Adversarial Robustness of Deep Learning Models: An Error-Correcting Codes based Approach

by

Samarth Gupta

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Transportation

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2020

Signature redacted

Author .....
Department of Civil and Environmental Engineering
January 15, 2020

Signature redacted

Certified by .....
Saurabh Amin
Associate Professor of Civil and Environmental Engineering
Thesis Supervisor

Signature redacted

Accepted by .....
Colette L. Heald
Professor of Civil and Environmental Engineering
Chair, Graduate Program Committee

Adversarial Robustness of Deep Learning Models: An Error-Correcting Codes based Approach

by

Samarth Gupta

Submitted to the Department of Civil and Environmental Engineering
on January 15, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Transportation

Abstract

Efficient operation and control of modern day urban systems such as transportation networks is now more important than ever due to huge societal benefits. Low cost network-wide sensors generate large amounts of data which needs to processed to extract useful information necessary for operational maintenance and to perform real-time control. Modern Machine Learning (ML) systems, particularly Deep Neural Networks (DNNs), provide a scalable solution to the problem of information retrieval from sensor data. Therefore, Deep Learning systems are increasingly playing an important role in day-to-day operations of our urban systems and hence cannot not be treated as standalone systems anymore. This naturally raises questions from a security viewpoint. Are modern ML systems robust to adversarial attacks for deployment in critical real-world applications? If not, then how can we make progress in securing these systems against such attacks?

In this thesis we first demonstrate the vulnerability of modern ML systems on a real world scenario relevant to transportation networks by successfully attacking a commercial ML platform using a traffic-camera image. We review different methods of defense and various challenges associated in training an adversarially robust classifier.

In terms of contributions, we propose and investigate a new method of defense to build adversarially robust classifiers using Error-Correcting Codes (ECCs). The idea of using Error-Correcting Codes for multi-class classification has been investigated in the past but only under nominal settings. We build upon this idea in the context of adversarial robustness of Deep Neural Networks. Following the guidelines of codebook design from literature, we formulate a discrete optimization problem to generate codebooks in a systematic manner. This optimization problem maximizes minimum hamming distance between codewords of the codebook while maintaining high column separation. Using the optimal solution of the discrete optimization problem as our codebook, we then build a (robust) multi-class classifier from that codebook.

To estimate the adversarial accuracy of ECC based classifiers resulting from different codebooks, we provide methods to generate gradient based white-box attacks. We discuss estimation of class probability estimates (or scores) which are in itself

useful for real-world applications along with their use in generating black-box and white-box attacks. We also discuss differentiable decoding methods, which can also be used to generate white-box attacks.

We are able to outperform standard all-pairs codebook, providing evidence to the fact that *compact* codebooks generated using our discrete optimization approach can indeed provide high performance. Most importantly, we show that ECC based classifiers can be *partially robust even without any adversarial training.* We also show that this robustness is simply not a manifestation of the large network capacity of the overall classifier. Our approach can be seen as the first step towards designing classifiers which are *robust by design.* These contributions suggest that ECCs based approach can be useful to improve the robustness of modern ML systems and thus making urban systems more resilient to adversarial attacks.

Thesis Supervisor: Saurabh Amin
Title: Associate Professor of Civil and Environmental Engineering

Acknowledgments

I would like to thank my advisor Prof. Saurabh Amin for his guidance and help throughout my time at MIT. The current form of this thesis is the result of numerous invaluable discussions with him. He has been a constant source of inspiration. I am fortunate to have him as my advisor.

I would also like to express my gratitude to Prof. Aleksander Mądry for his guidance and for allowing me to attend his lab's group discussions and meetings. I have immensely benefited from participating in his lab's reading group.

I am also grateful to Prof. Moshe E. Ben-Akiva for providing me with a RA position in ITS lab during my first year at MIT.

I am extremely grateful to the MST program administrator Kiley Clapper for her support and guidance. She has been extremely supportive and helpful on several occasions. I would also like to thank other member of the CEE-HQ including Max, Sarah and many others who work tirelessly to ensure the well-being of students.

I would also like to thank my family and friends for making this journey possible.

Finally, I would like to acknowledge and thank the numerous financial support received through Robert Thurber Fellowship, Schoettler Scholarship Fund and through the Office of Graduate Education MIT.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There is an ever increasing need for intelligent operation of today's resource-constrained urban-systems to cater to the need of growing population. Transportation, water, gas and electricity distribution networks are primary examples of such urban systems. These urban systems need to be intelligently monitored and controlled. Transportation networks pose an interesting challenge due to the spatial limitation for expansion (limited land), high cost associated with expansion (if possible), increasing usage due to rising population and continued growth in private vehicle ownership (especially in growing economies). This mismatch of highly constrained *supply* and rising *demand*, exacerbates the problem of traffic congestion on a daily basis. Therefore, it is imperative to utilize the existing infrastructure in an *optimal* manner. This critical need to mitigate congestion by effective utilization of existing infrastructure is one of the main motivations behind the (now prominent) vision of *Intelligent Transportation Systems* (ITS) or *smart-cities*.

In recent years, many urban systems are increasingly relying on data-driven operations in an effort to achieve faster response to emergency situations and real-time control in day-to-day nominal situations. These capabilities is largely enabled by the network-wide deployment of a variety of low-cost, high resolution sensors and remotely controllable actuators (or control devices/mechanisms). Transportation networks operationally rely on heterogeneous data collected through a variety of sources such as traffic-cams, loop-detectors, in-vehicle GPS devices, etc. Sensors provide multi-resolution heterogeneous data to the network operators, who are

responsible for implementing control (via actuators) to regulate the state of transportation systems. Ramp-meters, traffic-lights, toll-gantries and speed limits (and other alerts) implemented through Variable message signs (VMS) are common for controlling transportation networks. With sensors and actuators in place, the operation of a transportation network can be viewed as a *closed-loop (or feedback) control of a dynamical system.* Security of control systems for critical infrastructure is an important problem because of huge societal losses associated when the security of such systems is compromised.

Automated handling and processing of raw data, especially for *real-time control* becomes challenging due to following reasons:

1) Heterogeneity in data collected through different sensors.

2) Lack of scalability and low accuracy of classical pattern recognition systems.

3) Inability to develop resilient systems in the face of random failures of certain sensors.

4) High setup and operational costs.

5) Privacy

In most cities, the system operators in Traffic Management Centres (TMCs) have access to a large amounts of real-time traffic-camera data. However due to above outlined reasons, a lot of this data is often discarded and not directly used in real-time decision making. Instead, low resolution noisy data collected through loop-detectors is commonly used for estimating vehicle-counts and link flow speeds. On the other hand, camera data is mainly used by human operators for monitoring and maintaining situational awareness.

Fortunately, modern Machine Learning (ML) tools have the potential to solve the problem of *information retrieval* from large scale heterogeneous raw data. In recent years, the field of machine learning (particularly Deep Learning) has seen significant advancements. Numerous success stories have been reported on achieving human level performance on various image classification tasks, object detection, speech translation, reinforcement learning etc. For a detailed discussion on these advancements, see the Nature article by LeCun *et al.* [22]. Human-level performance, high scalability

and ability to learn from multiple heterogeneous data sources makes deep learning an attractive ML tool for extracting valuable information from sensor-data to improve the operation of urban transportation systems. While this can be a major advancement in the field of ITS, naive adoption of deep learning models without having a complete theoretical understanding of their behavior can lead to many undesirable consequences. Given that we currently lack a systematic theory for these models, we need to be prudent to have some operational guarantees, or at least be aware of the situations where these models may fail. Importantly, incorporating Deep Learning models in operational control of transportation systems raises new questions from a *security* viewpoint.

Are modern deep learning systems robust enough to be deployed in real-world urban systems? If not, how can we train deep learning models which are adversarially robust?

1.1 Motivating example

We provide a real-world example to highlight the lack of robustness of the state-of-the-art ML systems. Consider an image taken from a road traffic camera in Cambridge, Massachusetts, shown in figure 1-1. We pass this image through a commercial object detection service provided by Google through their Google Cloud Vision (GCV) platform[1]. GCV correctly identifies "car" as an object in the image, see figure 1-2. We now as *attacker*, perturb this image in a specific manner such that the added perturbations[2] are almost *imperceptible* to human eye. GCV now instead of identifying "car", identifies "ladder" as an object in the image, see figure 1-2. Missed-detection of car is in itself undesirable; however identifying "ladder" is by all means erroneous and unacceptable. This is one of the several examples which signify that despite their high *nominal* accuracy commercial ML systems are not adversarially robust. This limits their trustworthiness for safety-critical applications such as traffic control and

[1] https://cloud.google.com/vision/

[2] In chapter 2, section 2.4 we will discuss on how to generate such attacks in detail.

incident management.

Figure 1-1: A traffic-cam image

We note some key aspects of the above mentioned attack: Firstly, we did not use any information about the underlying classifier(s) such as the GCV model architecture and corresponding parameter values. Secondly, GCV does not provide a full list of objects (or classes) from which it chooses the final object(s). Indeed, these unknowns make attacking GCV harder (from the viewpoint of an attacker) than a regular multi-class classifier for which the total number of classes and the probability estimates for each class is typically known. The fact that even with little to almost no-information, we are able to successfully orchestrate an attack clearly demonstrates the vulnerability of modern ML systems.

We believe that the contextual nature of the above example makes the broader problem of robustness of ML systems even more intriguing. One can reason as to *why* GCV would classify pavement markings as a ladder. Geometrically, pavement markings in the image indeed share the same structure or form as that of a ladder. Also, the bounding box provide by GCV shown in figure 1-2 is quite well-positioned covering the region correctly under the assumption that its a ladder. Thus, the example also highlights how current ML systems do not account for the contextual information in decision-making and typically do not provide appropriate reasoning or interpretation with their predictions. This limitation provides another reason to study the adversarial robustness of modern ML systems.

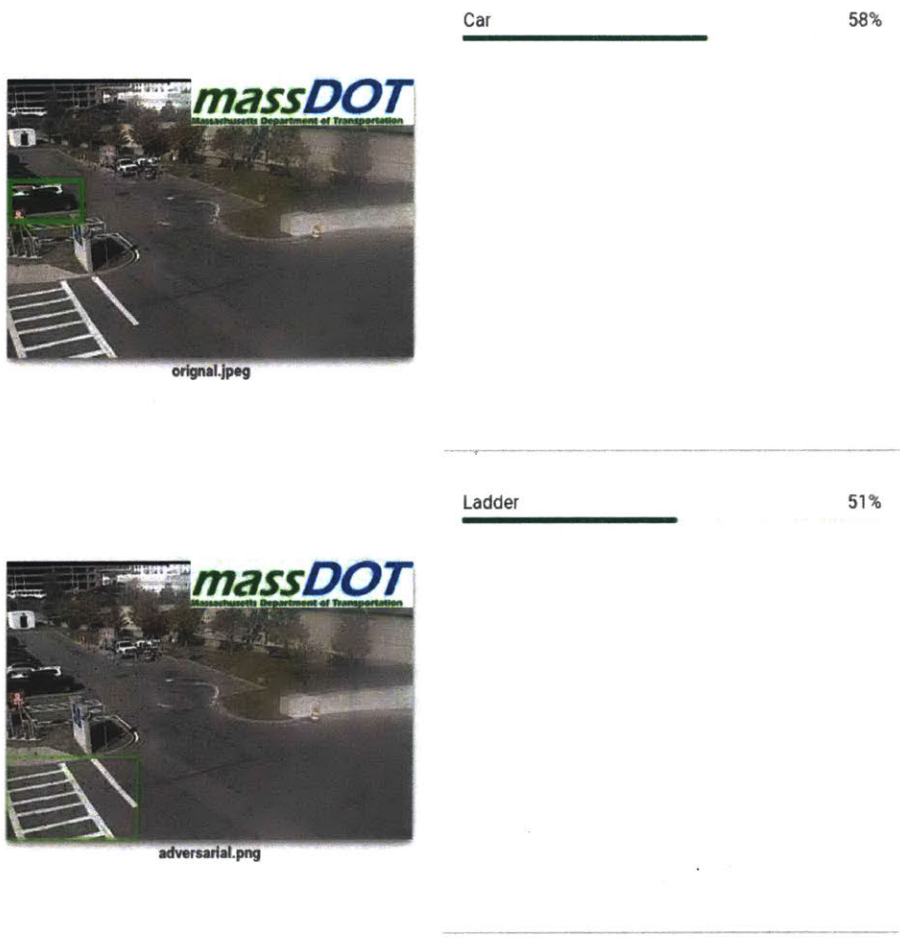


Figure 1-2: Successful demonstration of an adversarial attack on GCV using a traffic-cam image from Massachusetts Department of Transportation

We now briefly discuss how the brittleness of ML systems to adversarial perturbations can comprise the security of the overall system. As mentioned earlier, network operators in Traffic management centres (TMCs) often depend on image data from Closed-Circuit Television (CCTV) cameras for remote monitoring and situational awareness. However, various TMCs are contemplating the use of modern ML methods for automatic monitoring and real-time estimation of road network conditions. Based on these estimates, TMC operators aim to provide real-time control in the form of ramp-metering, tolls and traffic lights. There is an ever increasing effort to streamline and automate this process in order to reduce human effort. However, due to limited financial resources and lack of in-house ML expertise, TMC operators

out of choice may end up relying on using commercial services like GCV or using out-of-the-box pre-trained classifiers provided by a third party.

Thus, we need to pro-actively consider and address all security related concerns arising from the brittleness of ML systems, even if they may appear to be unlikely. For example an adversary (external hacker or malicious insider) can manipulate the vehicle count estimates of a network operator (obtained via processing the feed of a traffic camera) by placing an adversarial patch or object on the road. Even manipulating a single camera in a strategic manner can have network wide effects due to disturbance propagation characteristic of congested transportation networks. We conjecture that post-hoc detection of such an attack can be very difficult as one may need to first localize the spatial origin of disturbance and then manually review hundreds of hours of regional camera footage. To make things even worse, in many TMCs, camera data is either not stored or often discarded after a certain period of time, typically in few hours or a week. For further details on the operational guidelines regarding storage of traffic camera data we refer the reader to the report [21] from US Department of Transportation (DOT).

From a system's viewpoint, one can approach the above problem in two ways :

1. In anticipation of such attacks, one can aim to design and operate urban systems under the worst case scenarios or provide improved resiliency guarantees for well defined class of attacks. For a gentle introduction to secure control see Cárdenas *et al.* [9]. However, including resilience generally comes at the cost of reduced system performance under nominal conditions. Furthermore, this approach may not provide any operational guarantees in terms of thwarting *all* possible attacks. Robust and secure control continues to be an active and growing field of research (see Cárdenas *et al.* [10]) and more progress is needed in improving the trade-off between security and efficiency.

2. A complementary approach to secure control would be to fix the vulnerabilities at their source; for example, by designing ML systems with robustness guarantees against a large class of adversarial perturbations. In some applications this

approach might be easier and more beneficial. One can view this as the first line of defense on an individual sensor level in the overall cyber-physical system. Given the success and wide adoption of Deep Neural Networks (DNNs) in solving a variety of complex tasks it is imperative to design robust ML systems. Making progress on this problem is important because a large number of ML applications are open-loop in nature, where recovery from an adversarial attack may not be possible without incurring instantaneous harm. For instance, Sharif *et al.* were able to bypass face recognition systems using eyeglass frames as an accessory. Carlini *et al.* [11] demonstrated the spoofing of a voice recognition system. In such open-loop scenarios it is necessary to have a robust ML system in place.

1.2 Problem Formulation

A classification system is a prototypical example of a ML system that is useful in a variety of operational situations in transportation networks. A highly relevant example to our discussion would be of high-occupancy vehicle (HOV) lanes, where vehicles only with a certain number of passengers (generally two or more) are allowed to drive in order to encourage car-pooling. For enforcement, vision systems are commonly used to classify whether a vehicle qualifies as an HOV or not. There have been multiple reports [2], [3] where the drivers use inflatable dolls or cardboard cutouts to bypass detection. The driver in such situations can also be viewed as an attacker. Strict and fair compliance can only be ensured with a robust vision system in place, otherwise single drivers tend to misuse the system [1]. A machine learning system which classifies vehicle as a car, bus, motorbike etc. would be another example of classification system being used as a part of ITS.

In this thesis we aim to train a multi-class classifier which is robust over some predefined uncertainty set $\mathcal{S}$. We first briefly outline the general setup associated with a classification problem and then introduce the robust version.

Typically we are provided with N training data samples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$

drawn from some unknown underlying distribution $\mathcal{P}$. The goal is to learn a function f parameterized over θ which minimizes the expected risk $E_{(x,y)\sim\mathcal{P}}[L(f(x,\theta),y)]$, where $L(\cdot,\cdot)$ denotes some loss function. This can be written as:

$$\theta^* = \arg\min_{\theta} E_{(x,y)\sim\mathcal{P}}[L(f(x,\theta),y)] \tag{1.1}$$

Since the underlying distribution is unknown and we only have access to N i.i.d training examples, the expectation is generally replaced with a simple average over the training examples. This is commonly referred to as the Empirical Risk Minimization (ERM). Traditionally, this approach has been extremely successful in training classifiers with high prediction accuracy. However, numerous studies [30] [16], have show that the resulting classifier, particularly Deep neural networks are not robust to adversarial noise.

Before proceeding further, we first define what do we mean by *robust?* Suppose $\hat{y}_i$ be the correct class predicted by classifier f over some input x. An adversary adds some noise $\delta \in \mathcal{S}$ to perturb the input $x \mapsto x + \delta$ in a manner that the classifier now predicts a different class. A robust classifier should be resilient to such perturbations. The allowed set of perturbations from which the adversary can choose is referred to as the uncertainty set $\mathcal{S}$. To train a classifier which is robust to such perturbations for a given uncertainty set $\mathcal{S}$ is the main problem which we aim to study in this thesis.

The robust version of the optimization problem in eq. (1.1), i.e. the problem of adversarial learning can be written as:

$$\theta^*_{robust} = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{P}}[\max_{\delta\in\mathcal{S}} \mathcal{L}(x+\delta, y, \theta)] \tag{1.2}$$

In the above formulation, the inner constrained optimization problem aims to maximize the loss around a given x and the outer unconstrained optimization problem aims to minimize the loss over all possible perturbations or equivalently, over *worst-case* perturbations. Following [16] [23], the set of allowed perturbations $\mathcal{S}$ is usually defined as the ϵ-radius norm ball (l_2 or l_∞) around a given x. As $\mathcal{P}$ is unknown, expectation is again replaced by a simple sample average over the i.i.d training data

points. The resulting min-max problem (1.3) is solved by minimizing the loss over the adversarial examples which are generated by solving the inner problem.

$$\hat{\theta}^*_{robust} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} [\max_{\delta \in \mathcal{S}} \mathcal{L}(x_i + \delta, y_i, \theta)] \tag{1.3}$$

Solving the inner optimization problem is equivalent to generating an adversarial attack, therefore, for a fixed set of model parameters θ, attacker (or the adversary) aims to solve this problem, while the defender aims to solve the outer problem as a defense. In chapter 2, we will discuss different methods from literature [23] [33] [26] for training a robust model.

1.3 Proposed Approach

In this thesis we investigate a new method of defense to build an adversarially robust classifier using Error-Correcting Codes (ECCs). The idea of using Error-Correcting Codes for multi-class classification was proposed by Dietterich and Bakiri in [13]. We revisit this idea in the context of adversarial robustness. Following the guidelines of codebook design from [13], we formulate a discrete optimization problem to generate codebooks in a systematic manner.

Using the optimal solution of the discrete optimization problem as our predefined codebook, we then build a multi-class classifier using that codebook. To evaluate the adversarial accuracy of ECC based classifiers resulting from different codebooks, we provide methods to generate gradient based white-box attacks to rigorously estimate the adversarial accuracy. Most importantly, we show that ECC based classifiers can be *robust without any adversarial training.* This can be seen as the first step towards designing classifiers which are *robust by design.* We also investigate the effect of adversarial training on the overall accuracy of the codebook. We also show that the robustness achieved without adversarial training is simply not because of the network capacity of the resulting classifier.

Chapter 2

Background and Literature review

In this chapter we discuss different methods of training a robust model and some nuances associated with correctly estimating the adversarial accuracy of a model including Certification and Verification techniques. For the ease of discussion in subsequent chapters, we enumerate different types of threat model.

2.1 Adversarial Training

In this section we briefly discuss the adversarial training methods of Mądry *et al.* [23] and Goodfellow *et al.* [16] to train a robust model. Recall the min-max formulation from previous chapter:

$$\hat{\theta}^*_{robust} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} [\max_{\delta \in \mathcal{S}} \mathcal{L}(x_i + \delta, y_i, \theta)] \tag{2.1}$$

To solve the above min-max formulation, adversarial training methods generate adversarial examples by solving the inner maximization problem and then solves the outer minimization problem over these examples essentially treating them as training data points. The inner problem being a constrained optimization problem is generally solved to generate adversarial training examples using gradient based methods particularly, Projected Gradient Descent (PGD), to maintain feasibility. Solving the inner problem using PGD is also referred to as PGD-attack [23]. Another

popular variant known as Fast-Gradient-Sign Method (FGSM) proposed in [16] is basically a single step PGD attack. Using the adversarial examples (generated via solving the inner problem) as training data points, solving the outer problem is same as standard training but with adversarial examples. Mądry et. al [23] highlights that stronger attacks used to generate adversarial examples for training, lead to more robust networks. These attack based adversarial training methods try to explicitly solve the inner problem without being discouraged by the non-concave nature of the problem.

Recently other methods have been proposed which do not make use of adversarial examples during training. Two such methods proposed by Aditi *et al.* [26] and Wong & Kolter [33] fall under this category. These methods do not solve the inner maximization problem explicitly, instead they bypass solving the inner problem by find an upper bound on the optimal value of the inner maximization problem. They do so by first forming the convex relaxation of the inner problem and then writing the dual of the relaxed (convex) problem. They then try to solve the outer minimization problem over the model parameters θ by minimizing the upper bound. An important practically useful characteristic associated with these defense methods is that they are *certifiably* or *provably* robust. We will subsequently discuss this in more detail.

Before proceeding further, we would like to take a detour to discuss the evaluation of the adversarial accuracy of a given model. This detour will help us to better understand the *certifiable /provable* nature of the aforementioned training methods.

2.2 Estimating Adversarial Accuracy

Evaluation of natural or clean accuracy over an example (generally from test-set) is straightforward, and can be easily done by a simple forward pass through the neural network to compute the class scores and then simply predict the class as the one with the maximum score. However, calculation of the adversarial accuracy is not straightforward as it involves solving a *non-concave* constrained maximization problem. Recall that since we are in the domain of *worst-case* analysis, if the model

correctly predicts over *all* allowed perturbations in $\mathcal{S}$ or equivalently if the model correctly predicts for the *worst-case* perturbation, *only then* we can correctly conclude that the model is robust for that particular example. Due to highly non-convex nature of neural networks, solving the non-concave constrained maximization problem remains the main challenge in correct evaluation of the adversarial accuracy.

We now mathematically define the problem of evaluating the adversarial accuracy. Suppose c be the true class associated with a given input x and let $i \in \{1, \ldots, k\}/\{c\}$ be the target class for which the attacker is trying to generate an adversarial perturbation. Attacker aims to solve the following non-convex problem:

$$f^*(x) = \max_{\delta \in \mathcal{S}} f_i(x + \delta) - f_c(x + \delta) \tag{2.2}$$

For a valid[1] adversarial perturbation, the objective function value of this problem would be strictly positive for some target class i. Note that for the purposes of calculating adversarial accuracy, we do not really care about the *exact* maximizer of (2.2). Instead we are mainly interested in determining one of the following two things:

1. Can we find a feasible solution of (2.2) for which the objective function value is positive? (Category I)
2. Whether the optimal value of the maximizer is strictly positive or not? (Category II)

Different attacks including gradient-based PGD or FGSM attempts to solve the above outlined problem (2.2). Given the non-concave nature of the problem, these attacks do not provide any gaurantee in terms of finding the optimal solution, and hence simply aims at finding a feasible solution to (2.2) with positive objective function value. These attacks therefore fall in Category I. If these attacks fail in generating an adversarial perturbation (especially if the attack is weak), we can *only* conclude that the model is merely robust against that particular attack. The efficacy or strength of a certain attack (or an adversary) is empirically estimated by the number of examples (from the test-set) for which the adversary is able to generate an adversarial

[1] an adversarial perturbation does not necessarily need to be the arg max of (2.2)

perturbation or fool the model. However, failure of any particular attack(s) on a given example *does not guarantee the absence of an adversarial perturbation.*

From the above discussion we can conclude that since different attacks do not provide any guarantee in case of a failure, therefore, such (*non-optimal*) attacks can only provide an upper bound on the true adversarial accuracy, implying that the true adversarial accuracy potentially could be much lower. The strength of the attack will govern the tightness of this upper bound. Therefore: First, to avoid having a false sense of security it is imperative to evaluate against *strong* adversaries. Second, more importantly we need methods using which we can safely compute the *true* adversarial accuracy or atleast get a good lower bound. Interestingly, the need to estimate true adversarial accuracy of a model has given rise to various *Certification* and *Verification* methods.

2.2.1 Certification

Certification methods provide an alternative way of approaching problem (2.2) by determining the sign of f^* (the optimal value of the maximizer of (2.2)). Instead of solving (2.2) explicitly, if we can upper-bound f^* and show that if $f^* < 0$ (strictly), for all target classes, then we can safely conclude that the model is robust to any allowed perturbation for a given x. Equivalently, we can say that we have generated a *certificate of robustness.* These methods fall in Category II.

The upper bound on f^* is generated using the convex relaxations of (2.2). Subsequently, either the convex relaxation (as primal) is directly solved for optimality or its dual is considered. Recall that any feasible dual solution provides a valid upper bound to the primal. Therefore, if we can easily find a feasible dual solution for which the objective function value is negative or if the optimal dual (or primal) solution has negative objective function value, then we have essentially established that $f^* < 0$. However, if the optimal value of the dual is positive, then we cannot conclude anything useful. In such cases we cannot provide a certificate of robustness even though the model may actually be robust for that input x. In practice, depending upon the tightness of the convex relaxation and the size of the uncertainty set $\mathcal{S}$, the up-

per bound can be loose. Therefore, certification methods, although mathematically elegant and useful in practice, are still *incomplete*.

2.2.2 Verification

We now discuss Verification methods which are complete in the sense that they can solve problem (2.2) *exactly* for DNNs with ReLU non-linearity. An important step in this direction was made by Katz *et al.* in [20] as ReLUplex. In this method the authors formulate the problem under the Satisfiability Modulo Theories (SMT) framework and extend the Simplex method to incorporate ReLU constraints to optimally solve (2.2). Their work was motivated by a critical real-world application to verify neural networks used in airborne collision avoidance system for unmanned aircrafts. Subsequently many other works followed based on Mixed Integer Linear Programming (MILP) formulations of ReLU non-linearity [31]. Note that these formulations provides an exact representation of the ReLU network and hence the problem (2.2) can be solved as a MILP. Despite the discrete nature of MILPs, we can still optimally solve them by either using Cutting plane methods or Branch and Bound methods.

Verification methods can also potentially help in training more robust networks. Recall that in the min-max formulation (2.1), the inner maximization problem is solved using gradient based attacks like PGD or FGSM, however due to non-optimality of these attacks, one can only expect to get a good candidate solution. Verification methods can alternatively be used to solve the inner problem optimally or generate attacks stronger than PGD or other attacks. Training against a stronger adversary can lead to more robust networks. To highlight this connection was part of our motivation to digress from our initial discussion on adversarial training. However, given the lack of scalability of current verification methods, it remains to be seen whether these methods will ever be useful in training more robust models.

2.3 Robust Training

After providing a high-level overview of different Certification and Verification methods we now come back to our original problem of understanding robust training methods proposed by Aditi *et al.* [26] and Wong & Kolter [33]. Both these methods form a convex relaxation of the non-convex DNN, thus generating a *differentiable* certificate and subsequently optimizing the model parameters over this certificate.

2.3.1 Aditi *et al.* [26]

Aditi *et al.* in [26] first upper bound eq. (2.2) by maximum of the l_1-norm of the gradients over $\mathcal{S}$. For a two layer network, this is further upper bounded by a non-convex Quadratic Program(QP). Subsequently, a convex relaxation of this QP in the form of a Semi-definite program is formed using the famous MAX-CUT result of Goemans and Williamson [15]. Finally, the upper bound is of the form:

$$\underbrace{f(x)}_{f_i(x)-f_c(x)} + \underbrace{\max_{yy^T \geq 0, \text{diag}(yy^T) \leq 1} \boldsymbol{q}(y, \theta)}_{\text{Regularizer}} \tag{2.3}$$

where x is the input (or the training data) and y is the set of decision variables of the second part of the upper bound (which is a Semi-Definite program). Note that the second part $\boldsymbol{q}(\cdot)$, is independent of x (input from training-set) and only depends on model parameters θ(i.e. the weights of the DNN) and a set of variables y, therefore can be treated as a *regularizer*.

The second part of the upper bounds requires solving a semi-definite program which can be computationally expensive. To bypass this, Aditi *et al.* [26] takes the dual of $\boldsymbol{q}(\cdot)$ such that the resulting form involves computation of the top eigenvalues which is fast. Finally, this is optimized over model parameters θ and variables y using stochastic gradient based methods.

After the training of the model parameters is completed, for further discussion the model parameters θ's are therefore assumed to be fixed. To generate a certificate of

robustness on a new x (from the test-set), one needs to optimally re-solve the second part of eq. (2.3) involving a SDP. However, as this is independent of x, therefore, needs to be solved only once (for every pair of classes). Note that the above method is limited to two layer networks and l_∞-norm based uncertainty set, although it can handle different non-linearities like ReLU, Sigmoid, etc.

2.3.2 Convex outer polytope method of Wong and Kolter [33]

We now discuss the convex-outer polytope approach of training a robust model (particularly a DNN with ReLU non-linearity) proposed by Wong and Kolter in [33]. This method although limited to ReLU nonlinearity however, can handle networks with more than two layers and can be extended to other convex l_p-norm based uncertainty sets and not just l_∞-norm. Consider a Deep neural network with L layers and ReLU as the activation function for all $L-1$ hidden layers. Let $\theta = \{W_1, b_1, \ldots, W_L, b_L\}$ be the set of weights and biases. The following set of equations correspond to the layer-wise transformation of any input $\tilde{x}$:

$$z_0 = \tilde{x} \tag{2.4}$$

$$\hat{z}_i = z_{i-1}W_i + b_i \text{ for } i = 1, 2, \ldots, L \tag{2.5}$$

$$z_i = \max(\hat{z}_i, 0) \quad \text{for } i = 1, 2, \ldots, L-1 \tag{2.6}$$

The final output $f(\tilde{x})$ is simply the vector $\hat{z}_L$. Hence, the score of j^{th} class, i.e. $f_j(\tilde{x}) = \hat{z}_L^j$. In above, eq. (2.5) represents a simple linear transformation, eq. (2.6) represents elements wise non-linearity. This non-linearity is the main reason due to which we end up with a non-linear non-convex program for problem (2.2). To deal with this ReLU non-linearity, Wong and Kolter [33] proposed a convex-relaxation using only linear inequalities. Suppose that for any particular ReLU we know the pre-activation input lower and upper bounds, i.e. l, u. Given these bounds, we can now

represent the convex region shown in fig. 2-1 using the following set of inequalities:

$$z \geqslant 0 \tag{2.7}$$

$$z \geqslant \hat{z} \tag{2.8}$$

$$-u\hat{z} + (u - l)z \leqslant -ul \tag{2.9}$$

We can now use the above set of **linear** inequalities to replace the non-linear ReLU.

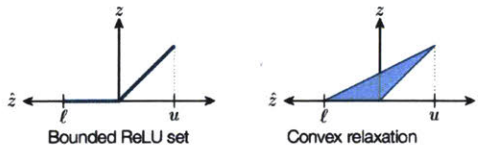


Figure 2-1: Bounded linear relaxation of ReLU non linearity (Image adapted from [33])

We will therefore end up with a linear program. This is commonly referred to as the *LP-relaxation* of the ReLU network.

Wong and Kolter [33] shows that the dual of the LP-relaxation takes a very special form which is similar to a deep neural network and therefore is much easy to work with. Identifying this is one of their important contributions and allowed them to work with large networks. Also, note that one needs to compute the pre-activation bounds to ReLU's, authors provide a method for that as well. Recall that any dual solution gives a bound on the primal, therfore instead of optimizing the dual, authors construct a feasible solution to this dual. This feasible solution gives a bound on the inner-problem. Finally, authors optimize this bound over the network parameters θ to obtain a robust model. Once the training is complete and the model parameters are fixed, we can then solve the convex LP-relaxation (or its dual) to generate a certificate of robustness depending on the optimal objective function value of the LP.

2.3.3 Summary

In our discussion so far, we have covered different methods of training a robust model, including how some of these methods are *certifiably/provably* robust. In principle a

particular certification method can be used to estimate the adversarial accuracy of a model which was trained against certificates generated via some *other* method. For instance, we can use the method of Aditi *et al.* [26] to certify a network trained using Wong and Kolter's [33] approach. However, Aditi *et al.* [26] showed that when doing so, vacous bounds are achieved, therefore, it is important to use the same certificates against which the network was trained.

For a better understanding, we discussed Certification and Verification methods in the context of training a robust model. However, given the need to correctly determine the adversarial accuracy of a model, there have been other independent studies which aim to propose new certification methods, for example Dvijotham *et al.* [14], Aditi *et al.* **[27]**. As this is a very popular and growing field of research, it is not possible to cover everything in detail. However, we would like to point out that, most of the recent studies like [14], [27] build upon the ideas of Aditi *et al.* [26] or Wong and Kolter [33]. Most importantly, all these studies including [26] and [33] make use of standard techniques from convex optimization, particularly *robust* optimization. For details on robust optimization refer to Ben-Tal *et al.* [7].

2.4 Different Threat Models

In order to study the robustness of a model against a particular adversary (or an attack), it is important to have a precise threat model in place or the *information* which an attacker can use to its own advantage. This becomes particularly important since the defender may inadvertently reveal some information which may appear to be innocuous but is still sufficient to compromise security. Moreover, since adversarial robustness of Deep learning models[2] is in itself a new and developing field, therefore the notion of a threat model provides a principled way to make progress. In this section we outline different threat models or the various type of attacks which are most commonly used to evaluate the robustness of a model. These attacks can be

[2]We acknowledge that robustness of different ML models has been extensively studied in the past, our discussion here is mainly concerning Deep Neural Networks, which have **re**gained popularity and adoption in recent years.

broadly categorized into white-box attacks, black-box attacks and partial-information attacks. Futher, each attack can have a different set of allowed perturbations $\mathcal{S}$ such as norm based perturbations.

In black-box setting, only the output of the classifier i.e. the class probabilities or score of each class is known to the attacker. No model information is available to the attacker, i.e. the network architecture and the weights of the network. In this setting, since only class probability estimates are available, therefore analytical computation of gradients is not possible. The problem is generally solved using off-the-shelf black-box optimizers mainly evolutionary algorithms such as Particle Swarm Optimization (PSO), Genetic Algorithms (GAs) etc. However, given the efficacy of gradient based attacks, one can also try to compute an estimate of the gradient and then use this estimate to run gradient-based attacks, for details see Ilyas *et al.* [19]. SPSA proposed by Spall in [29] is another black-box optimization method which is based on gradient estimation, see [28] for an overview.

In white-box setting, the class probability estimates along with the model architecture and weights are known to the attacker. White-box setting can also be referred to as complete information setting. In white-box setting, the projected gradient descent or the PGD-attack proposed in Mądry *et al.* [23] has emerged as one of the strongest known attack. Another very popular gradient based attack known as Fast-Gradient-Sign method (FGSM) was proposed by Goodfellow *et al.* in [16] and is still commonly used for benchmarking. FGSM can be viewed as simply a single step PGD attack.

In the previously described black-box setting class, probabilities or scores are available for *all* output classes of the classifier. Therefore, attacker can easily define a loss function and then maximize it. Since all possible output classes are known, generating targeted attacks is also straightforward. However, there can be a case where the attacker only has access to a *sub-set* of classes and a *confidence score* associated with each class in this subset. Since only a subset of output classes are known, introducing a target (adversarial) class is not straightforward as one does not even know what are the remaining classes. This setting is known as *partial-*

information setting. Google Cloud Vision (GCV) platform falls in this category. Our motivating example of successfully attacking GCV in chapter 1, shows that even in such setting, adversarial attacks are possible. For more details refer to Ilyas *et al.* [19].

We now discuss Transfer attacks or the phenonmenon of transferability [32]. In this setting the attacker does not have access to the actual weights or the class probabilities of the trained network. But instead, has access to the training data and the network architecture. The attacker may also not have complete information about the network architecture. Numerous studies [16] [32] have shown that adversarial examples generated using one model can also fool other independently trained models. However, adversarial training and network capacity can significantly reduce transferability [23] [16].

Another important aspect of a threat model is the space of allowed perturbations, i.e. the set $\mathcal{S}$. Most commonly, norm based perturbations around the input such as l_2 or l_∞ are considered. These perturbations although visually imperceptible, but are not semantically meaningful. For instance, recall from section 1.2, bypassing the HOV classifier on highways using inflatable dolls or cardboard cutouts would be one such real-world example.

Chapter 3

Error Correcting Codes based *Robust* Classification

In previous chapter, we discussed different methods of training a robust classifier. Most of these methods mainly approach to *directly* solve the min-max problem (1.2) by either explicitly solving the inner maximization problem using gradient based attacks such as PGD/FGSM or by upper-bounding it by forming its convex relaxation and then using duality. In this chapter, we introduce our approach of training a *robust* classifier using Error-Correcting Codes (ECCs).

Our goal is to train a classifier which achieves low adversarial loss, i.e we aim to do well on the inner problem of (1.2), but we do not directly optimize model parameters θ over the maximum loss(or its upper bound) by solving the inner problem. Instead, we rely on the error-correcting property of Error-Correcting Codes (ECCs). Therefore, corresponding to a given codebook, we train separate binary classifiers (or optimize their network weights θ's) and finally combine them into a single multi-class classifier. We first provide the motivation of using ECCs based on the adversarial robustness of binary classifiers, and we then introduce the idea of using error-correcting codes to compose multi-class classifiers from binary classifiers.

3.1 Robustness of binary Classifiers

The main motivation of our proposed approach arises from the observation that binary Deep Neural Networks (DNNs) exhibit high adversarial robustness. There can be two ways in which one can establish the *robust* of a classifier:

1. Adversarial accuracy for a given ε
2. The average maximum adversarial distortion for which the classifier predicts the correct class.

3.1.1 Adversarial Accuracy based comparison

On CIFAR10 dataset, Mądry et. al. [23] achieves an adversarial accuracy of around $45 - 50\%$ for $l_\infty = 8/255$. However, if we *adversarially* train all two class pairs on CIFAR10 dataset, i.e. $\binom{10}{2} = 45$ binary classifiers, the average adversarial accuracy of these binary classifiers is around 75%. Therefore a natural question arises: *Can we build a robust multi-class classification system composed of (robust) binary classifiers?* Given the high accuracy of these binary classifiers, we aim to construct a multi-class classifier by intelligently combining several robust binary classifiers (not necessarily two class pairs). Our final goal would be to outperform the multi-class system of Mądry et. al [23]. However, even if this is not possible, this new and interesting way of error-correcting codes based defense is in itself worthy of thorough investigation.

3.1.2 Adversarial distortion based comparison

Another way of evaluating the robustness of a model would be to find the maximum perturbation ε for which the classifier still predicts the correct class. Mathematically, this corresponds to the following optimization problem:
Let c be the correct class and $f_i(x)$ represent the output score of class $i \in \{1, \dots k\}$,

then:

$$\begin{aligned} \max_{x',\varepsilon} \ & \varepsilon \\ & f_c(x') > f_i(x'),\ \forall\ i \in \{1, \dots k\} \backslash \{c\} \\ & x - x' \leqslant \varepsilon \\ & x' - x \leqslant \varepsilon \\ & l \leqslant x' \leqslant u \end{aligned} \tag{3.1}$$

Solving the above optimization problem *optimally* may not possible due to non-convex nature of neural networks, instead we can try to form a convex-relaxation using methods enumerated in chapter 2 and then solve the relaxed version. For ReLU networks, the above problem will again simplify to a MILP. Solving the above problem for ReLU networks can be computationally expensive, even for smaller networks [31]. And since tightness of the convex-relaxed version would be heavily dependent on the size and the architecture of the network, therefore it may not provide us with an accurate way to compare with a multi-output network as used in Mądry *et al.* [23]. For these reasons, we mainly rely on adversarial accuracy based comparisons.

Before discussing the idea of using ECCs for classification, we would like to highlight one key observation made in Mądry et. al. [23]. Authors in [23] highlight that network capacity helps in achieving higher adversarial accuracy. For instance on CIFAR10 dataset, authors increase the width of the layers of their network by a factor of 10, and achieve higher adversarial accuracy against single-step FGSM attack and also against transfer-attacks. Apart from the error-correcting property of ECCs, this further provides support for our hypothesis of achieving robustness with ECCs. Combining the output of multiple classifiers, where each classifier solves some different classification problem, in a way increases the overall system (model) capacity and therefore can potentially have higher accuracy. Although, for our approach, we believe that this should provide only a second order effect on the overall performance.

3.2 Error-Correcting Codes (ECCs) for multi-class Classification

Consider that we are solving a k class classification problem, where $|k| > 2$ and our goal is to solve the k-class problem by only using binary classifiers. The two well known and commonly used methods of reducing multi-class problems to binary classification problem are one-vs-all (also known as one against all) and one-vs-one (also known as all-pairs).

In the one-vs-all case, k binary classifiers are trained, one for each class such that training examples from that class are considered positive $(+1)$ and training examples from rest of the $k-1$ classes are considered negative (-1). This system can be represented by a $k \times k$ matrix as shown in the figure 3-1a .

In the one-vs-one or all pairs method, $\binom{k}{2}$ binary classifiers are trained. For each distinct pair $(i, j) \in \{1, 2, \ldots, k\}^2 |\ i \neq j$, examples from class i are considered as positive $+1$ and examples from class j are considered negative -1 and all other examples are ignored. The examples ignored can be represented by 0. The final system can be represented by $k \times \binom{k}{2}$ matrix[1] as shown in the figure 3-1b .

At test time, a new example can be classified by majority voting, where final prediction of each hypothesis provides a vote for a particular class. The final predicted class is the one with the most votes (ties are broken at random).

	f1	f2	f3	f4
C1	+1	-1	-1	-1
C2	-1	+1	-1	-1
C3	-1	-1	+1	-1
C4	-1	-1	-1	+1

(a) one-vs-all

	f1	f2	f3	f4	f5	f6
C1	+1	+1	+1	0	0	0
C2	-1	0	0	+1	+1	0
C3	0	-1	0	-1	0	+1
C4	0	0	-1	0	-1	-1

(b) one-vs-one or all-pairs

Figure 3-1: Two different codebooks for a 4-Class problem. Image adapted from [25]

[1] From henceforth, we will use coding matrix and codebook interchangeably.

The above two schemes although intuitive are in fact two special cases of a generalized framework of Error Correcting Output Codes proposed by Dietterich and Bakiri [13]. In this paradigm, each class is encoded with a unique **codeword** or a string composed of length l resulting in a coding matrix $\mathcal{M}$ of size $k \times l$. The entries of the coding matrix M are either taken from the set $\{+1, -1\}$ or $\{+1, 0 - 1\}$, i.e. $M \in \{+1, -1\}^{k \times l}$ or $\mathcal{M} \in \{+1, 0, -1\}^{k \times l}$.

$$\text{Binary Coding} : \mathcal{M} \in \{+1, -1\}^{k \times l}$$
$$\text{Ternary Coding} : \mathcal{M} \in \{+1, 0, -1\}^{k \times l}$$

Each column in the matrix $\mathcal{M}$, represents a binary learning problem. Training examples belonging to different classes $C1 \ldots Ck$ with entries $\{\pm 1\}$ are partitioned into two classes where all $\{+1\}$ entry examples constitute the positive class and $\{-1\}$ examples constitute the other class. In case of ternary coding, training examples with entry 0 are not incorporated in the training set and are considered irrelevant. Let $f_1(x), \ldots, f_l(x)$ represent the l binary hypotheses learnt for each column of $\mathcal{M}$. For a new sample x, after evaluation on all l hypotheses we get an encoding $\vec{f}(x)$ of length l which we denote as :

$$\vec{f}(x) = [f_1(x), \ldots, f_l(x)]$$

For a binary deep neural network, let $f_{s_{+1}}(x)$ denote the output of the logit corresponding to class $+1$ and $f_{s_{-1}}(x)$ for class -1. Then,

$$f_s(x) = \begin{cases} +1 & \text{if } f_{s_{+1}}(x) > f_{s_{-1}}(x) \\ -1 & \text{otherwise} \end{cases} \qquad \forall \ s \ \in \{1, \ldots, l\} \tag{3.2}$$

Finally we need to associate $\vec{f}(x)$ with any one of the rows of the coding matrix $\mathcal{M}$ or correspondingly a class. This is commonly done by using **Hamming decoding**. Hamming distance $d_H(\cdot, \cdot)$ between $\vec{f}(x)$ and each row or codeword $\mathcal{M}(r, \cdot)$ is computed and the one with the minimum hamming distance d_H is chosen. Hamming distance, in case of **binary** vectors measure the number of places in which the

two vectors disagree. However, in case of **ternary** codes if $\mathcal{M}(r,s)$ is zero, $1/2$ is contributed to the sum for that component.

$$\hat{y} = \underset{r}{\operatorname{argmin}}\ d_H(\mathcal{M}(r,\cdot), \vec{f}(x))$$
$$d_H(\mathcal{M}(r,\cdot), \vec{f}(x)) = \sum_{s=1}^{l} \left(\frac{1 - \operatorname{sign}\big(\mathcal{M}(r,s) \times f_s(x)\big)}{2} \right)$$

$\operatorname{sign}(z) = +1$ if $z > 0, -1$ if $z < 0$ and 0 otherwise.

3.2.1 Binary Codes

For Binary codes, as mentioned before the coding matrix is $\mathcal{M} \in \{+1, -1\}^{k \times l}$. As the final accuracy of the classifier depends on the error-correcting ability of the coding matrix $\mathcal{M}$, it is important to choose a coding matrix carefully to have low test error. Mathematically, if each codeword or row of the coding matrix $\mathcal{M}$ has a hamming distance of at-least d with every other row, then such a code can correct *at-least* $\lfloor \frac{d}{2} \rfloor$ *errors*. Equivalently, the closed Hamming balls of radius $\lfloor \frac{d}{2} \rfloor$ around each codeword are disjoint.

Lets now consider a 3 class problem[2] for which we want to find a valid binary coding matrix $\mathcal{M}$. All the possible columns are shown in table 3.1. Note that columns f_5 to f_8 are simply complements of columns f_1 to f_4, therefore we can simply ignore them. Moreover, f_1 has all zeros, therefore does not represent a valid classifier. Finally, f_2, f_3, f_4 will constitute our coding matrix.

[2]We borrow this example from Dietterich and Bakiri [13].

Classes	Code words							
	f1	f2	f3	f4	f5	f6	f7	f8
C1	0	0	0	0	1	1	1	1
C2	0	0	1	1	1	1	0	0
C3	0	1	0	1	1	0	1	0

Table 3.1: Different binary hypothesis possible for a 3 class problem.

For a code of fixed length l, as highlighted in [13], it is desirable to choose a code with the following two properties:

Row Separation: Hamming distance between any two pair of codewords should be large or the two codewords should be well separated.

Column Separation: Every column h_i should be uncorrelated with all other columns $h_j, i \neq j$. This can be achieved by having large Hamming distance between columns as well. Note that maximum Hamming distance would be achieved if the two columns are complementary to each other, however this would result in essentially learning the same discriminant function, therefore, to avoid this, every column including its *complement* should be uncorrelated with all other columns.

Requirement for large row separation follows from the error-correcting property. A codebook with large hamming distance among its rows can correct for more errors, therefore in order to have a high prediction accuracy it is desirable have to large row separation. However, the need for large column separation is not so straightforward. To understand this we need to discuss an important assumption of Error-correcting codes from the perspective of communication over a noisy channel. The whole idea of *encoding* a signal and then transmitting the code over a noisy channel is useful only if the noise or the error made during transmission due to this noise is *random.* Then by having a sufficiently large *encoding* we can recover the original signal at the receiving end with very high accuracy.

For our case if the two columns or classifiers make errors in their predictions on the same inputs, i.e. their outputs are correlated then such columns will defeat

the purpose of encoding. Therefore, in order to avoid correlated hypotheses, it is important to have large column separation. Also note that since maximum column separation is achieved for complementary columns, which correspond to the same classification problem or hypothesis, therefore, column separation should not be too high as well. Because if the column separation is too high then a column may be correlated to the *complement* of the other column. In subsequent section 3.2.3, we will formalize these concepts mathematically to formulate a discrete optimization problem to generate codebooks.

3.2.2 Ternary Codes

For Ternary codes, the elements of the coding matrix are chosen from a larger set $\{-1, 0, +1\}$, or $\mathcal{M} \in \{-1, 0, +1\}^{k \times l}$. One-vs-one or all-pairs codebook is one example of ternary codes, see figure 3-1b for a 4-class problem. We will discuss more about ternary codes in section 3.2.4 under tree based classifiers.

3.2.3 Codebook generation

In our discussion so far, we have provided an overview of how EECs can be used for classification including different categories of codes such binary and ternary codes. Now the question arises: *How to select a particular codebook for which the resulting k-class classifier has high accuracy (both nominal and adversarial)?*
For the discussion of codebook generation in this section, we limit ourselves to binary codes. Following the guidelines of Dietterich and Bakiri in [13], we formulate a discrete optimization problem and solve it using Integer Programming (IP) solvers to generate a codebook.

Recall our 3 class example from section 3.2.1 where in table 3.1 we highlight all the possible columns which are part of the final codebook. This enumeration of columns can be easily generalized to k classes, and the final coding matrix will have $(2^k - 2)/2 = 2^{k-1} - 1$ columns. This can be categorized as *Exhaustive Coding* [13]. For $2 < k < 7$, using an exhaustive code can be feasible as the number of binary

classifiers needed are small, however for large k the number of columns required grow exponentially. For a 10 class problem, there would be 511 valid columns. Therefore, it is necessary to select only a subset of columns while *maintaining high row and column separation.*

The column subset selection can be either formulated as a propositional satisfiability problem and then solved using an off-the-shelf SAT solver, or equivalently it can be formulated and solved as an Integer Programming problem. The SAT formulation for $8 \leqslant k \leqslant 11$ proposed in [13], aims to attempt a solution to the following problem: For a predefined number of columns L and some value d, is there a solution such that the Hamming distance between any two columns is between d and $L - d$? Instead, we modify this to find a solution which maximizes the minimum Hamming distance between any two rows. Let $x_i, i \in 1, \ldots 2^{k-1} - 1$ be a boolean variable associated with each column of the exhaustive code, and let x_{ij} be the boolean variable which represents the outcome of AND operation between variables x_i and x_j for all $(i, j) \in \{1, \ldots, 2^{k-1} - 1\}^2 | i \neq j$. Basically, the variable $x_{ij} = 1$ ensures that columns i and j in the final solution satisfy the column separation criteria. We now provide the IP formulation to generate a codebook.

$$\max \min \ \{d_H^{1,2},\ d_H^{1,3},\ \ldots,\ d_H^{k-1,k}\} \tag{3.3}$$

$$\sum_{i=1}^{2^{k-1}-1} x_i \leqslant L \tag{3.4}$$

$$d\, x_{ij} \leqslant d_H\big(\mathcal{M}(\cdot,i), \mathcal{M}(\cdot,j)\big)\, x_{ij} \leqslant (L-d)\, x_{ij} \quad \forall\ (i,j) \in \{1,\ldots,2^{k-1}-1\}^2 |\ i \neq j \tag{3.5}$$

$$x_{ij} \leqslant x_i \tag{3.6}$$

$$x_{ij} \leqslant x_j \tag{3.7}$$

$$x_i + x_j - 1 \leqslant x_{ij} \tag{3.8}$$

$$d_H^{s,t} = \sum_{i=1}^{2^{k-1}-1} \Big(\frac{1 - sign(\mathcal{M}(s,\cdot), \mathcal{M}(t,\cdot))}{2}\Big) x_i \qquad \forall\ (s,t) \in \{1,\ldots,k\}^2 |\ s \neq t \tag{3.9}$$

$$x_i \in \{0,1\} \qquad \forall\ i \in \{1,\ldots,2^{k-1}-1\} \tag{3.10}$$

$$x_{ij} \in \{0,1\} \qquad \forall\ (i,j) \in \{1,\ldots,2^{k-1}-1\}^2 |\ i \neq j \tag{3.11}$$

In the above formulation, max-min can be easily simplified by introducing an auxiliary variable t, where $t = \min\ \{d_H^{1,2},\ d_H^{1,3},\ \ldots,\ d_H^{k-1,k}\}$ and corresponding constraints $t \leqslant d_H^{1,2}$; $t \leqslant d_H^{1,3}$; $\ldots$ $t \leqslant d_H^{k-1,k}$. Eq. (3.5) ensures large column separation if $x_{ij} = 1$. Equations (3.6) and (3.7) ensure that if $x_{ij} = 1$ then both columns i and j are included in the solution, i.e. $x_i = 1$ and $x_j = 1$. Conversely, Equation (3.8) ensures that if columns i and j are selected then $x_{ij} = 1$.

Some discussion regarding the dimensionality of the above problem:

There are $2^{k-1} - 1 \approx \mathcal{O}(2^{k-1})$ binary variables for each column and for each pair of columns there are $\binom{2^{k-1}-1}{2} \approx \mathcal{O}(2^{2k-3})$ binary variables. Therefore, the total number of binary variables are of the order of $\mathcal{O}(2^{2k-3})$. The total number of constraints are of the order of $\mathcal{O}(2^{2k-1})$. For a 10 class problem, i.e. $k = 10$, there would be around $130,000$ variables and $650,000$ constraints. Modern, IP solvers like Gurobi and CPLEX can handle such large problems, however for $k > 10$, solving the above optimization problem may not be tractable. Note that Dietterich and Bakiri in [13], only aims to find a feasible solution using a SMT solver but not the optimal solution. This is reasonable and can be attributed to the fact that their study was done almost

two and a half decades ago. Since then IP solvers have made significant advances.

The above proposed IP, although easy to follow, becomes intractable for $k > 10$. The main reason is that we introduced a binary variable for every pair of columns to ensure that in the final solution eq. (3.5) (corresponding to large column separation) is satisfied. We now propose an alternative formulation in which we do not need to introduce variable x_{ij} for every pair of columns. Let $\mathcal{G}_p$ represents the set of all pairs of columns, i.e. $\mathcal{G}_p = \{(i,j) \in \{1,\ldots,2^{k-1}-1\}^2 |\ i \neq j\}$ and $|\mathcal{G}_p| = \binom{2^{k-1}-1}{2}$. We can now divide the set $\mathcal{G}_p$ into two disjoint subsets $\mathcal{G}_p^{feas}$ and $\mathcal{G}_p^{inf}$, such that $\mathcal{G}_p = \{\mathcal{G}_p^{feas}, \mathcal{G}_p^{inf}\}$. The set $\mathcal{G}_p^{feas}$ contains only those i,j pairs which *satisfy* the column separation criteria in eq. (3.5) and set $\mathcal{G}_p^{inf}$ contains the i,j pairs which *do not* satisfy the column separation criteria.

$$
\begin{aligned}
\mathcal{G}_p &= \Big\{(i,j) \in \{1,\ldots,2^{k-1}-1\}^2 |\ i \neq j\Big\} \\
\mathcal{G}_p^{feas} &= \Big\{(i,j) \in \{1,\ldots,2^{k-1}-1\}^2 |\ i \neq j; d \leqslant d_H\big(\mathcal{M}(\cdot,i), \mathcal{M}(\cdot,j)\big)\ \leqslant (L-d)\Big\} \\
\mathcal{G}_p^{inf} &= \mathcal{G}_p \backslash \{\mathcal{G}_p^{feas}\}
\end{aligned}
$$

The interesting thing to note is that for any pair $i,j \in \mathcal{G}_p^{feas}$ we do not need impose the column separation constraint (eqn 3.8) as this we have already ensured while constructing the set $\mathcal{G}_p^{feas}$. Therefore, we do not need to introduce variable x_{ij} for these pairs, and hence correspondingly we also do not need to include equations (3.6), (3.7) and (3.8). Now lets consider the i,j pairs in set G_p^{inf}. We know that these pairs do not satisfy the column separation criteria, therefore for any pair *at-most* only one of the two columns can be included in the final solution. For these pairs we know the variable x_{ij} has to be 0 in any feasible solution. For these pairs since we only need to ensure that at-most one column is selected, this can be achieved by setting $x_{ij} = 0$ in eq. (3.8). Finally, for every i,j pair in G_p^{inf}, we only need to include one constraint: $x_i + x_j - 1 \leqslant 0$ for every pair.

The final IP reduces to the following form:

$$\max \min \{ d_H^{1,2},\ d_H^{1,3},\ \ldots,\ d_H^{k-1,k} \} \tag{3.12}$$

$$\sum_{i=1}^{2^{k-1}-1} x_i \leqslant L \tag{3.13}$$

$$x_i + x_j \leqslant 1 \qquad \forall\ (i,j) \in \mathcal{G}_p^{inf} \tag{3.14}$$

$$d_H^{s,t} = \sum_{i=1}^{2^{k-1}-1} \Big(\frac{1 - sign(\mathcal{M}(s,\cdot), \mathcal{M}(t,\cdot))}{2} \Big) x_i \qquad \forall\ (s,t) \in \{1,\ldots,k\}^2 |\ s \neq t \tag{3.15}$$

$$x_i \in \{0,1\} \qquad \forall\ i \in \{1,\ldots,2^{k-1}-1\} \tag{3.16}$$

The above formulation does not contain any x_{ij} term. This significantly reduces the number of variables along with the number of constraints in the previous formulation. The total number of constraints will now mainly be decided by the size of the set $\mathcal{G}_p^{inf}$, corresponding to eq. (3.14). We would like to point out that in case if the IP solver does not terminate for larger k, i.e. $k > 11$, it will still provide us with a feasible solution and moreover it will also provide us with an upper-bound on our objective function value. This will provide us with some idea on how far we are from an optimal solution if a feasible solution with the same value as that of an upper-bound exists. This is another major benefit of our IP based approach compared to the SMT based approach of Dietterich and Bakiri [13].

We now discuss some aspects of the codebook design which the above formulation does not take into account. The above IP based method of codebook generation for binary codes tries to ensure high row and column separation, however these are simply guidelines to design codebooks which could potentially result in good classifiers. This procedure does not take into account the hardness of the resulting hypotheses or the prediction accuracy of the individual hypotheses. Therefore, it cannot ensure *optimality* of the resulting multi-class classifier. For instance, a classifier resulting from a sub-optimal solution (or codebook) to the above IP could have higher final accuracy in comparison to the optimal solution (codebook). Ideally one would like to formulate the problem of codebook design as a learning algorithm which incorporates

the final accuracy of the resulting classifier and hence can iteratively improve upon it. The final codebook should be decided from the data (or the training set) and should not be simply *pre-defined.*

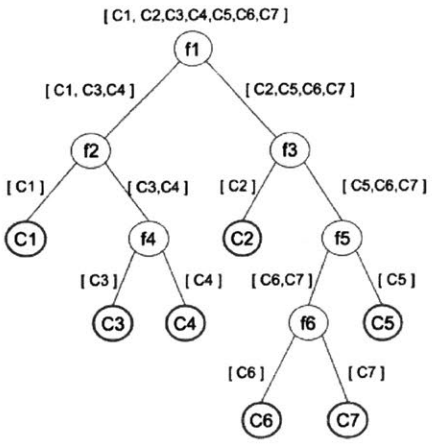
Crammer and Singer [12] showed that the problem of *optimal* discrete codebook design is intractable and NP-complete. Instead of designing discrete codes, they made progress in the design of continuous codes. Pujol *et al.* in [25], proposed a method based on hierarchical partition of the classes resulting in a binary tree or a ternary code book. We will discuss tree based classifiers in section 3.2.4. Martin *et al.* in [5], recently proposed a new method called Error Correcting Factorization. The problem of *optimal* codebook design is still an open problem.

We note that *random codes* is another way of arbitrarily generating codebooks as done in Allwein *et al.* [4] for the purpose of benchmarking. If the elements are chosen uniformly at random from $\{+1, -1\}$, then the resulting codebooks are called *dense codes* and if the elements are taken from $\{-1, 0, +1\}$ then the resulting codebooks are called *sparse codes.* In sparse codes, 0 is chosen with probability 1/2 and ± 1 are each chosen with probability 1/4. Generally, $10,000$ random matrices are generated and after discarding the invalid matrices, the one with the largest minimum Hamming distance among rows is selected. Allwein *et al.* [4] used codebooks of length $\lceil 10 \log_2 k \rceil$ and $\lceil 15 \log_2 k \rceil$ for dense codes and sparse codes respectively.

3.2.4 Tree based Classifiers

Tree based classifiers are another way to reduce a k class problem to different binary (or smaller class) classification problems. Each node of the tree is a binary (or ternary) classification problem. Starting at the root node, one moves down the tree based on the outcome at each node untill a leaf node is reached.

A binary tree based classifier can be represented as a ternary code [25]. This can be easily verified from the schematic shown in figure 3-2. However, note that every ternary code may not have a corresponding binary tree structure, one-vs-one would be one such example.



	f1	f2	f3	f4	f5	f6
C1	+1	+1	0	0	0	0
C2	-1	0	+1	0	0	0
C3	+1	-1	0	+1	0	0
C4	+1	-1	0	-1	0	0
C5	-1	0	-1	0	-1	0
C6	-1	0	-1	0	+1	+1
C7	-1	0	-1	0	+1	-1

Figure 3-2: A binary Tree and its corresponding coding matrix

We briefly overview the method of Pujol *et al.* in [25] as their method (known as DECOC [3]) is relevant to our discussion on codebook generation in previous section 3.2.3. Authors in [25] proposed a method of generating a ternary codebook which involves using training data to decide the columns of the final codebook, thus making progress towards optimal codebook design. Using the fact that any k-class problem can be reduced to a binary tree as shown in figure 3-2 and every such binary tree corresponds to a ternary codebook, authors in [25], tries to generate one such binary tree. Note that each node of the binary tree in figure 3-2 corresponds to a column in the codebook. Also, any such binary tree will have $k-1$ columns corresponding to its internal nodes. At each parent node, starting at the root node, authors try to find the *partition* of the classes (into two sub-sets) which has the maximum discriminative power. Finding the best partition at each node requires searching through an exponential number of candidate partitions. To deal with this, authors resort to Floating search methods [24], with Mutual Information as the criteria to decide on the quality of a partition. Finally, for a new example at test time, authors used Euclidean distance instead of Hamming distance for decoding to assign a particular class to that example.

In the above outlined DECOC approach, in contrast to our IP based approach, there is no emphasis on the error-correcting capability, therefore, in some sense the

[3]DECOC: **D**iscriminant **E**rror **C**orrecting **O**utput **C**ode

DECOC approach trades error-correcting capability with classifier performance for each hypothesis. In terms of experimental results, Pujol *et al.* [25] compare their approach on nine datasets from the UCI database and on traffic sign dataset. Authors used Adaboost and Decision Tree as the classifier for each hypothesis. Authors benchmark their approach against predefined codebooks such as one-vs-one, one-vs-all and randomly generated sparse and dense codes. On the UCI datasets, DECOC although outperforms all other predefined codebooks, however, the gain in performance over one-vs-one is not very high. One-vs-all performs poorly and is easily outperformed by randomly generated dense and sparse codes. This clearly indicates that one-vs-one although a predefined codebook still provides high accuracy therefore a good candidate to benchmark against our own codebooks generated via the IP based approach in chapter 5. On the traffic sign dataset with 32 classes, DECOC outperforms one-vs-one by around $3-4\%$ with only using 31 hypotheses in comparison to the 496 hypothesis used by one-vs-one. This shows that it is possible to find a compact codebook with high prediction accuracy.

3.3 ECC based *Robust* Classifiers

In this section we put everything together, mainly extending the discussion of section 3.1 and 3.2 to form a k-class classifier using ECCs, however now we finally focus on the *adversarial robustness* of the final classifier. The two main components of our ECC based approach are, first solving the IP to generate a codebook and then training a classifier for every column of the pre-defined codebook. In all our subsequent discussion including results in chapter 5, we will adhere to this approach.

Ideally we would like to achieve robustness by only training each of the classifiers nominally, and hoping that error-correcting property of the codebook along with the inherent robustness of binary classifiers will provide resistance against adversarial attacks. We will put this hypothesis to test in our experiements in chapter 5. Although, this maybe a lot to ask given the issue of error-correlation among resulting hypotheses which plagues most codebooks in *nominal* setting as highlighted by Dietterich

and Bakiri in [13]. It will be interesing to see if the same issue persists in adversarial setting as well. However, a natural extension would be to robustify each of the hypothesis using any one of the methods outlined earlier in chapter 2. We will also test this in chapter 5 using PGD based adversarial training for each of the column hypothesis.

An important aspect, which we have not discussed in this chapter is that of evaluating the robustness of ECC based classifers. From our discussion in chapter 2, this mainly depends on the threat model and the strength of the attack. In chapter 4, we will discuss about this aspect in detail.

Chapter 4

Adversarial Evaluation of ECC based Classifiers

In previous chapter, we proposed a method to train a (robust) multi-class classifier using Error Correcting Codes (ECCs). In this chapter, we propose methods to evaluate the adversarial accuracy of different ECCs based classifiers resulting from various codebooks. We first highlight the challenges associated in evaluating the adversarial accuracy of these classifiers for different threat models which we discussed in chapter 2. We also provide solutions to these challenges by building upon existing literature.

The primary goal of this thesis is to build a classifier that is robust against all allowed perturbations (or attacks) by an adversary. Therefore, for a given input $x + \delta$ and allowed set of perturbations, i.e. $\delta \in \mathcal{S}$, we need to determine the adversarial accuracy of the trained classifier. It is pertinent to evaluate against the strongest possible adversary, as a false sense of robustness can lead to major security related incidents as already outlined in chapter 1. There is a growing body of literature for both defense and attacks almost reminiscent of an arms race. Most importantly, it is often easy to overestimate the adversarial accuracy of a classifier or a defense method. Athalye *et al.* in [6] circumvented several defense methods which relied on gradient obfuscation.

In this chapter we mainly focus on black-box and white-box attacks as other attacks under different threat models can be derived from these two type of attacks.

We aim to evaluate the robustness of ECC based classifiers against state-of-the art attacks or strongest possible adversary in both black-box and white-box setting. We particularly aim to do a rigorous evaluation in white-box setting as it provides a upper bound (potentially not too loose) on the true adversarial accuracy of the classifier. In the space of white-box attacks, gradient based PGD-attack proposed by Mądry *et al.* in [23] has emerged as one of the strongest attack. The success and wide adoption of PGD-attack, apart from being a strong adversary, can also be attributed to the fact that most of the modern deep learning models are easily differentiable. With the advent of modern GPUs and frameworks like pyTorch and Tensorflow which support automatic differentiation, computation of gradients wrt to model parameters w and input x is extremely efficient.

The main requirements for both black-box and white-box attacks can be reduced to the following:

1. Class probability estimates to compute the loss function.

2. Gradients of the loss-function with respect to (w.r.t.) the input x.

For ECC based classifiers computation of both, the class probability estimates and the gradients is not straightforward. This is due to the discrete nature of decoding (such as Hamming decoding) scheme involved when deciding on the final class. Discrete encoding of classes to make use of the error-correcting property comes at the cost of losing differentiability. However, in subsequent discussion we show that we can come up with alternative techniques to generate strong attacks while still enjoying the benefits of encoding.

We first try to identify the issues associated with Hamming decoding. Two main issues to note are:

1. It does not provide us with class probability estimates.

2. It does not take into account the magnitude of the predictions of the individual hypothesis which in many cases correspond to a "confidences-score".

Both of the above issues need to addressed, however the requirement of class probability estimates (or class scores) is absolutely necessary for running black-box attacks. The second issue is important as it can significantly improve the final accuracy (both nominal and adversarial) of the classifier [4].

4.1 Class Probability Estimates

In chapter 3, we described Hamming decoding as a way to associate a final prediction class with the output vector obtained after passing the input x through each of the individual hypothesis. This procedure does not provide us with class membership probability estimates. Apart from our need to run black-box attacks, class probability estimates can be useful in many applications areas where the final classification rule is designed based on these estimates. Neyman Pearson classifier would be one such example. In anomaly detection or automated monitoring of critical infrastructure, even a low probability or indication of disturbance warrants manual inspection. Generally in these situations cost of missed detection is very high compared to false alarm. Recall our car-ladder adversarial example from chapter 1. For a parking-lot operator or a traffic monitoring agency, cost of missed-detection may lead to financial losses or may compromise the security of a facility under surveillance.

Hastie and Tibshirani in [17] proposed a method to estimate class membership probabilities for all-pairs. This was later generalized by Zadrozny [34] for arbitrary codebooks. Hastie and Tibshirani's motivation in [17] was twofold:

1. To solve the problem of getting calibrated estimates for all-pairs codebook in itself.
2. To use these estimates to make predictions in order to achieve better classification accuracy over the majority voting rule.

Our motivation of getting probability estimates is driven by the need to compute a loss function from these estimates to generate an attack. With these estimates we can easily run different black-box attacks. Moreover if it turns out that we can

differentiate this loss function[1] with respect to x, then we can also extend this to gradient-based white-box attacks.

It is important to note that we don't aim for highly calibrated estimates. Instead we are simply interested in getting a crude estimate of the loss function. For codebooks like all-pairs and one-vs-rest, we can do something quite intuitive and simple. For all-pairs, we can simply add the output of each hypotheses to its corresponding class to get a score for each class. We can then use these scores to compute a loss function. Importantly, we maintain differentiability through each operation. Figure 4-1 shows the whole procedure.

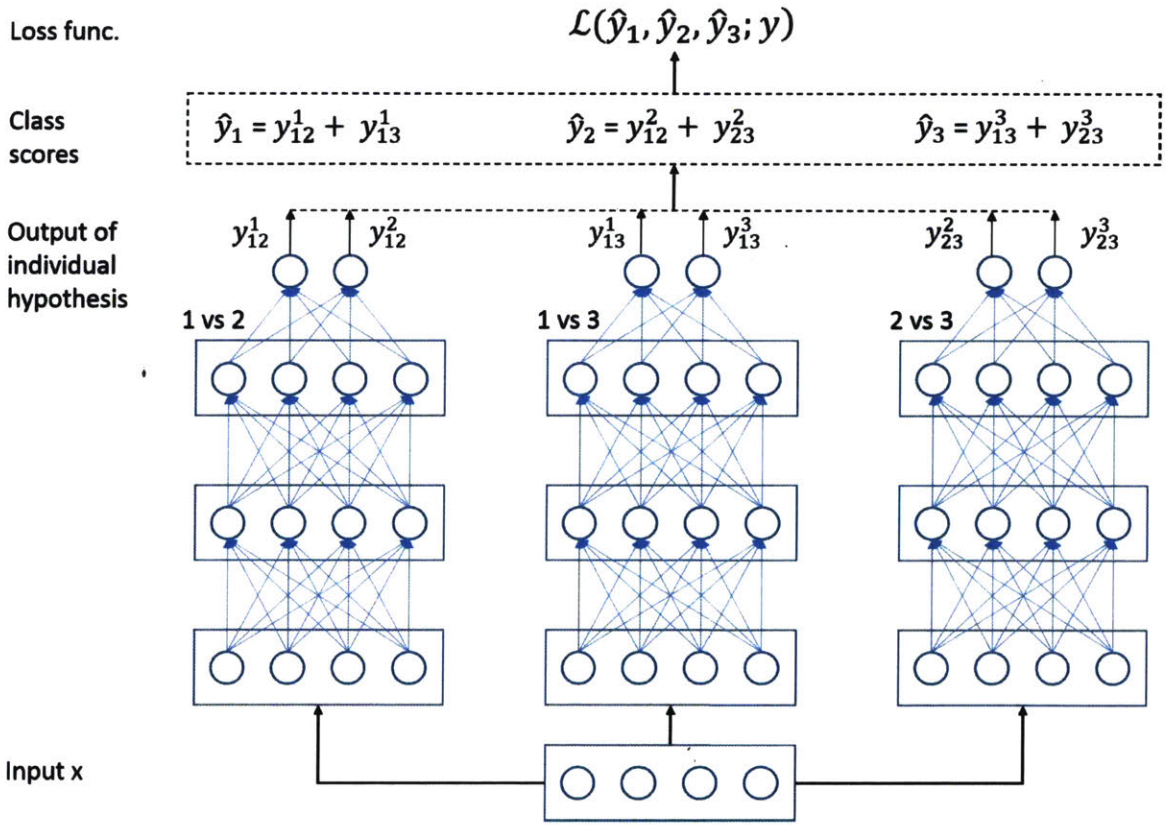


Figure 4-1: Combining output of individual hypotheses of all-pairs to generate class scores while maintaing differentiability.

[1] It might be the case that in the process of calculating probability estimates one may loose differentiability even before computing the loss function from these estimates

We now describe the method of Hastie and Tibshirani in [17] to compute estimates for all-pairs. Let $r_{ij}(x)$ be the output of the classifier trained over examples from class i as positives and class j as negatives. Mathematically we can write class membership probabilities $p_i(x) = P(C = c_i|X = x)$ in terms of $r_{ij}(x)$ as:

$$r_{ij}(x) = P(C = c_i|C = i \vee C = j, X = x) = \frac{p_i(x)}{p_i(x) + p_j(x)} \tag{4.1}$$

As we have $k(k-1)/2$ pairs, correspondingly we will have $k(k-1)/2$ equations of the above form. Additionally we will have another constraint of $\sum_{i=1}^{k} p_i(x) = 1$. To simplify notation, we drop x and rewrite the equations as:

$$r_{ij} = \frac{p_i}{p_i + p_j} \quad \forall \ \ i, j \ \ \text{pairs where} \ \ i \neq j \tag{4.2}$$

$$\sum_{i=1}^{k} p_i = 1 \tag{4.3}$$

We need to solve the above set of of equations where r_{ij}'s are known constants and p_i's are the unknown variables. Using the summation in eq. (4.3) we can reduce the number of variable to $k-1$ and the final number of constraints to $k(k-1)/2$. However, for such a system of equations there may not exist a feasible solution. To solve this problem, Hastie and Tibshirani proposed fitting the Bradley-Terry model [8] for pairwise comparisons by minimizing the weighted KL-divergence between r_{ij} and $\hat{r}_{ij}$. Let n_{ij} denote the total number of training examples in class i and j. The KL divergence can be written as:

$$KL = \sum_{i \neq j} n_{ij} \left(r_{ij} log\Big(\frac{r_{ij}}{\hat{r}_{ij}}\Big) + (1 - r_{ij}) log\Big(\frac{1 - r_{ij}}{1 - \hat{r}_{ij}}\Big) \right) \tag{4.4}$$

To now minimize the above (eq. (4.4)) while maintaining consistency between $\hat{r}_{ij}$ and $\hat{p}_i$, Hastie and Tibshirani proposed the following algorithm:

Algorithm 1 Pairwise Coupling for all-pairs (Hastie and Tibshirani [17])

1: Initialize with some random guess for $\hat{p}_i$ and compute the corresponding $\hat{r}_{ij}$
2: **for** $t = 1, 2, \ldots T$ **do**
3: **for** $i = 1, 2, \ldots k$ **do**
4: $\hat{p}_i \leftarrow \hat{p}_i \frac{\sum_{j \neq i} n_{ij} r_{ij}}{\sum_{j \neq i} n_{ij} \hat{r}_{ij}}$
5: **end for**
6: $\hat{p}_i \leftarrow \frac{\hat{p}_i}{\sum_{i=1}^{k} \hat{p}_i}$ Re-normalize $\hat{p}_i$
7: $\hat{r}_{ij} \leftarrow \frac{\hat{p}_i}{\hat{p}_i + \hat{p}_j}$ Recompute $\hat{r}_{ij}$
8: **end for**

We now describe the generalized version of the above algorithm was proposed by Zadrozny in [34] for arbitrary code books. For an arbitrary codebook $\mathcal{M}$, we have an estimate $r_b(x)$ for each column b of $\mathcal{M}$. Let I denote the set of classes for which $\mathcal{M}(\cdot, b) = 1$ and J denote the set of classes for which $\mathcal{M}(\cdot, b) = -1$. We can now write :

$$r_b(x) = P(\bigvee_{c \in I} C = c | \bigvee_{c \in I \cup J} C = c, X = x) = \frac{\sum_{c \in I} p_c(x)}{\sum_{c \in I \cup J} p_c(x)}$$

We have l constraints for each column of the above form along with the $\sum_{i=1}^{k} p_i(x) = 1$ constraint. We drop x to simplify notation and then rewrite the system of equations as :

$$r_b = \frac{\sum_{c \in I} p_c}{\sum_{c \in I \cup J} p_c} \quad \forall \; b \in \{1, 2, \ldots l\} \tag{4.5}$$

$$\sum_{i=1}^{k} p_i = 1 \tag{4.6}$$

From eq. (4.5) & (4.6), we have $l + 1$ set of equations and k variables. Our aim is to estimate unknown p_i's such that the above system of equations is satisfied. Generally, for any codebook, the total number of columns l is larger than the number of classes k, hence there may not exist a feasible solution to (4.5) and (4.6). Therefore, analogous to Hastie-Tibshirani, Zadrozny in [34] proposed minimizing the

KL divergence between r_b and $\hat{r}_b$. Let n_b denote the number of training examples on which each column hypothesis is trained. Zadrozny proposed the following *iterative* algorithm:

Algorithm 2 Pairwise Coupling for arbitrary codebooks (Zadrozny [34])

1: Initialize with some random guess for $\hat{p}_i$ and compute the corresponding $\hat{r}_b$
2: **for** $t = 1, 2, \ldots T$ **do**
3: **for** $i = 1, 2, \ldots k$ **do**
4: $\hat{p}_i \leftarrow \hat{p}_i \frac{\sum_{b\ s.t.\ M(i,b)=1} n_b r_b + \sum_{b\ s.t.\ M(i,b)=-1} n_b(1-r_b)}{\sum_{b\ s.t.\ M(i,b)=1} n_b \hat{r}_b + \sum_{b\ s.t.\ M(i,b)=-1} n_b(1-\hat{r}_b)}$
5: **end for**
6: $\hat{p}_i \leftarrow \frac{\hat{p}_i}{\sum_{i=1}^{k} \hat{p}_i}$ Re-normalize $\hat{p}_i$
7: $\hat{r}_b \leftarrow \frac{\sum_{c \in I} \hat{p}_c}{\sum_{c \in I \cup J} \hat{p}_c}$ Recompute $\hat{r}_b$
8: **end for**

For black-box attacks we can compute the estimates using the above described algorithms, however because of the iterative nature of these estimates differentiability is not maintained. Therefore, we cannot use them to run white-box attacks. For white-box attacks we can resort to *non-iterative* estimates proposed by Zadrozny. The non-iterative estimates are given as:

$$\hat{p}_i^* = \sum_{b\ s.t.\ M(i,b)=1} r_b + \sum_{b\ s.t.\ M(i,b)=-1} (1 - r_b) \tag{4.7}$$

For all-pairs codebook, the non-iterative estimates (roughly) corresponds to the same estimates which we obtain previously shown in figure 4-1. Equation (4.7) extends this to arbitrary codebooks.

4.2 Differentiable decoding methods

In the previous section we tried to get around the problem of non-differentiability of Hamming decoding by estimating class probabilities. Using non-iterative probability

estimates we can even run white-box attacks. For the sake of completeness, in this section we discuss alternative methods to generate white-box attacks by substituting Hamming decoding with some other *differentiable* decoding methods. We mainly discuss loss-based and L_1/L_2 decoding.

4.2.1 Loss-based decoding

Loss-based decoding was proposed by Allwein *et al.* in [4], to take into account the loss function and the magnitude of predictions of individual hypotheses when making the final predictions. The magnitude of predictions of individual hypotheses often corresponds to a "confidence-score". Hamming decoding does not take into account this information, i.e. the "magnitude" of predictions but instead simply rely on the sign of predictions.

Allwein *et al.* [4] mainly worked with margin-based classifiers and showed that loss-based decoding is superior to Hamming decoding in terms of bounds on the training error. A margin-based binary classifier is a real valued function or hypothesis $f(x) : \mathcal{X} \mapsto R$. It is trained on examples $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in \mathcal{X}$ and $y_i \in \{+1, -1\}$. Generally the quantity $yf(x)$ is referred to as the *margin* of the classifier for an input x. The training error can be written as 0-1 loss function:

$$\frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{-y_i f(x_i) \geqslant 0\} \tag{4.8}$$

where $\mathbb{1}\{\}$ is the indicator function. Minimizing the above 0-1 loss function or the training error is hard due to the discrete nature of the function, therefore usually some other non-negative smooth and continuous loss function is used. Some commonly used loss functions for binary classification which can be written in terms of the margin of the classifier, $z = yf(x)$ are:

Hinge loss: $(1 - z)_+$

Logistic loss: $\log(1 + e^{-z})$

Exponential loss: e^{-z}

Square loss[2] : $(1-z)^2$

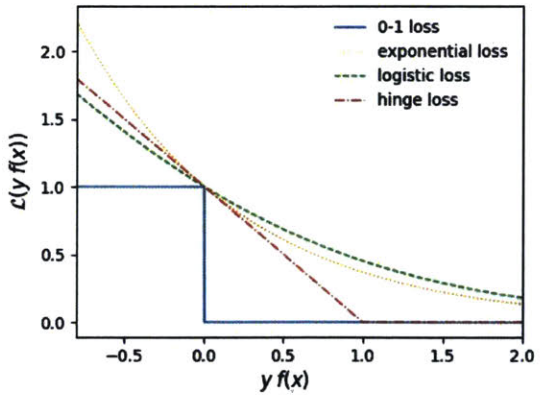


Figure 4-2: Different margin-based loss functions

The main requirement of loss-based decoding is that for a given input x, the output of each of the binary classifiers should represent a score, i.e. for positive class examples the classifier should output positive values and for negative class examples the classifier should output negative values. However, as pointed out by Zadrozny in [34], if the binary classifier outputs values which represent probability estimates $r_b(x)$, then these can be easily converted to margin scores by subtracting 1/2 from these estimates.

Mathematically, loss-based decoding can be represented as:

$$\hat{y} = \underset{r}{\operatorname{argmin}} \; d_L\big(\mathcal{M}(r,\cdot), \vec{\boldsymbol{f}}(x)\big)$$

$$d_L\big(\mathcal{M}(r,\cdot), \vec{\boldsymbol{f}}(x)\big) = \sum_{b=1}^{l} \mathcal{L}\big(\mathcal{M}(r,b)\boldsymbol{f_b}(x)\big)$$

where $\vec{\boldsymbol{f}}(x)$ represents a real-valued vector of length l. Each element of this vector $\boldsymbol{f_b}(x) \in \mathbb{R}$ represents the output of the binary classifier. $\mathcal{L}(\cdot)$ is the non-negative loss function, for example $\mathcal{L}(z) = e^{-z}$, see figure 4-2 for different loss functions. To maintain consistency of notation, a small recap. In chapter 3 while introducing Hamming decoding, $\vec{f}(x)$ denoted a binary-valued vector of length l, i.e. $\vec{f}(x) \in \{+1,-1\}^l$ and each element $f_b(x)$ of the vector $\vec{f}(x)$, represented the class label, i.e.

[2] As $y \in \{+1,-1\}$, therefore $(y-f(x))^2 = y^2(y-f(x))^2 = (y^2 - yf(x))^2 = (1-z)^2$

$+1$ or -1 for the b^{th} hypothesis.

Most importantly, we observe that this alternate method of decoding is differentiable and therefore can be used to generate white-box attacks. Moreover, it could be the case that loss-based decoding empirically provides better prediction accuracy than Hamming-decoding, then we can even replace Hamming decoding altogether.

4.2.2 L_1/L_2 decoding

Dietterich and Bakiri in [13] proposed using L_1 decoding for classifiers which output class probability estimates and have been trained on examples $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in \mathcal{X}$ and $\mathbf{y_i} \in \{\mathbf{0}, \mathbf{1}\}$, where $\mathbf{1}$ represents positive class examples and $\mathbf{0}$ represents negative class examples. Since they only worked with binary codes, therefore instead of using $\{-1, +1\}$ encoding, they used $\{0, 1\}$ encoding. They define L_1 decoding as:

$$\hat{y} = \underset{r}{\operatorname{argmin}} \; L_1\big(\mathcal{M}(r, \cdot), \vec{\boldsymbol{f}}(x)\big)$$

$$L_1\big(\mathcal{M}(r, \cdot), \vec{\boldsymbol{f}}(x)\big) = \sum_{b=1}^{l} \big| \underbrace{\mathcal{M}(r, b)}_{\{0,1\}} - \underbrace{\boldsymbol{f_b}(x)}_{[0\ 1]} \big|$$

We now show that this L_1-decoding is equivalent to loss-based decoding with hinge loss as the loss function. We can convert probability estimates to scores by subtracting $1/2$, i.e. $\tilde{f}(x) = f(x) - 1/2$. Therefore, with our usual encoding of $\{-1, +1\}$ for binary codebooks, we have:

$$\begin{aligned} L_1 &= |y - \tilde{f}(x)| \\ &= |y(y - \tilde{f}(x))| \quad \text{as } y \in \{+1, -1\} \\ &= |y^2 - y\tilde{f}(x)| \\ &= |1 - y\tilde{f}(x)| \\ &= |1 - z| \end{aligned}$$

Also, as $0 \leqslant f(x) \leqslant 1 \implies -1/2 \leqslant \tilde{f}(x) \leqslant 1/2 \implies -1/2 \leqslant \overbrace{y\tilde{f}(x)}^{z} \leqslant 1/2$

Therefore $L_1(z) = |1 - z| = |1 - z|_+$ when $-1/2 \leqslant z \leqslant 1/2$, i.e. same as hinge loss.

Similarly, one can also use L_2 norm instead of L_1 norm. Recall our discussion from section 3.2.4, the DECOC approach of learning ternary codebooks proposed by Pujol et. al. [25]. In [25], authors use L_2-norm or Euclidean decoding instead of Hamming decoding, highlighting the fact that at the time of decoding, zero entries in ternary codebooks can cause ties due to equal Hamming distances for multiple classes. To avoid this, authors preferred Euclidean decoding. Also, squared L_2 norm based decoding is equivalent to loss-based decoding with squared loss as the loss function.

Chapter 5

Experiments

In this chapter we finally train and evaluate the robustness of ECCs based classifiers. This can be divided into three main steps:
1) Generation of a codebook by solving the IP proposed in chapter 3 using appropriate values of the IP parameters.
2) Training each hypothesis of the resultant codebook (generated in step 1) to form a multi-class classifier.
3) Run PGD based white-box attack using methods outlined in chapter 4, to estimate the adversarial accuracy of the multi-class classifier (from previous step).

5.1 Estimating Error-Correlation between individual hypotheses of a codebook

In our earlier discussion in chapter 3, section 3.2.1, we highlighted that in communicating over a noisy channel, Error-Correcting Codes are powerful only when the errors made due to noise are random. For classification setup like ours, this implies that any two hypotheses (or classifiers) should not make errors on the same inputs. To avoid this, we ensured large column separation (eq. (3.5)) in our IP formulation. Large column separation reduces correlation by ensuring that the underlying decision

boundary of each hypothesis is sufficiently different from all other hypotheses in the codebook. However, due to the fact that IP formulation does not take into account the underlying data distribution (or the training data), we may still end up with hypotheses whose final predictions (or errors) are correlated. Therefore, measurement of such pairwise correlations between hypotheses can provide us with insights to better understand the final performance of a particular codebook. Moreover, it also will provide us with corroborative evidence to the fact that correlation between hypotheses should be avoided.

For now we assume that we have already trained each of our individual hypotheses for a given codebook (say $\mathbf{\Gamma_1}$). Also, let N_{test} denote the number of images in our test-set. For every binary classifier (corresponding to a column) in the codebook, we can compute the 0-1 loss (eq. (4.8)) for all images in the test set so that we have a vector $h_l \in \{0,1\}^{N_{test}} \ \forall \ l \in \{1,\ldots,L\}$. We can now compute the error-correlations between these binary vectors h_i & $h_j \ \forall \ (i,j) \ \in \ \{1,\ldots,L\}^2 | i \neq j$. This can be represented as a $L \times L$ matrix, which we will refer to as the correlation matrix $\mathcal{C}$ in our subsequent discussion.

Note that simply computing the correlations between binary vectors h_i and h_j will not be a true indicator of error-correlation. If two hypothesis have high accuracy, then examples for which the two hypotheses makes correct prediction will contribute to a high correlation value which is undesirable. High accuracy of the two hypotheses should definitely have a say in the error-correlation measure, and more importantly, should inversely contribute to the error-correlation measure. The number of examples in the test set for which both the hypothesis of a particular pair makes an error can be calculated as follows:

$$\mathcal{C}_{i,j}^{bw} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \mathbb{1}\{h_i[n] = 1 \wedge h_j[n] = 1\} \qquad (\textbf{both wrong}) \tag{5.1}$$

Similar to equation (5.1), we can also compute the the number of examples for

which both the hypotheses makes a correct prediction. This can be calculated as:

$$\mathcal{C}^{bc}_{i,j} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \mathbb{1}\{h_i[n] = 0 \wedge h_j[n] = 0\} \qquad \text{(both correct)} \tag{5.2}$$

We can now combine eqn: (5.1) and (5.2) to form our error-correlation measure as follows:

$$\mathcal{C}_{i,j} = \frac{\sum_{n=1}^{N_{test}} \mathbb{1}\{h_i[n] = 1 \wedge h_j[n] = 1\}}{\sum_{n=1}^{N_{test}} \mathbb{1}\{h_i[n] = 0 \wedge h_j[n] = 0\}} \tag{5.3}$$

The above measure can attain values greater than one but more importantly it takes into account the correct predictions made by individual hypotheses as well. The magnitude of this error-correlation measure (or the values in the error-correlation matrix C) will help us in understanding the accuracy of the overall classifier or codebook. In subsequent sections we will show that if the ECC based classifier has relatively higher accuracy then indeed error-correlation matrix has entries with lower values.

5.2 Experimental Setup

For all our experiments, we use CIFAR10 image dataset. This dataset comprises of 10 classes with 50000 images in the training set and 10000 images in the test set. In terms of computation resources, we run all our experiments on a system with a single 1080Ti Nvidia GPU, Intel Core i7-6800K CPU and 24 GB RAM. We use ResNet-18 [18] (as our Deep Neural Network architecture) as our binary classifier for all individual hypotheses. We use non-iterative probability estimates (eq. (4.7)) to compute our final class probabilities. We use these probability estimates to make predictions (as a substitute to Hamming or L_1/L_2 decoding). Furthermore, we use these estimates to compute a loss function (Cross-Entropy Loss) and then generate white-box PGD-attacks to evaluate the robustness of the overall classifier (see figure 4-1). We work with l_∞ norm as our set of allowed perturbations $\mathcal{S}$ with $\epsilon = 8$.

$$\mathcal{S}(x') = \{x \in R^d \mid ||x - x'||_\infty \leqslant \epsilon \;\; ; l \leqslant x \leqslant u\}$$

For CIFAR10, all images are of size $d = 32 \times 32 \times 3$ and for a valid image the pixel

values should be between $l = 0$ and $u = 255$.

We solve the IP formulated in chapter 3 (in Julia using Gurobi) with $k = 10, L = 20$ and $d = 4$, to generate our codebook. After solving the IP, we get the following optimal codebook shown in table 5.1. We denote this codebook by $\mathbf{\Gamma_1}$.

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	f20
C1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C2	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
C3	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
C4	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1
C5	0	1	1	0	1	0	0	0	1	1	0	1	0	0	1	0	1	1	0	1
C6	1	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0	1	1	0
C7	1	0	1	1	0	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0
C8	0	0	1	1	0	0	1	1	0	1	1	1	0	1	0	1	0	0	0	1
C9	1	1	0	1	0	0	1	0	1	0	0	1	1	0	0	1	0	1	0	1
C10	1	0	1	0	1	0	0	1	1	0	0	1	1	1	0	1	1	0	0	0

Table 5.1: Codebook $\mathbf{\Gamma_1}$ corresponding to the optimal solution of the IP solved with parameters $k = 10, L = 20$ and $d = 4$.

5.3 Evaluating codebook $\mathbf{\Gamma_1}$

We now work towards training and estimating the nominal and adversarial accuracy of the codebook $\mathbf{\Gamma_1}$ generated in the previous section. For the training part of the individual hypotheses, there can be two cases, first we can adversarially train each of the hypotheses on their individual classification problem and second we only perform nominal (or natural) training of each hypothesis. We investigate both of these cases for codebook Γ_1.

5.3.1 Evaluating codebook $\mathbf{\Gamma_1}$ with adversarially trained hypotheses

We adversarially train each individual hypothesis using the min-max approach [23]. The natural and adversarial accuracy of the resulting individual hypotheses is shown in table 5.2 .

We can now combine the above trained classifiers to form our multi-class classifier. Using non-iterative probability estimates (eq. 4.7) we can compute the class scores

Hypothesis	Binary classes	Adv. Accuracy	Natural Accuracy
f1	{1,6,7,9,10} , {2,3,4,5,8}	0.620	0.589
f2	{1,5,9} , {2,3,4,6,7,8,10}	0.799	0.755
f3	{1,5,7,8,10} , {2,3,4,6,9}	0.5	0.5
f4	{1,4,7,8,9} , {2,3,5,6,10}	0.587	0.552
f5	{1,4,5,6,10} , {2,3,7,8,9}	0.519	0.509
f6	{1,3,7} , {2,4,5,6,8,9,10}	0.739	0.720
f7	{1,3,6,8,9} , {2,4,5,7,10}	0.655	0.594
f8	{1,3,4,8,10} , {2,5,6,7,9}	0.5	0.5
f9	{1,3,4,5,7,9,10} , {2,6,8}	0.709	0.700
f10	{1,3,4,5,6,7,8} , {2,9,10}	0.915	0.805
f11	{1,2,8} , {3,4,5,6,7,9,10}	0.735	0.705
f12	{1,2,5,6,8,9,10} , {3,4,7}	0.701	0.7
f13	{1,2,4,9,10} , {3,5,6,7,8}	0.823	0.727
f14	{1,2,4,6,7,8,10} , {3,5,9}	0.7	0.7
f15	{1,2,4,5,7} , {3,6,8,9,10}	0.602	0.548
f16	{1,2,3,7,8,9,10} , {4,5,6}	0.735	0.704
f17	{1,2,3,5,10} , {4,6,7,8,9}	0.609	0.526
f18	{1,2,3,5,6,7,9} , {4,8,10}	0.738	0.701
f19	{1,2,3,4,6} , {5,7,8,9,10}	0.562	0.532
f20	{1,2,3,4,5,8,9} , {6,7,10}	0.699	0.7

Table 5.2: Adversarial and Natural accuracy of individual hypotheses (adversarially trained) in codebook $\mathbf{\Gamma_1}$

to make predictions and we use these estimates to run PGD attack. We achieve an overall nominal accuracy of around 42.27% and an adversarial accuracy of around 30.66%.

The reason for low natural accuracy of the overall classifier is due to low natural accuracy of each individual hypotheses. And low natural accuracy of individual hypotheses is due to adversarial training. Our next experiment will provide us with a better understanding of the effect of adversarial training of each hypothesis on the overall accuracy. In passing, we also compute the error-correlation matrices using the natural test-set images (table 5.3) and also for the adversarial examples generated for the overall classifier (table 5.4). We will discuss these matrices in next section once we have computed these matrices for the non-adversarially trained hypotheses as well.

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	f20
f1	-	0.152	0.642	0.480	0.599	0.270	0.355	0.517	0.256	0.063	0.145	0.261	0.108	0.143	0.346	0.187	0.289	0.158	0.428	0.453
f2	0.152	-	0.325	0.187	0.311	0.068	0.133	0.163	0.017	0.034	0.088	0.018	0.048	0.222	0.177	0.152	0.252	0.032	0.119	0.015
f3	0.642	0.325	-	0.743	0.936	0.408	0.466	1.000	0.319	0.097	0.423	0.331	0.144	0.333	0.635	0.307	0.632	0.384	0.556	0.500
f4	0.480	0.187	0.743	-	0.591	0.297	0.414	0.743	0.265	0.066	0.311	0.413	0.235	0.161	0.479	0.230	0.226	0.359	0.488	0.287
f5	0.599	0.311	0.936	0.591	-	0.210	0.443	0.936	0.290	0.097	0.231	0.311	0.278	0.318	0.635	0.496	0.587	0.393	0.746	0.471
f6	0.270	0.068	0.408	0.297	0.210	-	0.210	0.396	0.006	0.044	0.120	0.302	0.050	0.175	0.283	0.023	0.254	0.024	0.335	0.185
f7	0.355	0.133	0.466	0.414	0.443	0.210	-	0.574	0.314	0.046	0.228	0.204	0.076	0.249	0.321	0.186	0.316	0.204	0.467	0.215
f8	0.517	0.163	1.000	0.743	0.936	0.396	0.574	-	0.320	0.107	0.428	0.498	0.303	0.333	0.667	0.284	0.625	0.501	0.798	0.334
f9	0.256	0.017	0.319	0.265	0.290	0.006	0.314	0.320	-	0.023	0.286	0.002	0.043	0.001	0.271	0.143	0.176	0.159	0.396	0.189
f10	0.063	0.034	0.097	0.066	0.097	0.044	0.046	0.107	0.023	-	0.051	0.019	0.041	0.041	0.076	0.017	0.071	0.033	0.085	0.030
f11	0.145	0.088	0.423	0.311	0.231	0.120	0.228	0.428	0.286	0.051	-	0.010	0.067	0.019	0.312	0.018	0.216	0.152	0.338	0.007
f12	0.261	0.018	0.331	0.413	0.311	0.302	0.204	0.498	0.002	0.019	0.010	-	0.160	0.198	0.331	0.170	0.227	0.185	0.401	0.199
f13	0.108	0.048	0.144	0.235	0.278	0.050	0.076	0.303	0.043	0.041	0.067	0.160	-	0.046	0.188	0.127	0.109	0.152	0.235	0.044
f14	0.143	0.222	0.333	0.161	0.318	0.175	0.249	0.333	0.001	0.041	0.019	0.198	0.046	-	0.233	0.189	0.430	0.020	0.267	0.000
f15	0.346	0.177	0.635	0.479	0.635	0.283	0.321	0.667	0.271	0.076	0.312	0.331	0.188	0.233	-	0.262	0.425	0.235	0.617	0.182
f16	0.187	0.152	0.307	0.230	0.496	0.023	0.186	0.284	0.143	0.017	0.018	0.170	0.127	0.189	0.262	-	0.233	0.161	0.342	0.143
f17	0.289	0.252	0.632	0.226	0.587	0.254	0.316	0.625	0.176	0.071	0.216	0.227	0.109	0.430	0.425	0.233	-	0.144	0.504	0.150
f18	0.158	0.032	0.384	0.359	0.393	0.024	0.204	0.501	0.159	0.033	0.152	0.185	0.152	0.020	0.235	0.161	0.144	-	0.259	0.144
f19	0.428	0.119	0.556	0.488	0.746	0.335	0.467	0.798	0.396	0.085	0.338	0.401	0.235	0.267	0.617	0.342	0.504	0.259	-	0.244
f20	0.453	0.015	0.500	0.287	0.471	0.185	0.215	0.334	0.189	0.030	0.007	0.199	0.044	0.000	0.182	0.143	0.150	0.144	0.244	-

Table 5.3: Correlation matrix over Natural Examples in the Test Set

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	f20
f1	-	0.198	0.676	0.540	0.640	0.296	0.421	0.564	0.275	0.154	0.185	0.276	0.208	0.163	0.415	0.221	0.348	0.201	0.488	0.464
f2	0.198	-	0.367	0.253	0.355	0.088	0.189	0.210	0.032	0.077	0.117	0.027	0.092	0.251	0.226	0.170	0.299	0.057	0.163	0.030
f3	0.676	0.367	-	0.790	0.950	0.432	0.532	1.000	0.325	0.227	0.457	0.332	0.257	0.333	0.697	0.327	0.679	0.435	0.629	0.500
f4	0.540	0.253	0.790	-	0.660	0.328	0.491	0.791	0.288	0.167	0.356	0.430	0.323	0.196	0.551	0.263	0.301	0.402	0.561	0.308
f5	0.640	0.355	0.950	0.660	-	0.242	0.516	0.950	0.303	0.224	0.270	0.317	0.371	0.325	0.691	0.524	0.644	0.436	0.794	0.477
f6	0.296	0.088	0.432	0.328	0.242	-	0.242	0.421	0.010	0.096	0.143	0.309	0.094	0.185	0.313	0.034	0.283	0.036	0.368	0.192
f7	0.421	0.189	0.532	0.491	0.516	0.242	-	0.635	0.346	0.126	0.272	0.226	0.156	0.278	0.386	0.223	0.380	0.253	0.537	0.244
f8	0.564	0.210	1.000	0.791	0.950	0.421	0.635	-	0.328	0.239	0.463	0.498	0.399	0.333	0.708	0.312	0.676	0.540	0.835	0.334
f9	0.275	0.032	0.325	0.288	0.303	0.010	0.346	0.328	-	0.077	0.308	0.005	0.096	0.004	0.300	0.162	0.208	0.185	0.420	0.193
f10	0.154	0.077	0.227	0.167	0.224	0.096	0.126	0.239	0.077	-	0.114	0.059	0.131	0.093	0.176	0.058	0.175	0.088	0.198	0.085
f11	0.185	0.117	0.457	0.356	0.270	0.143	0.272	0.463	0.308	0.114	-	0.017	0.120	0.031	0.360	0.028	0.270	0.185	0.386	0.014
f12	0.276	0.027	0.332	0.430	0.317	0.309	0.226	0.498	0.005	0.059	0.017	-	0.198	0.198	0.343	0.183	0.245	0.194	0.415	0.199
f13	0.208	0.092	0.257	0.323	0.371	0.094	0.156	0.399	0.096	0.131	0.120	0.198	-	0.093	0.260	0.172	0.208	0.200	0.326	0.098
f14	0.163	0.251	0.333	0.196	0.325	0.185	0.278	0.333	0.004	0.093	0.031	0.198	0.093	-	0.247	0.198	0.446	0.032	0.283	0.000
f15	0.415	0.226	0.697	0.551	0.691	0.313	0.386	0.708	0.300	0.176	0.360	0.343	0.260	0.247	-	0.298	0.489	0.277	0.664	0.214
f16	0.221	0.170	0.327	0.263	0.524	0.034	0.223	0.312	0.162	0.058	0.028	0.183	0.172	0.198	0.298	-	0.262	0.182	0.379	0.160
f17	0.348	0.299	0.679	0.301	0.644	0.283	0.380	0.676	0.208	0.175	0.270	0.245	0.208	0.446	0.489	0.262	-	0.189	0.572	0.165
f18	0.201	0.057	0.435	0.402	0.436	0.036	0.253	0.540	0.185	0.088	0.185	0.194	0.200	0.032	0.277	0.182	0.189	-	0.297	0.169
f19	0.488	0.163	0.629	0.561	0.794	0.368	0.537	0.835	0.420	0.198	0.386	0.415	0.326	0.283	0.664	0.379	0.572	0.297	-	0.262
f20	0.464	0.030	0.500	0.308	0.477	0.192	0.244	0.334	0.193	0.085	0.014	0.199	0.098	0.000	0.214	0.160	0.165	0.169	0.262	-

Table 5.4: Correlation matrix over Adversarial Examples for the overall classifier generated from Test Set using PGD-attack

5.3.2 Evaluating codebook Γ_1 with nominally trained hypotheses

We now nominally train each of the individual hypotheses in codebook $\mathbf{\Gamma_1}$. Natural and adversarial accuracy of each resulting hypothesis is reported in table 5.5. Its no suprise that the adversarial accuracy for all hypotheses is zero, however note the increase in the natural accuracy of individual hypotheses in comparison to the previous case (table 5.2).

We now combine the nominally trained classifiers to form the multi-class classifier. Identical to previous case, we use non-iterative probability estimates (eq. 4.7) to compute the class scores to make predictions and use these estimates to run PGD-attack.

Hypothesis	Binary classes	Adv. Accuracy	Natural Accuracy
f1	{1,6,7,9,10},{2,3,4,5,8}	0.0	0.786
f2	{1,5,9},{2,3,4,6,7,8,10}	0.0	0.873
f3	{1,5,7,8,10},{2,3,4,6,9}	0.0	0.817
f4	{1,4,7,8,9},{2,3,5,6,10}	0.0	0.785
f5	{1,4,5,6,10},{2,3,7,8,9}	0.0	0.794
f6	{1,3,7},{2,4,5,6,8,9,10}	0.0	0.858
f7	{1,3,6,8,9},{2,4,5,7,10}	0.0	0.819
f8	{1,3,4,8,10},{2,5,6,7,9}	0.0	0.762
f9	{1,3,4,5,7,9,10},{2,6,8}	0.0	0.850
f10	{1,3,4,5,6,7,8},{2,9,10}	0.0	0.942
f11	{1,2,8},{3,4,5,6,7,9,10}	0.0	0.862
f12	{1,2,5,6,8,9,10},{3,4,7}	0.0	0.797
f13	{1,2,4,9,10},{3,5,6,7,8}	0.0	0.864
f14	{1,2,4,6,7,8,10},{3,5,9}	0.0	0.816
f15	{1,2,4,5,7},{3,6,8,9,10}	0.0	0.78
f16	{1,2,3,7,8,9,10},{4,5,6}	0.0	0.853
f17	{1,2,3,5,10},{4,6,7,8,9}	0.0	0.822
f18	{1,2,3,5,6,7,9},{4,8,10}	0.0	0.824
f19	{1,2,3,4,6},{5,7,8,9,10}	0.0	0.813
f20	{1,2,3,4,5,8,9},{6,7,10}	0.0	0.830

Table 5.5: Adversarial and Natural accuracy of individual hypotheses (nominally trained) in codebook $\mathbf{\Gamma_1}$

We now achieve an overall nominal accuracy of around **76.25**% and an adversarial accuracy of around **16.48**%. High nominal accuracy is in itself impressive, but achieving adversarial accuracy of 16% is quite suprising given the fact that adversarial accuracy of all individual hypotheses (classifiers) is ***zero***.

This result clearly shows the potential of our proposed approach. Using this result we can now clearly explain the poor performance in previous case. In the previous case we achieved a higher adversarial accuracy of around 30%, but a very low natural accuracy of around 42%. We conjecture that in the previous case we took an extremely conservative approach by *independently* robustifying each classifer. Implying that instead of underscoring the robustness of individual classifier, we should have ideally generated the adversarial examples for the overall classifier (jointly using all hypotheses) and then should have used these examples for independent adversarial training.

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	f20
f1	-	0.055	0.087	0.111	0.098	0.057	0.087	0.129	0.083	0.019	0.064	0.105	0.079	0.088	0.113	0.067	0.085	0.088	0.092	0.099
f2	0.055	-	0.051	0.065	0.061	0.041	0.058	0.051	0.026	0.014	0.033	0.039	0.030	0.066	0.067	0.042	0.063	0.035	0.053	0.023
f3	0.087	0.051	-	0.082	0.095	0.061	0.073	0.104	0.053	0.024	0.059	0.084	0.053	0.072	0.094	0.061	0.072	0.068	0.110	0.061
f4	0.111	0.065	0.082	-	0.098	0.063	0.099	0.124	0.069	0.022	0.064	0.111	0.086	0.093	0.114	0.070	0.092	0.101	0.086	0.071
f5	0.098	0.061	0.095	0.098	-	0.079	0.084	0.125	0.067	0.026	0.070	0.088	0.060	0.089	0.106	0.094	0.086	0.081	0.089	0.065
f6	0.057	0.041	0.061	0.063	0.079	-	0.056	0.074	0.031	0.025	0.038	0.080	0.035	0.063	0.065	0.060	0.067	0.045	0.066	0.041
f7	0.087	0.058	0.073	0.099	0.084	0.056	-	0.104	0.074	0.018	0.045	0.087	0.060	0.075	0.118	0.071	0.075	0.075	0.079	0.061
f8	0.129	0.051	0.104	0.124	0.125	0.074	0.104	-	0.078	0.028	0.069	0.123	0.088	0.090	0.121	0.085	0.091	0.127	0.107	0.083
f9	0.083	0.026	0.053	0.069	0.067	0.031	0.074	0.078	-	0.011	0.057	0.054	0.046	0.049	0.082	0.043	0.053	0.066	0.059	0.066
f10	0.019	0.014	0.024	0.022	0.026	0.025	0.018	0.028	0.011	-	0.020	0.011	0.017	0.020	0.031	0.008	0.025	0.016	0.027	0.016
f11	0.064	0.033	0.059	0.064	0.070	0.038	0.045	0.069	0.057	0.020	-	0.029	0.030	0.049	0.072	0.037	0.051	0.052	0.064	0.030
f12	0.105	0.039	0.084	0.111	0.088	0.080	0.087	0.123	0.054	0.011	0.029	-	0.086	0.075	0.104	0.075	0.068	0.086	0.076	0.066
f13	0.079	0.030	0.053	0.086	0.060	0.035	0.060	0.088	0.046	0.017	0.030	0.086	-	0.046	0.074	0.044	0.045	0.068	0.051	0.043
f14	0.088	0.066	0.072	0.093	0.089	0.063	0.075	0.090	0.049	0.020	0.049	0.075	0.046	-	0.093	0.057	0.107	0.056	0.073	0.049
f15	0.113	0.067	0.094	0.114	0.106	0.065	0.118	0.121	0.082	0.031	0.072	0.104	0.074	0.093	-	0.070	0.089	0.086	0.101	0.074
f16	0.067	0.042	0.061	0.070	0.094	0.060	0.071	0.085	0.043	0.008	0.037	0.075	0.044	0.057	0.070	-	0.057	0.061	0.060	0.047
f17	0.085	0.063	0.072	0.092	0.086	0.067	0.075	0.091	0.053	0.025	0.051	0.068	0.045	0.107	0.089	0.057	-	0.057	0.076	0.045
f18	0.088	0.035	0.068	0.101	0.081	0.045	0.075	0.127	0.066	0.016	0.052	0.086	0.068	0.056	0.086	0.061	0.057	-	0.072	0.070
f19	0.092	0.053	0.110	0.086	0.089	0.066	0.079	0.107	0.059	0.027	0.064	0.076	0.051	0.073	0.101	0.060	0.076	0.072	-	0.065
f20	0.099	0.023	0.061	0.071	0.065	0.041	0.061	0.083	0.066	0.016	0.030	0.066	0.043	0.049	0.074	0.047	0.045	0.070	0.065	-

Table 5.6: Correlation matrix over Natural Examples in the Test Set

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	f20
f1	-	0.653	1.279	1.224	1.399	0.667	1.003	1.490	0.864	0.251	0.799	0.804	0.666	0.883	1.430	0.666	0.924	0.898	1.239	0.989
f2	0.653	-	0.634	0.665	0.712	0.366	0.539	0.733	0.355	0.150	0.361	0.418	0.301	0.537	0.729	0.338	0.519	0.377	0.608	0.411
f3	1.279	0.634	-	1.211	1.250	0.668	0.938	1.365	0.759	0.325	0.766	0.734	0.562	0.840	1.328	0.626	0.870	0.841	1.114	0.925
f4	1.224	0.665	1.211	-	1.345	0.626	0.949	1.456	0.781	0.280	0.727	0.773	0.647	0.850	1.368	0.633	0.883	0.849	1.179	0.924
f5	1.399	0.712	1.250	1.345	-	0.778	1.055	1.512	0.866	0.352	0.846	0.825	0.626	0.930	1.492	0.761	0.970	0.927	1.266	1.027
f6	0.667	0.366	0.668	0.626	0.778	-	0.511	0.815	0.352	0.220	0.370	0.488	0.294	0.501	0.760	0.425	0.526	0.375	0.673	0.456
f7	1.003	0.539	0.938	0.949	1.055	0.511	-	1.143	0.666	0.226	0.533	0.615	0.506	0.655	1.066	0.510	0.698	0.662	0.917	0.734
f8	1.490	0.733	1.365	1.456	1.512	0.815	1.143	-	0.958	0.375	0.906	0.913	0.732	1.021	1.596	0.768	1.066	1.014	1.352	1.110
f9	0.864	0.355	0.759	0.781	0.866	0.352	0.666	0.958	-	0.152	0.533	0.493	0.394	0.500	0.920	0.373	0.550	0.606	0.764	0.655
f10	0.251	0.150	0.325	0.280	0.352	0.220	0.226	0.375	0.152	-	0.227	0.116	0.187	0.221	0.371	0.112	0.279	0.148	0.319	0.177
f11	0.799	0.361	0.766	0.727	0.846	0.370	0.533	0.906	0.533	0.227	-	0.326	0.307	0.533	0.870	0.334	0.531	0.539	0.764	0.536
f12	0.804	0.418	0.734	0.773	0.825	0.488	0.615	0.913	0.493	0.116	0.326	-	0.448	0.520	0.868	0.440	0.524	0.503	0.686	0.581
f13	0.666	0.301	0.562	0.647	0.626	0.294	0.506	0.732	0.394	0.187	0.307	0.448	-	0.389	0.697	0.298	0.399	0.468	0.532	0.449
f14	0.883	0.537	0.840	0.850	0.930	0.501	0.655	1.021	0.500	0.221	0.533	0.520	0.389	-	0.963	0.430	0.711	0.501	0.812	0.600
f15	1.430	0.729	1.328	1.368	1.492	0.760	1.066	1.596	0.920	0.371	0.870	0.868	0.697	0.963	-	0.683	1.006	0.964	1.288	1.068
f16	0.666	0.338	0.626	0.633	0.761	0.425	0.510	0.768	0.373	0.112	0.334	0.440	0.298	0.430	0.683	-	0.457	0.417	0.619	0.485
f17	0.924	0.519	0.870	0.883	0.970	0.526	0.698	1.066	0.550	0.279	0.531	0.524	0.399	0.711	1.006	0.457	-	0.565	0.856	0.644
f18	0.898	0.377	0.841	0.849	0.927	0.375	0.662	1.014	0.606	0.148	0.539	0.503	0.468	0.501	0.964	0.417	0.565	-	0.819	0.707
f19	1.239	0.608	1.114	1.179	1.266	0.673	0.917	1.352	0.764	0.319	0.764	0.686	0.532	0.812	1.288	0.619	0.856	0.819	-	0.900
f20	0.989	0.411	0.925	0.924	1.027	0.456	0.734	1.110	0.655	0.177	0.536	0.581	0.449	0.600	1.068	0.485	0.644	0.707	0.900	-

Table 5.7: Correlation matrix over Adversarial Examples (generated using PGD) from the Test Set

In table 5.6 and 5.7, we compute the error-correlation matrices for the nominally trained individual hypotheses. Note that the gap between natural and adversarial accuracy is almost around 60%, which is quite substantial. *Can we understand this gap using error-correlation matrices?* It turns out that indeed this gap is quite clearly reflected in the error-correlation matrices. With a simple bird's-eye view, one can infer that the values in the two tables (5.6 and 5.7) differ by atleast an order of magnitude. Therefore, low error-correlation between hypotheses is synonymous to high accuracy. Note that this behavior is also reflected in error-correlation matrices of the previous experiment, however since there the gap between natural and adversarial accuracy is only around 12%, therefore the difference in the values of the corresponding error-

correlation matrices (table 5.3 and 5.4) is not stark, but with a close inspection one can definitely notice the difference.

5.4 Evaluating All-pairs (1-vs-1) Codebook

We now do a similar analysis on all-pairs codebook (figure 3-1b). This experiment is important for the purposes of benchmarking as all-pairs codebook has shown good performance in many independent studies [25] [13] in the *nominal* setting. However, to the best of our knowledge, its performance in the *adversarial* setting has not been investigated in the literature. We report the accuracy of the individual classifiers resulting from nominal and adversarial training in table 5.8.

Hypothesis	Pairs	Nominally Trained		Adversarially Trained	
		Adv. Accuracy	Natural Accuracy	Natural Accuracy	Adv. Accuracy
f1	{10},{1}	0.0	0.9215	0.9065	0.775
f2	{10},{2}	0.0	0.8885	0.6525	0.619
f3	{10},{3}	0.0	0.9485	0.9115	0.816
f4	{10},{4}	0.0	0.956	0.9205	0.8
f5	{10},{5}	0.0	0.9685	0.9275	0.8385
f6	{10},{6}	0.0	0.964	0.9245	0.84
f7	{10},{7}	0.001	0.976	0.954	0.8625
f8	{10},{8}	0.0	0.9585	0.9185	0.7905
f9	{10},{9}	0.0	0.951	0.8735	0.7455
f10	{9},{1}	0.0	0.908	0.6895	0.631
f11	{9},{2}	0.0	0.9465	0.8595	0.756
f12	{9},{3}	0.0	0.9495	0.922	0.8115
f13	{9},{4}	0.0	0.954	0.9305	0.826
f14	{9},{5}	0.0	0.9635	0.9565	0.8625
f15	{9},{6}	0.0	0.965	0.9595	0.874
f16	{9},{7}	0.002	0.9745	0.9725	0.9035
f17	{9},{8}	0.0	0.97	0.959	0.8505
f18	{8},{1}	0.0	0.9505	0.933	0.8375
f19	{8},{2}	0.0	0.969	0.9755	0.874
f20	{8},{3}	0.0	0.8895	0.8535	0.6785
f21	{8},{4}	0.0	0.8905	0.7835	0.635
f22	{8},{5}	0.0	0.8705	0.781	0.69
f23	{8},{6}	0.0	0.8635	0.6955	0.6205
f24	{8},{7}	0.0	0.951	0.897	0.7285
f25	{7},{1}	0.0005	0.9645	0.94	0.8725
f26	{7},{2}	0.0005	0.9725	0.9705	0.87
f27	{7},{3}	0.0	0.875	0.618	0.5805
f28	{7},{4}	0.0	0.8725	0.7105	0.629
f29	{7},{5}	0.0	0.8905	0.5425	0.5225
f30	{7},{6}	0.0	0.9205	0.8025	0.6875
f31	{6},{1}	0.0	0.9485	0.9135	0.833
f32	{6},{2}	0.0	0.9725	0.961	0.8615
f33	{6},{3}	0.0	0.841	0.7445	0.624
f34	{6},{4}	0.0	0.7325	0.5	0.5
f35	{6},{5}	0.0	0.861	0.7625	0.661
f36	{5},{1}	0.0	0.943	0.9015	0.825
f37	{5},{2}	0.0005	0.975	0.957	0.855
f38	{5},{3}	0.0	0.836	0.591	0.55
f39	{5},{4}	0.0	0.8335	0.767	0.6215
f40	{4},{1}	0.0	0.943	0.9115	0.822
f41	{4},{2}	0.0	0.9535	0.951	0.8245
f42	{4},{3}	0.0	0.8115	0.734	0.5915
f43	{3},{1}	0.0	0.8905	0.863	0.756
f44	{3},{2}	0.0	0.954	0.944	0.826
f45	{2},{1}	0.0005	0.9515	0.938	0.778
Average:		0.0001	0.922	0.848	0.750

Table 5.8: Accuracy of individual hypotheses for All-pairs codebook

For the case when all the hypotheses are adversarially trained the classifier resulting from all-pairs codebook achieves a nominal accuracy of 51.49% and an adversarial accuracy of 25.5%. On the other hand, when all the hypotheses are nominally trained, the resulting classifier achieves a nominal accuracy of 68.76% and an adversarial accuracy of 0.0%.

These results clearly indicates that adversarial accuracy of 16.48% achieved by codebook $\mathbf{\Gamma_1}$ is by no means trivial. In exactly the same setting, all-pairs achieves no robustness (0.0%). Also, $\mathbf{\Gamma_1}$ achieves a higher nominal accuracy of 76.25% in comparison to 68.76% achieved by all-pairs. Apart from the benefit of achieving higher accuracy, codebook $\mathbf{\Gamma_1}$ uses only 20 classifiers in comparison to the 45 classifiers used by all-pairs. This compactness significantly reduces the computation burden when doing predictions. In the adversarially trained case, all-pairs achieve a higher nominal accuracy but at the cost of lower adversarial accuracy in comparison to $\mathbf{\Gamma_1}$.

5.5 Role of Network Capacity in $\mathbf{\Gamma_1}$

In section 5.3.2, using only *nominally* trained classifiers, codebook $\mathbf{\Gamma_1}$ achieved an adversarial accuracy of 16.48% and nominal accuracy of 76.25%. Without any adversarial or robust training, robustness of 16% is significant and shows the potential of our proposed methodology. However, given that we are combining the output of 20 classifiers, each of which is a ResNet-18, a natural question arises:
Is network capacity (of the overall classifier) the main reason for this robustness?

Recall that to evaluate the robustness we combine the outputs of each of the hypotheses (individually trained before) and form a multi-class classifier using non-iterative probability estimates. We then do a PGD based evaluation of the resulting classifier. To investigate the role of network capacity, we now in the same manner, combine untrained hypotheses (ResNet-18) to form a multi-class classifier (say $\mathcal{F}(x)$), see figure 4-1. We now nominally train this 10-class classifier $\mathcal{F}(x)$ end-to-end over the training data (CIFAR10). $\mathcal{F}(x)$ has exactly the same network architecture and capacity as our classifier resulting from codebook $\mathbf{\Gamma_1}$.

We now evaluate the nominal and adversarial accuracy of $\mathcal{F}(x)$ using the same PGD attack which we used for $\mathbf{\Gamma_1}$. $\mathcal{F}(x)$ achieves a nominal accuracy of around 78% and an adversarial accuracy of 0.27%. The lack of robustness of $\mathcal{F}(x)$ shows that network capacity alone in itself is not the reason for robustness of $\mathbf{\Gamma_1}$. Note that this does not imply that we cannot improve the performance of $\mathbf{\Gamma_1}$ using individual classifiers of higher capacity. Exploring the effect of network capacity of individual hypotheses on the robustness of the overall classfier will be an interesting direction for future work.

Chapter 6

Conclusion and Future Work

6.1 Concluding Remarks

In this thesis, we investigated the vulnerability of modern Machine Learning (ML) systems to adversarial perturbations. As a motivating example we showed the brittleness of ML systems using a commercial computer vision service. From an urban-system's perspective we discussed how these vulnerabilities can impact the operation and security of modern transportation networks, which for efficient maintenance and control are increasingly moving towards data-driven solutions. Rapid adoption of ML systems in such critical real-world applications, provides enough reason to study and improve the robustness of modern ML systems.

We discussed different methods of training an adversarially robust model including the challenges associated in correctly estimating the adversarial accuracy of Deep Neural Networks (DNNs). These challenges have given rise to the new field of verifiable-AI. In terms of novel contributions, we have proposed and investigated a new method of defense to train adversarially robust DNNs for classification problems. Our proposed approach builds upon the idea of Error-Correcting Codes(ECCs) for classification tasks to achieve robustness. Using our approach we provide evidence to the fact that DNNs can be robust even without adversarial training. This unique and suprising outcome of our proposed method provides affirmation that ECCs can particularly be useful in the adversarial setting.

We first provided an efficient Integer Programming (IP) formulation to generate codebooks following the guidelines of codebook design from literature [13] [25]. This IP based method provides a systematic and optimization driven method of codebook generation. Furthermore it can be useful to study the effect of different guidelines, by generating new codebooks by simply adjusting the parameters of the IP. We then used the codebook (solution of the IP) to train a ECC based (robust) classifier. To rigorously estimate the adversarial accuracy of the resulting ECC based classifier, we also provided methods to perform white-box attacks.

We supported our claim with experiments on CIFAR10 dataset using l_∞-norm based perturbations with $\epsilon = 8$. Our IP generated codebook substantially outperformed one-vs-one codebook, both in terms of natural and adversarial accuracy, particularly for the case when the individual hypotheses were nominally trained. To further discern the benefit of using ECCs from network capacity, we showed that the robustness of our ECC based classifier (resulting from the IP generated codebook) is not a manifestation of high network capacity of the overall ECC based classifier.

6.2 Future Work

The experiments discussed in chapter 5 are encouraging and warrant further analysis and more exhaustive experiments. These include:

1. In the experimental setup in chapter 5 we used a threat model with $\epsilon = 8$, which results in large uncertainty sets. We should also evaluate our approach against smaller uncertainty sets (lower values of ϵ) and on simpler datasets including MNIST.

2. The role of the number of columns in the codebook should also be studied in more detail, although our current codebook is compact in comparison to one-vs-one, but there could be potentially more compact codebooks with similar robustness. Also, the effect of high column separation on the performance should be investigated. In summary, a sensitivity analysis with respect to the

parameters of the IP should be conducted.

3. We performed experiments by training all the hypotheses either nominally or adversarially, however there could be a case where only some of the hypotheses are adversarially trained and remaining hypotheses are nominally trained. This can significantly improve the adversarial accuracy (over the nominally trained case) with minimally reducing the natural accuracy.

4. Finally, for the purposes of benchmarking, one can advance the DECOC (Discriminant Error Correcting Output Codes) approach [25] discussed in chapter 3 section 3.2.4 to account for adversarial perturbations. Furthermore, influenced by DECOC approach, one can propose an iterative procedure which takes into account the accuracy of the resulting codebook (or even the individual classifiers) from the IP and then using this information to solve a new IP or generate a new codebook.

Bibliography

[1] https://boston.cbslocal.com/2019/09/12/carpool-lanes-hov-massachusetts-boston-traffic/.

[2] https://komonews.com/news/local/drivers-most-interesting-way-to-cheat-the-hov-lane-fails-11-20-2015.

[3] https://web.archive.org/web/20130616015258/http://www.kgw.com/news/blow-up-doll-flunks-seattle-hov-lane-ploy-121329989.html.

[4] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[5] Miguel Ángel Bautista Martin, Oriol Pujol, Fernando De la Torre, and Sergio Escalera. Error-correcting factorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(10).

[6] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, July 2018.

[7] A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.

[8] Ralph A. Bradley and Milton E. Terry. The rank analysis of incomplete block designs i:the method of paired comparisons. *Biometrika*, 39:324–345, 1952.

[9] Alvaro Cárdenas, Saurabh Amin, and Shankar Sastry. Secure control: Towards survivable cyber-physical systems. In *First International Workshop on Cyber-Physical Systems (WCPS2008)*. IEEE, June 2008.

[10] Alvaro A. Cárdenas, Saurabh Amin, and Shankar Sastry. Research challenges for the security of control systems. In *Proceedings of the 3rd Conference on Hot Topics in Security*, HOTSEC'08, pages 6:1–6:6, Berkeley, CA, USA, 2008. USENIX Association.

[11] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 513–530, Austin, TX, August 2016. USENIX Association.

[12] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2):201–233, May 2002.

[13] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2(1):263–286, January 1995.

[14] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18)*, pages 162–171, Corvallis, Oregon, 2018. AUAI Press.

[15] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995.

[16] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[17] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In *Advances in Neural Information Processing Systems 10*, NIPS '97, pages 507–513, Cambridge, MA, USA, 1998. MIT Press.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[19] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2137–2146, 2018.

[20] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[21] Stephen Kuciemba and Kathleen Swindler. Transportation management center video recording and archiving best general practices. Technical report, U.S. Department of Transportation - Federal Highway Administration, 2016.

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.

[23] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ArXiv*, abs/1706.06083, 2018.

[24] P. Pudil, F. J. Ferri, J. Novovicova, and J. Kittler. Floating search methods for feature selection with nonmonotonic criterion functions. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pages 279–283 vol.2, Oct 1994.

[25] Oriol Pujol, Petia Radeva, and Jordi Vitria. Discriminant ecoc: A heuristic method for application dependent design of error correcting output codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 28(6):1007–1012, June 2006.

[26] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *ArXiv*, abs/1801.09344, 2018.

[27] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 10900–10910, 2018.

[28] James C. Spall. An overview of the simultaneous perturbation method for efficient optimization. In *Airport Modeling and Simulation.*

[29] James C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[30] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[31] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[32] Florian Tramèr, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. The space of transferable adversarial examples. *ArXiv*, abs/1704.03453, 2017.

[33] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5286–5295. PMLR, 10–15 Jul 2018.

[34] Bianca Zadrozny. Reducing multiclass to binary by coupling probability estimates. In *Advances in Neural Information Processing Systems 14*, pages 1041–1048. MIT Press, 2002.