireless channel is well known, as it is in many full duplex links, a BER objective could be used to choose a necessary signal energy, while using the remaining power constraint for perturbation energy.

The attacks used in this work include FGSM [59]:

$$x^* = x + \epsilon \text{sign}(\nabla_x J(x, y_{\text{true}})), \tag{4.12}$$

where y_{true} is the ground truth label of x, a relatively simple and efficient attack when compared to the others in this work which minimizes $p(y_{\text{true}}|x^*)$. Additionally, we use the

One-Step Least Likely (stepLL) attack [5]:

$$x^* = x - \epsilon \text{sign}(\nabla_x J(x, y_{\text{LL}})), \tag{4.13}$$

where y_{LL} is the least likely predicted class of x as determined by a classifier, which uses the least likely class of the signal according to the class scores of the model to maximize $p(y_{\text{LL}}|x^*)$. This attack is used for adversarial training [5] because FGSM [59] perturbations are substantially deterministic and correlated to the true label. Consequently, adversarial models trained with FGSM attacks classified adversarial data more accurately than non-adversarial test data, while those trained with $y_{]textLL}$ attacks does not. We visualize some stepLL perturbations in Figure 4.6. Finally, we use the Iterative Least Likely (iterLL) attack [5]:

$$\begin{aligned} x^*_{j+1} &= \text{Clip}_{x,-\epsilon,\epsilon}\{x^*_j - \alpha \text{sign}(\nabla_x J(x, y_{\text{LL}}))\}, \\ x^*_0 &= x,\ j = 0, ..., N \end{aligned} \tag{4.14}$$

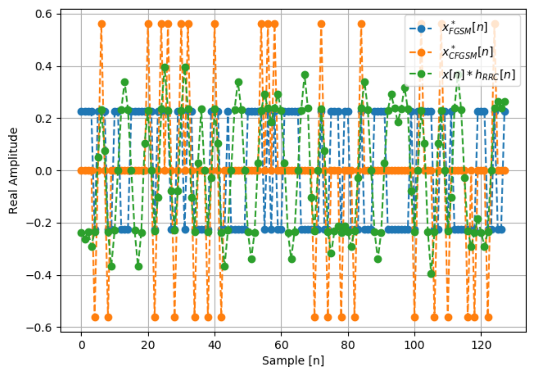
which achieves more powerful perturbations than its one step equivalent by recomputing the direction of the gradient multiple times. In our work, we sample the number of iterations $N \sim \mathcal{U}(2, 10)$ and compute the iteration step size as a ratio, $\alpha = \frac{2\epsilon}{N}$. We leave the investigation of Projected Gradient Descent (PGD) [5] to future work.

4.3 Signal Dependent Perturbation Design

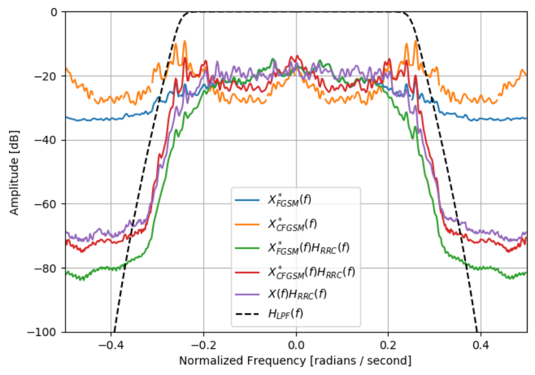
In this section, we present an approach employing two constraints on perturbation crafting that, to the best of the author's knowledge, have not been used in state-of-the-art wireless works. We utilize our modified RML2016.10A [143] dataset, FGSM [59] attacks, and the VT-CNN2 [2] modulation classifier.

4.3.1 Constructive Perturbations

We describe a novel constraint in the design of adversarial perturbations crafted to fool the model presented in Section 4.2.1, which can be applied to any state-of-the-art perturbation, such as those surveyed in Section 4.2.3. We assume the perturbation constraint does not allow for the transmitter to add perturbations that subtract, or add deconstructively, with the transmitted signal. Deconstructive perturbations may increase the number



(a) Time-domain In-Phase



(b) Frequency-domain

Figure 4.7: A visualization of a single pre-channel QPSK signal and CFGSM perturbations in the time-domain (a). CFGSM perturbations are signal-dependent and do not add deconstructively with the information being transmitted. We also display 78,125 QPSK signals and their constrained FGSM perturbations in a 512-bin FFT averaged by frequency bin (b), showcasing how much energy of the FGSM perturbations is attenuated because of their OOB frequencies.

Algorithm 10 Signal dependent perturbation crafting constraints.

```
 1: procedure GIVEN PERTURBATION $\eta$, TRANSMITTED SIGNAL $x$, AND PULSE SHAPING FILTER $h$
 2:     Enforce constructive addition constraint
 3:     for $i = 1, ..., n$ do
 4:         if sign($x_i$) $\neq$ sign($\eta_i$) or $i\%I \neq 0$ then
 5:             $\eta_i$=0
 6:         end if
 7:     end for
 8:     Enforce bandwidth equivalence constraint
 9:     $\eta^* = \eta * h$
10:     Scale perturbation to meet power requirement
11:     $\epsilon = \frac{\sqrt{10^{\frac{E_s}{E_p}/10} \sum_{i=1}^{n} |x_i|^2}}{||\eta^*||_1}$
12:     Add perturbation to signal
13:     $x^* = x + \epsilon\eta^*$
14: end procedure
```

of classification errors made by the adversarial eavesdropper, but at a significant cost to the BER of the communications chain. By setting the indices of the perturbation with opposite signs with respect to the transmitted signal to zero, the effectiveness of the perturbations is lessened, but the addition of the perturbation ceases to decrease communication effectiveness for phase and frequency-based modulation schemes. Note that this constraint is not beneficial for amplitude-based modulation schemes, where a large perturbation of signage matching that of the original signal will cause significant bit errors, even before transmission. Then, power allocated for the perturbation that is saved by this step should be allocated evenly across all remaining, non-zero perturbation indices. Using this constraint, the use of perturbations should not cause a single bit error before transmission. We call this constrained perturbation a Constructive-FGSM (CFGSM) perturbation.

4.3.2 Bandwidth Equivalent Perturbations

We present a second novel constraint for use in designing adversarial perturbations crafted to fool the model presented in Section 4.2.1, which may also be used for any state-of-the-art perturbation such as those described in Section 4.2.3. We assume the transmitter should implement a perturbation constraint that causes perturbations and the transmitted signal to have be Bandwidth Equivalent (BWE). The intuition of this constraint is that Out-Of-Band (OOB) energy is doubly wasteful. First, samples of the transmitted signal that are amplified by the perturbation are attenuated by the intended receiver's RFFE filters (*i.e.*, Band Pass Filters (BPF) and Low Pass Filters (LPF)) ubiquitous in all radio systems if the frequency of those samples is OOB. Consequently, the BER of the communications link is increased, which is not in the interest of the transmitter that designed these perturbations. Second, it is assumed that an eavesdropper would design an RFFE that isolates the signal from channel noise after the signal detection stage or by using channel meta data, which would also attenuate OOB perturbation energy. This would allow the eavesdropper to lower the perturbations power and increase their modulation classification PPV, which again is not in the interest of the transmitter designing perturbations. We note at this point that perturbations cannot be properly crafted before pulse shaping because data at that stage in the transmit chain do not properly represent the distribution of data being sent over the wireless channel. Both constraints are presented in Algorithm 10.

4.3.3 Constraint Analysis

To quantify the effectiveness of the constraints presented in Section 4.3.1 and Section 4.3.2, we perform a simulation to observe their impact on receiver BER and eavesdropper modulation classification PPV. To provide an objective benchmark for our BER results (Figure 4.8), we plot the non-coded theoretical limit for the QPSK modulation class, and consequently plot PPV results (Figure 4.9) for only the QPSK class as well. Each data

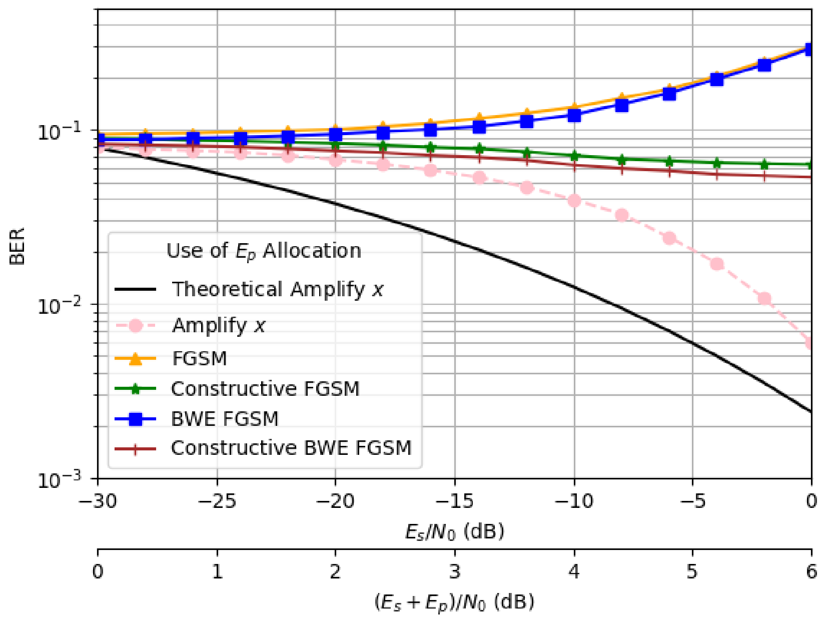


Figure 4.8: Non-coded BER for QPSK test signals for different uses and quantities of perturbation energy allocations. Noise and signal energy are fixed and equal at $E_s/N_0 = 0$ dB. When the perturbation energy allocation is re-allocated to the signal, and the signal is amplified uniformly, the SNR increases and BER decreases. The FGSM attack that is not signal dependent interferes with the transmission and increases BER proportional to the amount of energy allocated. The constructive FGSM attack only amplifies a subset of half $(1/I)$ of the indexes, such that BER decreases very slowly as the amount of energy allocated increases, but this is not the primary objective of increasing E_p, and this attack importantly does not increase BER. This crucially allows, if the transmitter's operator has chosen a E_s to satisfy a maximum BER requirement, the guarantee that the use of perturbations, for any E_p, will not increase BER and compromise that requirement. Many such Quality of Service (QoS) guarantees are mandatory in wireless protocols, and can be difficult to enforce in ML applications. BWE attacks are slightly lower BER than their non-BWE equivalent attacks because their energy is not filtered out by the LPF.

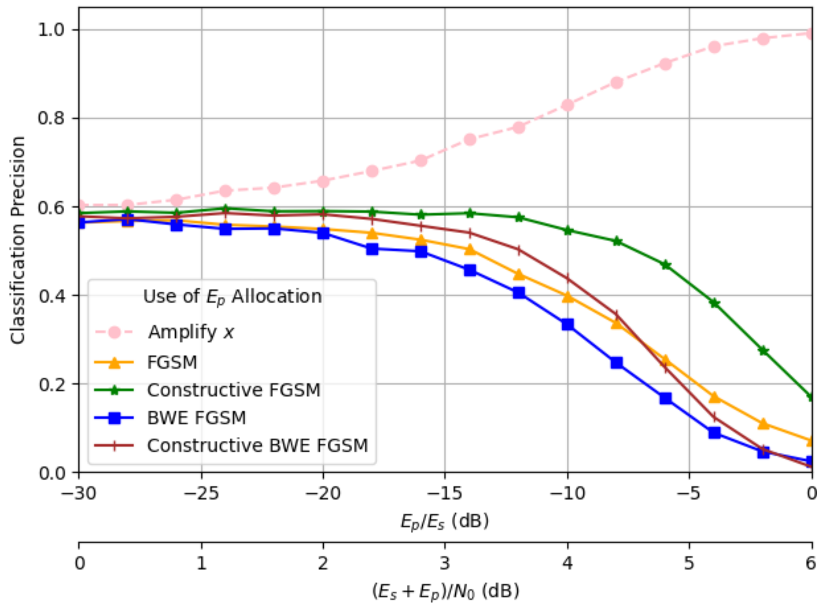


Figure 4.9: Modulation classification PPV for QPSK test signals for different uses and quantities of perturbation energy allocations. Noise and signal energy are fixed and equal at $E_s/N_0 = 0$ dB. When the perturbation energy allocation is re-allocated to the signal, and the signal is amplified uniformly, the SNR increases and the signal is easier for the eavesdropper to classify. The BWE FGSM attack is the most effective because no energy is filtered out, and the energy constraint is used optimally. The constructive FGSM attacks are less effective because a lot of energy is used on a subset of the samples rather than a little energy on the whole set, and the effectiveness of perturbations is proportional to the sum of amplitudes, not power.

point is computed by averaging 10,000 signals or bits for PPV or BER results, respectively. For BER computations, we employ the same RRC pulse shaping filter used by the simulated transmitter as a matched filter and assume the presence of AWGN only. Additionally we visualize the effect of our two constraints on a single QPSK signal in Figure 4.7.

Our results show that CFGSM perturbations contain a greater number of high-frequency components compared to FGSM perturbations due to similar time domain characteristics to impulse trains than square waves. Consequently, CFGSM perturbations are even more attenuated by a LPF if the BWE constraint is not enforced, which can be observed by a larger gap in the classification PPV between the BWE CFGSM and CFGSM than BWE FGSM and FGSM. Our classification results show the BWE CFGSM perturbations are only slightly less effective at fooling models other than BWE FGSM perturbations depending on SNR and perturbation strength, while guaranteeing that BER is only decreased with perturbations strength, never increased. This allows for the transmitter to be designed with guaranteed BER bounds while obfuscating its signals from eavesdroppers, a design consideration not seen in any other adversarial modulation classification work. In duplex radio systems where the wireless channel can be accurately estimated and it is determined that the transmitter has sufficient power to meet system BER requirements, our results show that a transmitter implementing these perturbation constraints can obfuscate its frequency or phase-based modulation scheme from eavesdroppers with zero cost to communication effectiveness.

4.4 Adversarial Training

In this section, we investigate the inverse of the problem of the perturbation crafting previously discussed in Section 4.3. Perturbation countermeasures are studied here by implementing the adversarial training scheme outlined in [5] using our RML2018.01A [3] inspired dataset and the VGG10 [4] inspired modulation classifier from Section 4.2.2, and all attacks presented in Section 4.2.3. We do so using perturbations crafted after first training a non-adversarial model, as in [209], such that we transfer the knowledge of the end results of training. The idea behind adversarial training is to train the model using mini-batches with both perturbations and non-adversarial signals:

$$\text{Loss} = \frac{1}{(m-k)+\lambda k}\left(\sum_{i=1}^{m-k}\mathcal{L}(x, y_{\text{true}}) + \lambda\sum_{i=1}^{k}\mathcal{L}(x^*, y_{\text{true}})\right), \tag{4.15}$$

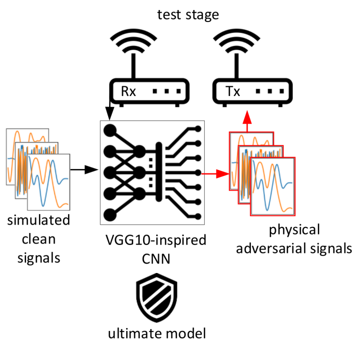
where m is the mini-batch size, k is the number of adversarial examples per mini-batch, $\mathcal{L}(\cdot)$ is categorical cross entropy loss, and λ is the weighting of learning step size for adversarial versus non-adversarial training examples. In this work, we use $m = 256$, $k = 38$, and $\lambda = 1$ such that we achieve what is an effectively equivalent training scheme as seen in [5], who choose $m = 32$, $k = 16$, and $\lambda = 0.3$. We quantify the similarity of these parameter choices as $\frac{m\lambda}{k} = 0.15$. As in [5], we randomly vary perturbation strength such that the adversarial trained model generalizes well to test-stage perturbations of different strengths. We accomplish this variation using a truncated Gaussian distribution as:

$$\begin{aligned} \epsilon^* &= \text{Clip}_{\epsilon,0,1}\{\epsilon + \delta\}, \\ \delta &\sim \mathcal{N}(0, 1/2), \end{aligned} \tag{4.16}$$

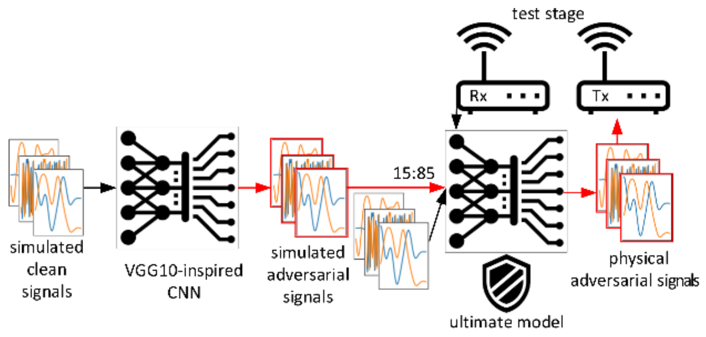
and refer to the value of E_s/E_p for this scheme as "sweeping". We perform the costly, relative to computer vision, training schemes presented in this section using a Intel Xeon Gold 6248 CPU node with 20 cores and 192 GB of RAM, and one NVIDIA Volta V100 GPU node with 32 GB of RAM.

4.4.1 Evaluation of Non-Adversarial Model

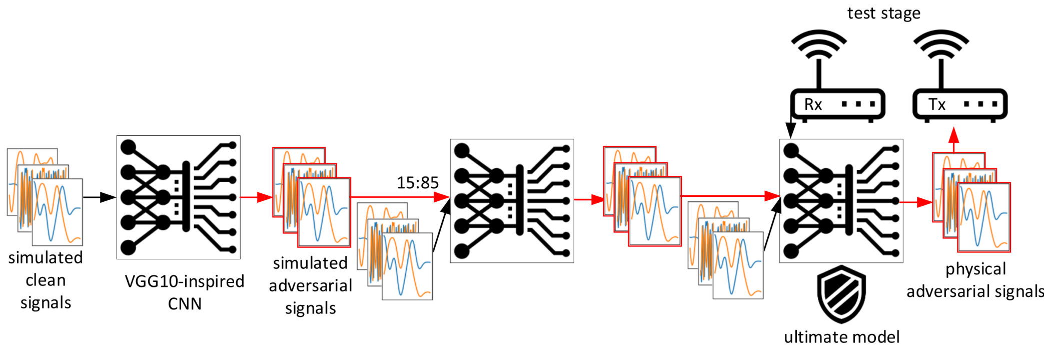
In evaluating the non-adversarial training scheme, we made a number of discoveries. We found that Frequency Shift Keying (FSK) modulation classes are the most difficult to fool, with only three false positives across all modulation orders of FSK in an FGSM attack. This is due to the frequency shifts between each symbol being so large, as well as due to the uniqueness of the FSK constellations with respect to amplitude and phase shifting schemes. The crafting of frequency-domain perturbation is the subject of ongoing research and will be the focus of a subsequent publication. When stronger attacks, deeper models, or larger perturbation energy are used, more FSK signals are fooled. We found that FGSM attacks perform better than stepLL attacks, because they lower the class score of the true class rather than increased the score of the least likely class. We also observed that the iterLL attack is the most effective attack because it most accurately ascends the gradient due to



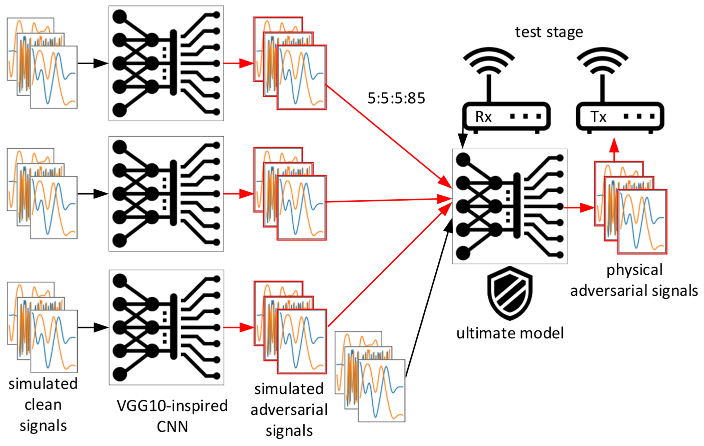
(a) Non-adversarial training



(b) Adversarial training

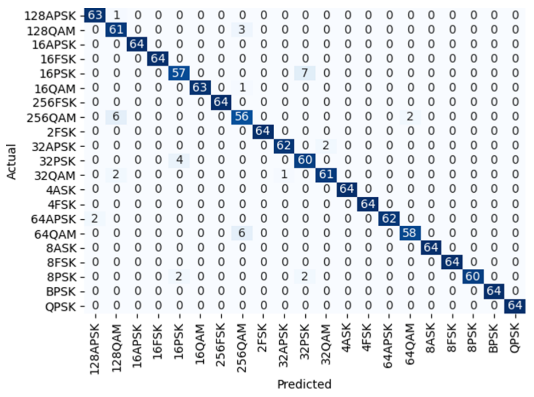


(c) Cascade adversarial training

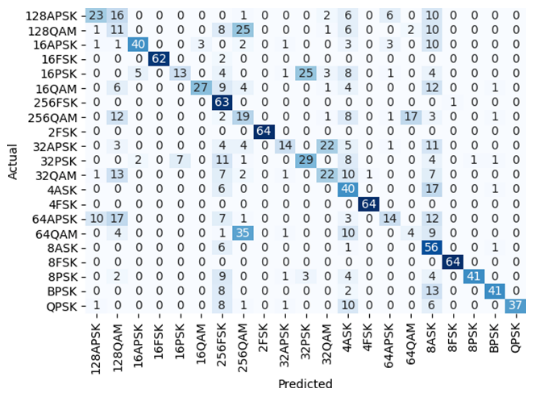


(d) Ensemble adversarial training

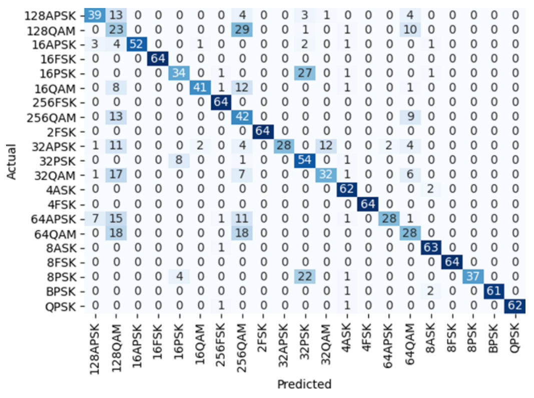
Figure 4.10: Our offline non-adversarial (a), adversarial (b), cascade (c), and ensemble (d) training schemes mostly follow those outlined in [5], although we decouple training by only generating perturbations from already trained models, as in [6]. Additionally, unlike any other work, we evaluate our model using perturbations crafted from gradients computed from the ultimate model, and do so using online, physical signal captures. Our reasoning is that if our system is vulnerable to an attack once, it can be attacked again, and to assume that the attack is done without knowledge of our countermeasure is overly optimistic. Each model and dataset is *i.i.d.*, and the training of the ultimate model is always done with the same number of weight updates as outlined in Section 4.2.2. For instance, if we produce a parallel set of adversarial training data using three models, we would train the ultimate model using three sets of 1.4/3 million signals for 20 epochs each.



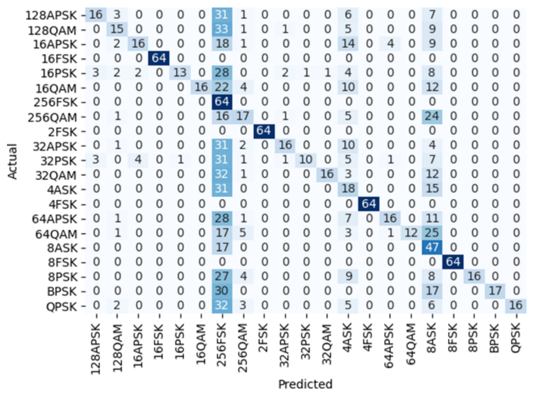
(a) Physical test



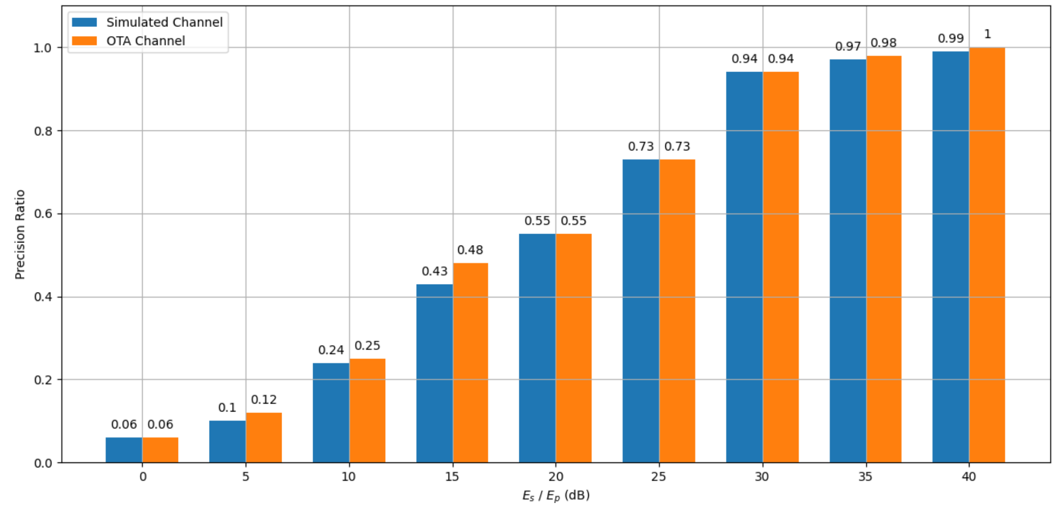
(b) Physical sweeping E_s/E_p FGSM test



(c) Physical sweeping E_s/E_p stepLL test



(d) Physical sweeping E_s/E_p iterLL test



(e) Physical fixed E_s/E_p stepLL

Figure 4.11: A class-by-class analysis of the effectiveness of each attack and strength of attack on the non-adversarial trained VGG10 model. Most false positives belong to the same one or two classes. IterLL attacks are the strongest, followed by FGSM, and stepLL. FSK classes are the most difficult to fool due to large frequency shifts between each symbol. Perturbations sent over a physical channel are slightly less effective than perturbations transmitted over a simulated wireless channel.

taking multiple, smaller steps. Additionally, most test sets showed that, when attacked, they attempt to fool all classifications to be one of a few classes. For instance, 57% of false positives caused by iterLL attacks on the non-adversarial trained model belonged to the 256FSK class, 23% to the 8 Amplitude Shift Keying (ASK) class, and 20% to all other classes. Finally, we observed that increasing perturbation strength decreases modulation classification PPV, which is to be expected. Specifically, $E_s/E_p = 0$ dB stepLL attacks are required to approach a PPV equal to that of a zero rule classifier, and that $E_s/E_p > 35$ dB stepLL attacks had no effect on physical test PPV. On average, perturbations sent over a physical channel are slightly less effective, relative to non-adversarial PPV, than perturbations transmitted over a simulated wireless channel.

4.4.2 Evaluation of Cascade and Parallel Models

Parallel [6] and cascade [209] adversarial training are parallel and sequential, respectively, methods of decoupling the generation of adversarial training examples from the model being trained. The theory behind parallel decoupling is that perturbations are transferable between models and that parallel adversarial training schemes will achieve a better approximation of the underlying distribution of perturbations than adversarial training using perturbations crafted from a single pre-trained model, providing greater protection against black box attacks or new white box attacks generated by the fully trained model. The knowledge transferred by a parallel set of perturbations is statistically diverse and high variance, competing with non-adversarial training data for learning capacity in small models [6], such that under fitting occurs if the model size is not increased appropriately.

The theory behind cascade adversarial schemes is that each iteration of training transfers additional information about how perturbations are crafted from already trained models to the ultimate model. We hypothesize there is some number of cascade training iterations and parallel set size that is optimal for a given scenario, and seek to identify the performance trends of these schemes via physical experimentation on models trained offline.

The number of training samples and number of training epochs for the ultimate model were held constant across all of these schemes (Figure 4.10) such that the resulting PPV of each scheme will be the result of the knowledge transferred by training perturbations and not the duration of training or quantity of data.

In Table 4.1, adversarial training maintains about 26% of its protection against current

Table 4.1: Effect of various adversarial training schemes on the modulation classification PPV of different partitions of data. step$_{i-1}$ perturbations refers to testing models using perturbations from the same distribution as training set perturbations, where step$_i$ perturbations refers to testing model using perturbations crafted after adversarial training. The adversarial training maintains $\sim 26\%$ of its protection against current step physical attacks compared to physical attacks crafted during training. Furthermore, as in [6], the model trained by the parallel training scheme is more accurate when evaluated on adversarial data at the cost of non-adversarial accuracy. In [6], this gain is seen only for black box attacks, not white box attacks. Our current step white box attacks are analogous to black box attacks from the perspective of adversarial training because test-phase perturbations are crafted from a different set of weights than that from which training perturbations are crafted. Finally, we observed that the cascade adversarial training scheme follows the same trend as the parallel scheme but with greater magnitude.

	Training Scheme			
PPV	Clean	Adversarial	Parallel	Cascade
Training	99.85	99.85	**99.90**	**99.90**
Clean Test	**99.22**	99.15	98.22	95.74
Clean physical Test	96.35	**96.80**	96.21	92.71
stepLL$_{i-1}$ Test	-	76.07	-	-
stepLL$_{i-1}$ physical Test	-	75.22	-	-
stepLL$_i$ Test	70.45	73.58	70.53	**77.63**
stepLL$_i$ physical Test	68.67	70.39	71.88	**81.53**

Table 4.2: An investigation of "label leaking" [5] occurring when using FGSM adversarial training schemes, justifying the use of the stepLL attack in training over the use of the FGSM attack. While we do not see evidence of label leaking for this dataset, we find that stepLL training yielded higher protection against iterative and FGSM attacks than FGSM training, which are the most dangerous attacks.

	Training Scheme		
PPV	Clean	FGSM	stepLL
Training	**99.85**	99.73	**99.85**
Clean physical Test	96.35	96.14	**96.80**
stepLL physical Test	68.67	**76.70**	70.39
iterLL physical Test	44.42	42.12	**44.49**
FGSM physical Test	59.23	60.94	**64.96**

step attacks compared to attacks used in training. Additionally, the ultimate models trained using the parallel training scheme perform worse in all scenarios except for attacks crafted using a model other than that used in adversarial training, or that their robustness is transferable at the cost of regularization. Finally, these models trained using the cascade scheme follow the same trends, but to a greater magnitude than parallel training schemes.

4.4.3 Label Leaking

Label leaking is described in [5] as when adversarial training with the use of ground truth labels in attacks such as FGSM [59] results in a trained model that tests better on adversarial data than non-adversarial data for an individual signal, with and without its added perturbation. Specifically, a label has leaked for a test signal if x^* is classified correctly but x is not. Label leaking is not possible in our experiments since we disjoint crafting by discarding x when we craft x^*, as in [6], which is one of the reasons we have used such a technique. However, we can still interpret the modulation classification PPV obtained on *i.i.d.* populations of adversarial and non-adversarial test signals to determine if models have been over trained with perturbations. This is because the intuition behind label leaking is that ground truth based attacks perform a deterministic transform on data that is highly correlated to the ground truth. As a consequence, if we define the PPV ratio of a model as:

$$\text{PPV ratio} = \frac{\text{PPV}(x^*)}{\text{PPV}(x)}, \tag{4.17}$$

then test sets with leaked labels will achieve a PPV ratio > 1.

To validate the presence and severity of label leaking in wireless experiments and contrast those findings with those in relatively high dimension, zero noise computer vision works [5], we implement the adversarial training methodology presented by Figure 4.10 with FGSM attacks. In Table 4.2, we do not see evidence of label leaking, but we do see evidence that stepLL training resulted in more robust models against iterative and FGSM attacks than FGSM training.

4.4.4 Evaluation of Models Trained with Iterative Attacks

In [5], the authors found that adversarial training with iterative attacks did not train models robust to iterative attacks. They hypothesized that they did not have the computational resources to train their Inception v3 [210] model on ImageNet [211] data with a large

Table 4.3: IterLL attacks are significantly more effective than stepLL attacks. StepLL training offer almost no defense against iterLL attacks. We are able to achieve iterLL trained models with a small level of defense against iterLL attacks, and higher defense against stepLL and FGSM attacks with no significant loss to non-adversarial performance.

	Training Scheme		
PPV	Clean	stepLL	iterLL
Training	99.85	99.85	**99.89**
Clean physical Test	96.35	**96.80**	96.65
FGSM physical Test	59.23	64.96	**65.06**
stepLL physical Test	68.67	70.39	**76.21**
iterLL physical Test	44.42	44.49	**45.88**

enough learning capacity to learn the complex distribution of iterative attacks. In [212], the authors reduced the computational cost of iterative Projected Gradient Descent (PGD) [5] attack training by generating Canadian Institute for Advanced Research (CIFAR)-10 and CIFAR-100 [213] adversarial perturbations during training by using the gradient computed for SGD, rather than computing it again. They achieve a moderate level of protection at a very low computational cost.

In this work, we performed iterLL adversarial training using a RML2018.01A inspired dataset to see what degree of protection we may obtain from iterLL and other attacks. We do so without the dual-use of the gradient as in [212] because crafting perturbations during training rather than after does not disjoint crafting as in [6]. Additionally, we hypothesized that our relatively low dimension data (*i.e.*, 8192 features/example for the RML2018.01A inspired dataset versus 544509 average features/example for ImageNet [211]), relatively smaller model (*i.e.*, 18.2×10^6 parameters in our VGG10 inspired model versus 24×10^6 parameters in Inception v3), and several years of computational resource advancements (*i.e.*, Volta 100 versus Tesla K80 Graphics Processing Units (GPUs)) will render the dual-use unnecessary.

In Table 4.3, we observed that iterLL attacks are 206% more effective than stepLL attacks for our dataset, model, and attack parameters. Additionally, stepLL training offered no significant defense against iterLL attacks, prompting the need for an iterLL training scheme. The results of our iterLL training are very positive, showing an increased defense against all attacks without losing non-adversarial performance. Most notably, it is the only

Table 4.4: Effect of model capacity on adversarial training, evaluated using physical test data. We find that adversarial training prevents overfitting from occurring when training our VGG10 model scaled by $\rho = 4$. We additionally find that stepLL perturbations crafted after adversarial training are more effective against deeper models, indicating a model capacity trade-off between non-adversarial and adversarial test classification PPV. Models that are too shallow additionally make lower confidence classifications than deep models, such that they are easier to fool. "Clean" is short hand for non-adversarial data.

	Training Scheme							
	$\rho = 0.5$		$\rho = 1$		$\rho = 2$		$\rho = 4$	
PPV	Clean	stepLL	Clean	stepLL	Clean	stepLL	Clean	stepLL
Training	99.71	99.71	99.85	99.85	99.89	99.88	99.90	99.89
Clean physical Test	95.46	96.06	96.35	96.80	97.47	97.49	97.40	**97.54**
stepLL physical Test	-	58.04	-	**70.39**	-	58.89	-	37.80

training scheme that achieved any level of protection against iterative attacks.

4.4.5 Model Capacity

In other works [5], the authors were unable to find a model deep enough to over fit in the presence of adversarial training using the stepLL method. In their work, and in ours, model width is scaled by increasing the number of convolutional filters in every convolutional layer by a factor ρ. While our model utilizes batch normalization to some effect, we do not find dropout to improve test-stage PPV.

In this work, we investigated the effectiveness of stepLL adversarial training as a regularizer in wireless experiments. We hypothesized that the relatively low dimension data, relatively small models, and several years of computational resource improvements will make it more feasible to scale to extreme ρ values.

In Table 4.4, we were able to scale $\rho \in [0.5, 4]$ before running out of memory. We found that at $\rho = 4$ the non-adversarial trained VGG10 began to over fit to training data because it had a lower physical test data classification PPV than the $\rho = 2$ non-adversarial trained model. However, with adversarial training, the model is regularized and physical test data classificaiton PPV continues to increase with ρ. Additionally, deeper models were more vulnerable to adversarial perturbations, which can be explained by [59], where it was shown that FGSM perturbations increased the magnitude of activations by $\epsilon \times n \times m$, where m is the average value of weights in a layer and n is the number of weights in a layer. We hypothesized that by increasing ρ, we are increasing n, such that perturbations, all else equal, will have a greater impact on classification PPV. We tested this hypothesis by computing the ratio of mean class score magnitudes between clean physical and stepLL physical test data for adversarial trained models with $\rho = 1$ and $\rho = 4$. We obtained resulting ratios of 0.39 and 0.33, failing to reject our hypothesis that perturbations increase the magnitude of class scores, on average, proportional to the number of weights in each layer of a CNN.

We observed that the shallow $\rho = 0.5$ model is also more vulnerable to attacks. One potential explanation for this is it made lower confidence classifications that are easier to fool. To test this, we computed, for physical test sets, the average difference in class scores between the largest and second largest class scores for $\rho = 0.5$ and $\rho = 1$ adversarial trained models. We found that they had an average top and second top class score difference of 71.97 and 91.11, respectively, failing to reject our hypothesis that the shallow model makes less confident classifications.

Consequently, we determined that model width must be carefully managed in adversarial training schemes to ensure that the model is deep enough to learn the non-adversarial and adversarial datasets, deep enough to make high-confidence classifications that require large changes to class scores to cause false positives, and shallow enough as not to become vulnerable to the compounding attribute of attacks. Additionally, we concluded that this trade-off is relatively advantageous for adversarial training of wireless spectrum sensing, signal classification, and modulation classification when compared to computer vision tasks, which tend to require much deeper models to learn relatively high dimension data distributions that have large state spaces.

4.5 Chapter Summary

We performed in Section 4.4, and outlined the details in Section 4.2.2, the first physical adversarial ML-based modulation class eavesdropping experiment. Given the significant research interest in modulation classification [3, 39, 199, 214–217] and adversarial wireless ML [76–82, 85, 86] this novel experiment is a significant real-world validation for many theoretical works that have experimented largely with simulated channel models and signals. In Section 4.3, we addressed two theoretical shortcomings we have seen in the design of perturbations in other works having to do with the bandwidth and sign constraints of signals.

Our simulations and experiments presented in chapter 4 yielded a number of findings and confirmations to the state-of-the-art, including:

- Training a CNN offline using channel models can achieve high accuracy modulation classification performance on physical signals.
- A constructive constraint can be added to adversarial perturbation crafting to enforce adversarial amplification, in which the transmitter's goals of maximizing communications capabilities do not conflict with those of evasion.
- A BWE constraint can be added to adversarial amplification, in which both evasion and communication metrics can be improved at the cost of passing the perturbation through a pulse shaping filter.
- Physical Adversarial amplification of a transmitter can reduce the classification accuracy of an eavesdropping receiver's trained ML classifier to as low as guessing despite phase, frequency, and amplitude noise sources from both the RFFE and the channel.
- Adversarial training of the eavesdropping receiver using simulated channel models can achieve some level of defense against adversarial amplification, where the best results are achieved when adversarial training is done using perturbations crafted from a fully trained, *i.i.d.* non-adversarial model.
- Label leaking does not appear to occur in low-dimensional data domains.
- Parallel and cascade adversarial training schemes over-emphasize adversarial examples during training, reducing testing accuracy for non-adversarial data. This defeats the primary objective of adversarial training, which is to increase robustness without sacrificing non-adversarial performance
- A measure of protection against iterative attacks is possible with iterLL training.
- The model width of the eavesdropping receiver must be carefully managed to achieve an "elbow" point in the trade-off between non-adversarial and adversarial test performance. Specifically, we found the CNN must be wide enough to make correct and high confidence classifications, wide enough to have the learning capacity for both adversarial and non-adversarial PDFs, and thin enough as not to compound the increase to the loss function caused by perturbations.

Chapter 5

Conclusion & Future Work

In this dissertation, we began in chapter 1 by motivating our work, presenting the state-of-the-art, technical challenges, and our contributions. The basics of the UL and SL algorithms used in the research contribution were discussed in sections in chapter 2. Specifically, CNNs and related SL algorithms were presented by Section 2.1, guidelines for implementing SL by Section 2.2, DPGMMs and related UL algorithms by Section 2.3, and adversarial perturbations by Section 2.4. Research contributions of this dissertation were presented in chapter 3 and chapter 4, which secure localization networks from spoofers via clustering and obfuscate communication links from SL-based eavesdroppers via perturbations, respectively.

5.0.1 Research Achievements

The proposed opportunistic, self-organizing, stationary transmitter, jointly localizing and classifying methodology demonstrated in chapter 3 that:

- Locating transmitters in low meta data scenarios with high accuracy is possible.
- The $N = 229$ spatial test set was able to recognize every statistically different RSS population as verified by a Kruskal-Wallis (kW) test.
- That test classified the source radio of transmissions with a 76.4% indicator variable PPV (93.7% when accounting for missed cluster detections).
- The detected transmitters were localized with an average of 1.71 meter variance and 1.19 meter bias.

- We confirmed the Gaussian assumption of our clustering algorithm by failing to deny that the PointNet CNN's location predictions were Gaussian distributed. We accomplished this verification via an AD test with critical statistics $A^{*2} = 1.0$ and observed statistics $A^2 = 0.39, 0.66$ for x-axis and y-axis samples, respectively.

Our simulations and experiments presented in chapter 4 yielded a number of findings and confirmations to the state-of-the-art, including:

- Training a CNN offline using channel models can achieve high accuracy modulation classification performance on physical signals.
- A constructive constraint can be added to adversarial perturbation crafting to enforce adversarial amplification, in which the transmitter's goals of maximizing communications capabilities do not conflict with those of evasion.
- A BWE constraint can be added to adversarial amplification, in which both evasion and communication metrics can be improved at the cost of passing the perturbation through a pulse shaping filter.
- Physical Adversarial amplification of a transmitter can reduce the classification accuracy of an eavesdropping receiver's trained ML classifier to as low as guessing despite phase, frequency, and amplitude noise sources from both the RFFE and the channel.
- Adversarial training of the eavesdropping receiver using simulated channel models can achieve some level of defense against adversarial amplification, where the best results are achieved when adversarial training is done using perturbations crafted from a fully trained, *i.i.d.* non-adversarial model.
- Label leaking does not appear to occur in low-dimensional data domains.
- Parallel and cascade adversarial training schemes over-emphasize adversarial examples during training, reducing testing accuracy for non-adversarial data. This defeats the primary objective of adversarial training, which is to increase robustness without sacrificing non-adversarial performance
- A measure of protection against iterative attacks is possible with iterLL training.

- The model width of the eavesdropping receiver must be carefully managed to achieve an "elbow" point in the trade-off between non-adversarial and adversarial test performance. Specifically, we found the CNN must be wide enough to make correct and high confidence classifications, wide enough to have the learning capacity for both adversarial and non-adversarial PDFs, and thin enough as not to compound the increase to the loss function caused by perturbations.

5.0.2 Open Questions

Based on the outcomes of the research presented in chapter 3, several open challenges and opportunities for further research remain:

- The SL-based multilateration could be improved: The current state-of-the-art [155–160] uses a de-noising auto encoder with unlabelled data and a CNN multilateration algorithm. We attempted multiple data generation schemes, but determined we did not have enough data to properly train a semi-supervised algorithm, and did not gather unlabelled data. The PointNet CNN architecture was what worked best for us, but our architecture design and hyper-parameter search was not exhaustive.

- A clustering algorithm: While a DPGMM seemed like the natural choice given the data set, there exist a number of clustering algorithms that do not require the number of clusters to be specified, including affinity propagation [218], mean shift [219], Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [220], Ordering Points To Identify Cluster Structure (OPTICS) [221], and Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [222].

- Measure performance with different metrics: While we use bias, variance, MSE, and precision metrics, there may exist other metrics that serve as a better comparison of clustering in the context of beacon localization and verification. Potential metrics include precision, Receiver Operating Characteristic (ROC), and Area Under the Curve (AUC).

- Capture a new data set: Costs constrained our number of receivers used, quality of hardware, and duration of our experiment. This resulted in a data set that is challenging at both the regression and clustering stages. Popular benchmarking data sets are by design very large relative to the data's variance such that results produced from the data are low variance and reproducible. Of particular interest is the collection of a TDoA set of features, which are more robust to adversarial attacks than RSS-based features [57].

We acknowledge the complexity and breadth of the research area of demodulation evasion by offering the following open challenges to the wireless ML security community:

- It is possible that our CNN simply do not have the learning capacity to learn all three PDFs from the parallel adversarial training scheme, or the complex perturbations from the cascade adversarial training scheme. We leave it to future work to determine the right balance of model width with these training schemes and those factors already researched in this chapter on the topic of model capacity.
- The task of maximizing the protection offered by the various adversarial training schemes presented in this chapter is unique to each dataset and SL model. We leave it to the reader to tune all hyper-parameters involved to improve the regularizing and protective effects of adversarial training for their scenario. We also note here that we did not pursue a determined tuning effort in this work to maximize regularization and robustness, but merely to determine trends.
- While we did not investigate the use of distillation, masking, semi-supervised detection, and network verification, we encourage the continued investigation of these methods as viable counter-measures to adversarial perturbations.
- An analysis of countering black box attacks, which are lower assumption, lower performing, and easier to implement in real systems. Such a study still has yet to be studied in an physical, experimental setting.
- The use of ensemble training, or multiple "ultimate models" that vote in a weighted scheme to determine classifications, will improve robustness and regularization of classifiers further, despite the full visibility of such a scheme to the adversary. We leave this task as an open challenge, specifically to determine the trade-offs between bagging, boosting, and stacking ensemble training schemes [179] in this scenario.
- The use of activation embedding regularization [209] in the low-dimension data space of wireless signals.

Bibliography

[1] S. M. Kay, “The multifamily likelihood ratio test for multiple signal model detection,” *IEEE Signal Processing Letters*, vol. 12, no. 5, pp. 369–371, May 2005.

[2] N. E. West and T. O’Shea, “Deep architectures for modulation recognition,” in *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, March 2017, pp. 1–6.

[3] T. J. O’Shea, T. Roy, and T. C. Clancy, “Over-the-air deep learning based radio signal classification,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[5] A. Kurakin, I. Goodfellow, S. Bengio *et al.*, “Adversarial examples in the physical world,” 2016.

[6] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, “Ensemble adversarial training: Attacks and defenses,” *arXiv preprint arXiv:1705.07204*, 2017.

[7] N. Strom, “Scalable distributed dnn training using commodity gpu cloud computing,” in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[8] Y. Roh, G. Heo, and S. E. Whang, “A survey on data collection for machine learning: a big data-ai integration perspective,” *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[9] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in neural information processing systems*, vol. 24, pp. 693–701, 2011.

[10] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[11] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A survey on machine-learning techniques in cognitive radios," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1136–1159, Third 2013.

[12] M. A. Alsheikh, S. Lin, D. Niyato, and H. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 4, pp. 1996–2018, Fourthquarter 2014.

[13] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self-organizing cellular networks," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2392–2431, Fourthquarter 2017.

[14] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4, pp. 2923–2960, Fourthquarter 2018.

[15] J. Wang, C. Jiang, H. Zhang, Y. Ren, K. C. Chen, and L. Hanzo, "Thirty years of machine learning: The road to pareto-optimal wireless networks," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 1472–1514, 2020.

[16] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3039–3071, 2019.

[17] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1153–1176, Secondquarter 2016.

[18] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's in-

telligent network traffic control systems," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2432–2455, Fourthquarter 2017.

[19] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4, pp. 2595–2621, Fourthquarter 2018.

[20] R. Kumar Dwivedi, S. Pandey, and R. Kumar, "A study on machine learning approaches for outlier detection in wireless sensor network," in *2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 2018, pp. 189–192.

[21] M. Kulin, C. Fortuna, E. De Poorter, D. Deschrijver, and I. Moerman, "Data-driven design of intelligent wireless networks: An overview and tutorial," *Sensors*, vol. 16, 06 2016.

[22] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *CoRR*, vol. abs/1803.04311, 2018. [Online]. Available: http://arxiv.org/abs/1803.04311

[23] Y. Liu, S. Bi, Z. Shi, and L. Hanzo, "When machine learning meets big data: A wireless communication perspective," *IEEE Vehicular Technology Magazine*, vol. 15, no. 1, pp. 63–72, 2020.

[24] T. J. O'Shea and J. Hoydis, "An introduction to machine learning communications systems," *CoRR*, vol. abs/1702.00832, 2017. [Online]. Available: http://arxiv.org/abs/1702.00832

[25] D. Crevier, *AI: The Tumultuous History of the Search for Artificial Intelligence.* USA: Basic Books, Inc., 1993.

[26] N. Benvenuto, M. Marchesi, F. Piazza, and A. Uncini, "Non linear satellite radio links equalized using blind neural networks," in *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, 1991, pp. 1521–1524 vol.3.

[27] S. Benedetto, E. Biglieri, and R. Daffara, "Modeling and performance evaluation of nonlinear satellite links-a volterra series approach," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-15, no. 4, pp. 494–507, 1979.

[28] S. A. Billings, S. Chen, and M. J. Korenberg, "Identification of mimo non-linear systems using a forward-regression orthogonal estimator," 1989, address: London. [Online]. Available: https://eprints.soton.ac.uk/251146/

[29] D. Tank and J. Hopfield, "Simple 'neural' optimization networks: An a/d converter, signal decision circuit, and a linear programming circuit," *IEEE Transactions on Circuits and Systems*, vol. 33, no. 5, pp. 533–541, 1986.

[30] J. Bruck and M. Blaum, "Neural networks, error-correcting codes, and polynomials over the binary n-cube," *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 976–987, 1989.

[31] S. Chen, G. Gibson, C. Cowan, and P. Grant, "Adaptive equalization of finite non-linear channels using multilayer perceptrons," *Signal Processing*, vol. 20, no. 2, pp. 107 – 119, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016516849090122F

[32] D. Kunz, "Channel assignment for cellular radio using neural networks," *IEEE Transactions on Vehicular Technology*, vol. 40, no. 1, pp. 188–193, 1991.

[33] B. Aazhang, B. . Paris, and G. C. Orsak, "Neural networks for multiuser detection in code-division multiple-access communications," *IEEE Transactions on Communications*, vol. 40, no. 7, pp. 1212–1222, 1992.

[34] W. . Fang, B. J. Sheu, O. T. . Chen, and J. Choi, "A vlsi neural processor for image data compression using self-organization networks," *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 506–518, 1992.

[35] M. Ibnkahla, N. J. Bershad, J. Sombrin, and F. Castanie, "Neural network modeling and identification of nonlinear channels with memory: algorithms, applications, and analytic models," *IEEE Transactions on Signal Processing*, vol. 46, no. 5, pp. 1208–1220, 1998.

[36] K. Iba, “Reactive power optimization by genetic algorithm,” *IEEE Transactions on Power Systems*, vol. 9, no. 2, pp. 685–692, 1994.

[37] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[38] H. L. Southall, J. A. Simmers, and T. H. O’Donnell, “Direction finding in phased arrays with a neural network beamformer,” *IEEE Transactions on Antennas and Propagation*, vol. 43, no. 12, pp. 1369–1374, 1995.

[39] A. K. Nandi and E. E. Azzouz, “Algorithms for automatic modulation recognition of communication signals,” *IEEE Transactions on Communications*, vol. 46, no. 4, pp. 431–436, 1998.

[40] F. C. Hoppensteadt and E. M. Izhikevich, “Pattern recognition via synchronization in phase-locked loop neural networks,” *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 734–738, 2000.

[41] R. C. Daniels, C. M. Caramanis, and R. W. Heath, “Adaptation in convolutionally coded mimo-ofdm wireless systems through supervised learning and snr ordering,” *IEEE Transactions on Vehicular Technology*, vol. 59, no. 1, pp. 114–126, 2010.

[42] D. J. Fagnant and K. Kockelman, “Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations,” *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[43] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. F. R. Jesus, R. F. Berriel, T. M. Paixão, F. W. Mutz, T. Oliveira-Santos, and A. F. de Souza, “Self-driving cars: A survey,” *Clinical Orthopaedics and Related Research*, vol. abs/1901.04407, 2019. [Online]. Available: http://arxiv.org/abs/1901.04407

[44] C. Rödel, S. Stadler, A. Meschtscherjakov, and M. Tscheligi, “Towards autonomous cars: The effect of autonomy levels on acceptance and user experience,” in *Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, ser. AutomotiveUI ’14. New York, NY, USA:

Association for Computing Machinery, 2014, pp. 11:1–11:8. [Online]. Available: http://doi.acm.org/10.1145/2667317.2667330

[45] M. Ibrahim, M. Torki, and M. Elnainay, "Cnn based indoor localization using rss time-series," 06 2018.

[46] S. A. Miller and B. R. Heard, "The environmental impact of autonomous vehicles depends on adoption patterns," *Environmental Science & Technology*, vol. 50, no. 12, pp. 6119–6121, 2016, pMID: 27285419. [Online]. Available: https://doi.org/10.1021/acs.est.6b02490

[47] R. Domínguez, E. Onieva, J. Alonso, J. Villagra, and C. González, "Lidar based perception solution for autonomous vehicles," in *2011 11th International Conference on Intelligent Systems Design and Applications*, Nov 2011, pp. 790–795.

[48] J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art," *Clinical Orthopaedics and Related Research*, vol. abs/1704.05519, 2017. [Online]. Available: http://arxiv.org/abs/1704.05519

[49] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *Clinical Orthopaedics and Related Research*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[50] J. Dickmann, J. Klappstein, M. Hahn, N. Appenrodt, H. Bloecher, K. Werber, and A. Sailer, "Automotive radar the key technology for autonomous driving: From detection and ranging to environmental understanding," in *2016 IEEE Radar Conference (RadarConf)*, May 2016, pp. 1–6.

[51] W. Rahiman and Z. Zainal, "An overview of development GPS navigation for autonomous car," in *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, June 2013, pp. 1112–1118.

[52] M. Zhou, X. Qu, and S. Jin, "On the impact of cooperative autonomous vehicles in improving freeway merging: A modified intelligent driver model-based approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 6, pp. 1422–1428, June 2017.

[53] D. Moser, P. Leu, V. Lenders, A. Ranganathan, F. Ricciato, and S. Capkun, "Investigation of multi-device location spoofing attacks on air traffic control and possible countermeasures," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 375–386. [Online]. Available: https://doi.org/10.1145/2973750.2973763

[54] M. Schäfer, P. Leu, V. Lenders, and J. Schmitt, "Secure motion verification using the doppler effect," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2016, pp. 135–145.

[55] M. Schäfer, V. Lenders, and J. Schmitt, "Secure track verification," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 199–213.

[56] M. Monteiro, A. Barreto, T. Kacem, J. Carvalho, D. Wijesekera, and P. Costa, "Detecting malicious ads-b broadcasts using wide area multilateration," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. IEEE, 2015, pp. 4A3–1.

[57] S. Capkun and J.-P. Hubaux, "Secure positioning of wireless devices with application to sensor networks," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3. IEEE, 2005, pp. 1917–1928.

[58] Y. Chen, W. Trappe, and R. P. Martin, "Attack detection in wireless localization," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 1964–1972.

[59] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[60] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[61] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

[62] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[63] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1765–1773.

[64] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," *arXiv preprint arXiv:1702.04267*, 2017.

[65] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *CoRR*, vol. abs/1602.02697, 2016. [Online]. Available: http://arxiv.org/abs/1602.02697

[66] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.

[67] O. Poursaeed, I. Katsman, B. Gao, and S. Belongie, "Generative adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4422–4431.

[68] N. Akhtar, J. Liu, and A. Mian, "Defense against universal adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3389–3398.

[69] A. Fawzi, O. Fawzi, and P. Frossard, "Analysis of classifiers' robustness to adversarial perturbations," *Machine Learning*, vol. 107, no. 3, pp. 481–508, 2018.

[70] W. Zhou, X. Hou, Y. Chen, M. Tang, X. Huang, X. Gan, and Y. Yang, "Transferable adversarial perturbations," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 452–467.

[71] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," in *Proceedings of the 35th International Conference on Machine Learning*,

ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 284–293. [Online]. Available: https://proceedings.mlr.press/v80/athalye18b.html

[72] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song, "Natural adversarial examples," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 15 262–15 271.

[73] V. L. Thing and J. Wu, "Autonomous vehicle security: A taxonomy of attacks and defences," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2016, pp. 164–170.

[74] S. Prasanna and S. Rao, "An overview of wireless sensor networks applications and security," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. 2, pp. 2231–2307, 2012.

[75] M. Winkler, K.-D. Tuchs, K. Hughes, and G. Barclay, "Theoretical and practical aspects of military wireless sensor networks," *Journal of Telecommunications and Information Technology*, pp. 37–45, 2008.

[76] M. Sadeghi and E. G. Larsson, "Adversarial attacks on deep-learning based radio signal classification," *CoRR*, vol. abs/1808.07713, 2018. [Online]. Available: http://arxiv.org/abs/1808.07713

[77] B. Kim, Y. E. Sagduyu, K. Davaslioglu, T. Erpek, and S. Ulukus, "Over-the-air adversarial attacks on deep learning based modulation classifier over wireless channels," 2020.

[78] B. Flowers, R. M. Buehrer, and W. C. Headley, "Evaluating adversarial evasion attacks in the context of wireless communications," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1102–1113, 2020.

[79] ——, "Communications aware adversarial residual networks for over the air evasion attacks," in *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, 2019, pp. 133–140.

[80] M. DelVecchio, V. Arndorfer, and W. C. Headley, "Investigating a spectral deception loss metric for training machine learning-based evasion attacks," *Proceedings of the 2nd ACM Workshop on Wireless Security and Machine Learning*, Jul 2020. [Online]. Available: http://dx.doi.org/10.1145/3395352.3402624

[81] J. Maroto, G. Bovet, and P. Frossard, "Safeamc: Adversarial training for robust modulation recognition models," 2021.

[82] L. Zhang, S. Lambotharan, G. Zheng, B. A. Sadhan, and F. Roli, "Countermeasures against adversarial examples in radio signal classification," *IEEE Wireless Communications Letters*, pp. 1–1, 2021.

[83] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," *arXiv preprint arXiv:1607.04311*, 2016.

[84] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4794–4802.

[85] R. Sahay, C. G. Brinton, and D. J. Love, "A deep ensemble-based wireless receiver architecture for mitigating adversarial attacks in automatic modulation classification," *IEEE Transactions on Cognitive Communications and Networking*, pp. 1–1, 2021.

[86] K. W. McClintick and A. M. Wyglinski, "Reproduction of" evaluating adversarial evasion attacks in the context of wireless communications" and" convolutional radio modulation recognition networks"," in *Proceedings of the Workshop on Benchmarking Cyber-Physical Systems and Internet of Things*, 2021, pp. 1–5.

[87] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 135–147. [Online]. Available: https://doi.org/10.1145/3133956.3134057

[88] G. Jin, S. Shen, D. Zhang, F. Dai, and Y. Zhang, "Ape-gan: Adversarial perturbation elimination with gan," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 3842–3846.

[89] B. Flowers, R. M. Buehrer, and W. C. Headley, “Evaluating adversarial evasion attacks in the context of wireless communications,” 2019.

[90] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: a statistical view of boosting,” *Annals of Statistics*, vol. 28, p. 2000, 1998.

[91] M. A. Nielsen, “Neural networks and deep learning,” *Determination Press*, 2015.

[92] S. Y. Fei-Fei Li, Justin Johnson, “Cs231n convolutional neural networks for visual recognition,” *http://cs231n.github.io/*, Stanford University, April 2018.

[93] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[94] S. Kay, *Fundamentals of Statistical Signal Processing: Detection theory*, ser. Prentice Hall Signal Processing Series. Prentice-Hall PTR, 1998. [Online]. Available: https://books.google.com/books?id=vA9LAQAAIAAJ

[95] T. S. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling, “Gauge equivariant convolutional networks and the icosahedral CNN,” *CoRR*, vol. abs/1902.04615, 2019. [Online]. Available: http://arxiv.org/abs/1902.04615

[96] 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)*, J. Furnkranz and T. Joachims, Eds., 2010, pp. 807–814.

[97] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[98] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, “Advances in optimizing recurrent networks,” *CoRR*, vol. abs/1212.0901, 2012. [Online]. Available: http://arxiv.org/abs/1212.0901

[99] T. Tieleman and G. Hinton, “RMSprop Gradient Optimization.” [Online]. Available: http://www.cs.toronto.edu/~{}tijmen/csc321/slides/lecture_slides_lec6.pdf

[100] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” EECS Department, University of

California, Berkeley, Tech. Rep. UCB/EECS-2010-24, Mar 2010. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html

[101] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[102] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: Where bigger models and more data hurt," *CoRR*, vol. abs/1912.02292, 2019. [Online]. Available: http://arxiv.org/abs/1912.02292

[103] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145 – 151, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608098001166

[104] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011, pp. 693–701. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/218a0aefd1d1a4be65601cc6ddc1520e-Paper.pdf

[105] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. a. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012, pp. 1223–1231. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf

[106] B. McMahan and M. Streeter, "Delay-tolerant algorithms for asynchronous distributed online learning," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014, pp. 2915–2923. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5cce8dede893813f879b873962fb669f-Paper.pdf

[107] S. Zhang, A. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," 2015.

[108] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016.

[109] H. Robbins, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 2007.

[110] J. Lucas, R. S. Zemel, and R. B. Grosse, "Aggregated momentum: Stability through passive damping," *CoRR*, vol. abs/1804.00325, 2018. [Online]. Available: http://arxiv.org/abs/1804.00325

[111] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[112] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[113] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *CoRR*, vol. abs/1904.09237, 2019. [Online]. Available: http://arxiv.org/abs/1904.09237

[114] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. [Online]. Available: http://arxiv.org/abs/1711.05101

[115] J. Ma and D. Yarats, "Quasi-hyperbolic momentum and adam for deep learning," *CoRR*, vol. abs/1810.06801, 2018. [Online]. Available: http://arxiv.org/abs/1810.06801

[116] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, "Adding gradient noise improves learning for very deep networks," 2015.

[117] D. Mishkin and J. Matas, "All you need is a good init," 2016.

[118] S. K. Kumar, "On weight initialization in deep neural networks," *CoRR*, vol. abs/1704.08863, 2017. [Online]. Available: http://arxiv.org/abs/1704.08863

[119] C. Cortes, M. Mohri, and A. Rostamizadeh, "L2 regularization for learning kernels," *arXiv preprint arXiv:1205.2653*, 2012.

[120] Y. Li and F. Liu, "Whiteout: Gaussian adaptive noise regularization in deep neural networks," 2018.

[121] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[122] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 448–456.

[123] ——, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning.* PMLR, 2015, pp. 448–456.

[124] M. Kulin, T. Kazaz, I. Moerman, and E. D. Poorter, "End-to-end learning from spectrum data: A deep learning approach for wireless signal identification in spectrum monitoring applications," *CoRR*, vol. abs/1712.03987, 2017. [Online]. Available: http://arxiv.org/abs/1712.03987

[125] R. N. Bracewell and R. N. Bracewell, *The Fourier transform and its applications.* McGraw-Hill New York, 1986, vol. 31999.

[126] J. Neter, M. H. Kutner, C. J. Nachtsheim, W. Wasserman *et al.*, "Applied linear statistical models," 1996.

[127] K. P. F.R.S., "On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. [Online]. Available: https://doi.org/10.1080/14786440109462720

[128] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[129] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[130] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends in Machine Learning*, vol. 12, no. 4, pp. 307 – 392, 2019. [Online]. Available: http://dx.doi.org/10.1561/2200000056

[131] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The american statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[132] C. K. Carter and R. Kohn, "On gibbs sampling for state space models," *Biometrika*, vol. 81, no. 3, pp. 541–553, 1994.

[133] R. M. Neal, "Annealed importance sampling," *Statistics and computing*, vol. 11, no. 2, pp. 125–139, 2001.

[134] W. R. Gilks and P. Wild, "Adaptive rejection sampling for gibbs sampling," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 41, no. 2, pp. 337–348, 1992.

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

[136] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection." Morgan Kaufmann, 1995, pp. 1137–1143.

[137] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[138] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[139] M. Kalli, J. Griffin, and S. Walker, "Slice sampling mixture models," *Statistics and Computing*, vol. 21, pp. 93–105, 01 2011.

[140] D. M. Blei and M. I. Jordan, "Variational inference for dirichlet process mixtures," *Bayesian Analysis*, vol. 1, pp. 121–144, 2005.

[141] J. Chen, J. Zhu, Y. W. Teh, and T. Zhang, "Stochastic expectation maximization with variance reduction," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 7967–7977. [Online]. Available: http://papers.nips.cc/paper/8021-stochastic-expectation-maximization-with-variance-reduction.pdf

[142] T. Rydén, "Em versus markov chain monte carlo for estimation of hidden markov models: a computational perspective," *Bayesian Analysis*, vol. 3, no. 4, p. 659–688, 2008. [Online]. Available: https://projecteuclid.org/euclid.ba/1340370402

[143] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in *International Conference on Engineering Applications of Neural Networks*. Springer, 2016, pp. 213–226.

[144] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, Feb 2018.

[145] Y. Cui and S. S. Ge, "Autonomous vehicle positioning with gps in urban canyon environments," *IEEE transactions on robotics and automation*, vol. 19, no. 1, pp. 15–25, 2003.

[146] I.-S. Koh and K. Sarabandi, "Polarimetric channel characterization of foliage for performance assessment of gps receivers under tree canopies," *IEEE Transactions on Antennas and Propagation*, vol. 50, no. 5, pp. 713–726, 2002.

[147] R. L. Fante and J. J. Vaccaro, "Wideband cancellation of interference in a gps receive array," *IEEE Transactions on Aerospace and Electronic systems*, vol. 36, no. 2, pp. 549–564, 2000.

[148] R. H. Mitch, R. C. Dougherty, M. L. Psiaki, S. P. Powell, B. W. O'Hanlon, J. A. Bhatti, and T. E. Humphreys, "Signal characteristics of civil gps jammers," in *Pro-*

ceedings of the 24th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2011), 2011, pp. 1907–1919.

[149] K. A. Fisher, "The navigation potential of signals of opportunity-based time difference of arrival measurements," 2005.

[150] J. F, Raquet, and M. Mikel, "Issues and approaches for navigation using signals of opportunity," 01 2007.

[151] J. Morales, P. F. Roysdon, and Z. M. Kassas, "Signals of opportunity aided inertial navigation," 2016.

[152] C. Yang, T. Nguyen, and E. Blasch, "Mobile positioning via fusion of mixed signals of opportunity," *IEEE Aerospace and Electronic Systems Magazine*, vol. 29, no. 4, pp. 34–46, 2014.

[153] J. A. McEllroy, "Navigation using signals of opportunity in the am transmission band," AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING, Tech. Rep., 2006.

[154] I. Guvenc and C.-C. Chong, "A survey on toa based wireless localization and nlos mitigation techniques," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, pp. 107–124, 2009.

[155] V. Moghtadaiee, A. G. Dempster, and S. Lim, "Indoor localization using fm radio signals: A fingerprinting approach," in *2011 International Conference on Indoor Positioning and Indoor Navigation*, 2011, pp. 1–7.

[156] C. Xiao, D. Yang, Z. Chen, and G. Tan, "3-d ble indoor localization based on denoising autoencoder," *IEEE Access*, vol. 5, pp. 12 751–12 760, 2017.

[157] X. Zhang, J. Wang, Q. Gao, X. Ma, and H. Wang, "Device-free wireless localization and activity recognition with deep learning," in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, 2016, pp. 1–5.

[158] M. Mohammadi, A. Al-Fuqaha, M. Guizani, and J. Oh, "Semisupervised deep reinforcement learning in support of iot and smart city services," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 624–635, 2018.

[159] M. Ibrahim, M. Torki, and M. Elnainay, "Cnn based indoor localization using rss time-series," 06 2018.

[160] A. Niitsoo, T. Edelhäußer, and C. Mutschler, "Convolutional neural networks for position estimation in tdoa-based locating systems," in *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2018, pp. 1–8.

[161] I. N. Sneddon, *Fourier transforms.* Courier Corporation, 1995.

[162] P. P. Gandhi and S. A. Kassam, "Optimality of the cell averaging cfar detector," *IEEE Transactions on Information Theory*, vol. 40, no. 4, pp. 1226–1228, 1994.

[163] R. Zhou, Y. Xiong, G. Xing, L. Sun, and J. Ma, "Zifi: Wireless lan discovery via zigbee interference signatures," in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, 2010, pp. 49–60.

[164] A. Norrdine, "An algebraic solution to the multilateration problem," in *Proceedings of the 15th international conference on indoor positioning and indoor navigation, Sydney, Australia*, vol. 1315, 2012.

[165] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator," *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, pp. 233–268, 2017.

[166] J. T. Chiang, J. J. Haas, J. Choi, and Y.-C. Hu, "Secure location verification using simultaneous multilateration," *IEEE Transactions on Wireless Communications*, vol. 11, no. 2, pp. 584–591, 2011.

[167] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics).* Berlin, Heidelberg: Springer-Verlag, 2006.

[168] S.-h. Jung, B.-c. Moon, and D. Han, "Unsupervised learning for crowdsourced indoor localization in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 15, no. 11, pp. 2892–2906, 2015.

[169] L. Li, W. Yang, and G. Wang, "Hiwl: An unsupervised learning algorithm for indoor wireless localization," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications.* IEEE, 2013, pp. 1747–1753.

[170] Y. Li, X. Hu, Y. Zhuang, Z. Gao, P. Zhang, and N. El-Sheimy, "Deep reinforcement learning (drl): Another perspective for unsupervised wireless localization," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6279–6287, 2019.

[171] L. Li, W. Yang, M. Z. Alam Bhuiyan, and G. Wang, "Unsupervised learning of indoor localization based on received signal strength," *Wireless Communications and Mobile Computing*, vol. 16, no. 15, pp. 2225–2237, 2016.

[172] K. W. McClintick, M. Page, T. Wickramarathne, and A. M. Wyglinski, "Machine learning-based roadside vehicular traffic localization via opportunistic wireless sensing," in *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2019, pp. 1–5.

[173] R. Kaune, "Accuracy studies for tdoa and toa localization," in *2012 15th International Conference on Information Fusion.* IEEE, 2012, pp. 408–415.

[174] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *CoRR*, vol. abs/1612.00593, 2016. [Online]. Available: http://arxiv.org/abs/1612.00593

[175] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *2007 IEEE conference on computer vision and pattern recognition.* IEEE, 2007, pp. 1–8.

[176] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET).* Ieee, 2017, pp. 1–6.

[177] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[178] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of artificial intelligence research*, vol. 11, pp. 169–198, 1999.

[179] M. Sewell, "Ensemble learning," *RN*, vol. 11, no. 02, pp. 1–34, 2008.

[180] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[181] J. Tellinghuisen, "Least squares with non-normal data: estimating experimental variance functions," *Analyst*, vol. 133, no. 2, pp. 161–166, 2008.

[182] Y. Zhou, J. Li, and L. Lamont, "Multilateration localization in the presence of anchor location uncertainties," in *2012 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2012, pp. 309–314.

[183] C. Jo and C. Lee, "Multilateration method based on the variance of estimated distance in range-free localisation," *Electronics Letters*, vol. 52, no. 12, pp. 1078–1080, 2016.

[184] A. Savvides, H. Park, and M. B. Srivastava, "The n-hop multilateration primitive for node localization problems," *Mobile Networks and Applications*, vol. 8, no. 4, pp. 443–451, 2003.

[185] "Ts508 quickstart guide - maxitpms," https://www.maxitpms.com/u/cms/www/201804/TS508_Quick%20Guide_V3.pdf, accessed: 2022-01-01.

[186] A. Kolodgie, P. Berges, R. Burrow, M. Carman, J. Collins, S. Bair, G. D. Moy, J. M. Ernst, and A. J. Michaels, "Enhanced tpms security through acceleration timed transmissions," in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, 2017, pp. 35–39.

[187] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[188] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, 2020.

[189] W. Stefansky, "Rejecting outliers in factorial designs," *Technometrics*, vol. 14, no. 2, pp. 469–479, 1972.

[190] F. W. Scholz and M. A. Stephens, "K-sample anderson–darling tests," *Journal of the American Statistical Association*, vol. 82, no. 399, pp. 918–924, 1987.

[191] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *International conference on machine learning*. PMLR, 2018, pp. 274–283.

[192] ——, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 274–283. [Online]. Available: https://proceedings.mlr.press/v80/athalye18a.html

[193] Y.-S. Wang, T.-W. Weng, and L. Daniel, "Verification of neural network control policy under persistent adversarial perturbation," *arXiv preprint arXiv:1908.06353*, 2019.

[194] C. R. Serrano, P. M. Sylla, and M. A. Warren, "Generate and verify: Semantically meaningful formal analysis of neural network perception systems," *arXiv preprint arXiv:2012.09313*, 2020.

[195] D. Gopinath, G. Katz, C. S. Pasareanu, and C. Barrett, "Deepsafe: A data-driven approach for checking adversarial robustness in neural networks," *arXiv preprint arXiv:1710.00486*, 2017.

[196] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification.* Springer, 2017, pp. 97–117.

[197] N. S. Alagha and P. Kabal, "Generalized raised-cosine filters," *IEEE Transactions on Communications*, vol. 47, no. 7, pp. 989–997, 1999.

[198] I. W. Selesnick and C. S. Burrus, "Generalized digital butterworth filter design," *IEEE Transactions on Signal Processing*, vol. 46, no. 6, pp. 1688–1694, 1998.

[199] N. E. West and T. J. O'Shea, "Deep architectures for modulation recognition," 2017.

[200] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[201] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[202] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics.* JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[203] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics.* JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[204] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[205] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[206] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[207] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[208] D. Wu, Y. Wang, S.-T. Xia, J. Bailey, and X. Ma, “Skip connections matter: On the transferability of adversarial examples generated with resnets,” *arXiv preprint arXiv:2002.05990*, 2020.

[209] T. Na, J. H. Ko, and S. Mukhopadhyay, “Cascade adversarial machine learning regularized with a unified embedding,” *arXiv preprint arXiv:1708.02582*, 2017.

[210] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[211] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, 2009, pp. 248–255.

[212] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!" *arXiv preprint arXiv:1904.12843*, 2019.

[213] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[214] H. Van Trees, *Optimum Array Processing: Part IV of Detection, Estimation, and Modulation Theory*, ser. Detection, Estimation, and Modulation Theory. Wiley, 2004. [Online]. Available: https://books.google.com/books?id=K5XJC_fMMAwC

[215] H. L. V. Trees, *Detection, Estimation, and Modulation Theory: Radar-Sonar Signal Processing and Gaussian Signals in Noise.* Melbourne, FL, USA: Krieger Publishing Co., Inc., 1992.

[216] C. Park, J. Choi, S. Nah, W. Jang, and D. Y. Kim, "Automatic modulation recognition of digital signals using wavelet features and svm," in *2008 10th International Conference on Advanced Communication Technology*, vol. 1, 2008, pp. 387–390.

[217] T. J. O'Shea and J. Corgan, "Convolutional radio modulation recognition networks," *CoRR*, vol. abs/1602.04105, 2016. [Online]. Available: http://arxiv.org/abs/1602.04105

[218] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[219] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[220] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[221] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.

[222] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," *ACM sigmod record*, vol. 25, no. 2, pp. 103–114, 1996.