Asymptotic Graph Neural Networks Improve Adversarial Robustness in Machine Vision

by

Michael O. Vertolli

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Cognitive Science

Carleton University
Ottawa, Ontario

Abstract

I propose and evaluate the first theory and computational model that explicitly describes the cognitive constraints—in the form of relational architectural constraints—that produce adversarial robustness in machine vision. This theory proposes a new form of Graph Neural Network (GNN), called an asymptotic GNN, that uses a nonlinearity with a vertical asymptote to constrain where the network should be sensitive and insensitive to variations in its substructure. This approach proposes a new invariance for artificial neural networks, similar to the translational invariance of convolutional layers. I call this type of invariance "substructure invariance" and I show how it is capable of producing adversarial robustness in machine vision.

To support these claims I first provide a detailed analysis of the two core components of the model: the asymptotic barrier function and learning branch. I then compare the asymptotic GNN to four state-of-the-art models on the MNIST dataset: one baseline and three adversarially robust models. I show that the asymptotic GNN outperforms all of the models on gradient-based adversarial attacks, mainly the DeepFool attack and the Momentum Iterative Method, and performs similar to the state-of-the-art on the decision-based PointWise attack. This establishes that the asymptotic GNN is a viable candidate for adversarial robust modeling. In subsequent analyses, I show that ablations to the core asymptotic barrier function and learning branch distinctly and differentiably impair the adversarial robustness of the asymptotic GNN. I then vary the basic structure of my model to demonstrate how

the PointWise attack is symptomatic of a fundamentally different type of problem than substructure-invariance, mainly a feedback-based denoising operation.

Acknowledgments

This work would not have been possible without various members of the academic community within and outside of Carleton University. First and foremost, I would like to thank my supervisor, Jim Davies, for his tireless support on all matters related to my academic career. Thanks to my committee member Raj Singh for his guidance and support. Thanks to Kasia Muldner for her support as a committee member. Thanks to Ingo Freund for his guidance and support through all the challenges in the last stages of my degree. Thanks to Waseem Gharbieh and Ehsan Amjadian for their editorial support. Thanks to Jeff Orchard for his support on the revisions of this document. Thanks to all the members of the Science of Imagination Lab for their patience and support. Finally, thank you to Jo-Anne Lefevre and all of the administration at the ICS for their support and dedication over the course of my academic career. Whatever success I have achieved is largely the consequence of the overwhelming support of this community, friends, and family. I would also like to give a special thanks to my partner, Vanessa Michela, for her endless patience, love, and support through my long, winding path through the academy.

This research was funded by the Natural Sciences and Engineering Research Council, which made it possible for me to dedicate my full-time to this work.

Contents

Abstract **1**

Acknowledgments **3**

Table of Contents **6**

List of Tables **7**

List of Figures **8**

List of Appendices **9**

Notation and Symbols **10**

1 Introduction **12**

1.1 Visual Nativism 14

1.2 The Structure of CNNs 18

1.3 Graph Neural Networks 21

1.4 Adversarial Robustness 26

1.5 Asymptotic Hierarchies 28

1.6 Outline 32

2 Model and Implementation **35**

2.1 Asymptotic GNNs 36

2.2 A Framework For The Asymptotic Barrier 39

2.3 The Product Aggregator 47
2.4 The Asymptotic Barrier 50
2.5 The Multiclass Asymptotic Barrier Function 57
2.6 Learning Branch 66
2.7 Substructure Processing 73
2.7.1 Basic setup 74
2.8 Summary 76

3 Baseline Experiments **79**
3.1 Baselines and SOTA Models 80
3.2 Evaluations of Adversarial Robustness 86
3.3 Quantitative Comparisons 89
3.4 Experiments 90
3.5 Methodology 90
3.6 Results 91
3.7 Discussion 93
3.7.1 Substructure VS Spatial Translation 93
3.7.2 Architecture VS Data Augmentation 95
3.7.3 Architecture VS Primitives 95
3.8 Summary 97

4 Evaluating the Essential Structure of Asymptotic GNNs **98**
4.1 Ablation Models 99
4.2 Experiments 102
4.3 Methodology 103
4.4 Results 103
4.5 Discussion 105
4.5.1 The asymptotic barrier ablation 105

4.5.2 The learning branch ablation 105
4.6 Summary 106

5 Modeling the Inductive Bias of PWA **107**
5.1 Alternative Design Choices for ϕ 109
5.2 Experiments 111
5.3 Methodology 111
5.4 Results 112
5.5 Discussion 112
5.6 Summary 114

6 Conclusion **116**
6.1 Limitations 118
6.2 Future Work 121

References **123**

Appendix **128**

List of Tables

1 Baseline CNN and asymptotic GNN adversarial results 92

2 Transfer attacks on CNNs and asymptotic GNN 92

3 Asymptotic GNN ablations for DFA, MIM, and PWA results 104

4 Transfer attacks for asymptotic GNN and ablations 104

5 Denoising DFA, MIM, and PWA results 112

6 Transfer attacks between asymptotic GNN, DAE, denoising sampler . 113

7 List of Hyperparameters 134

List of Figures

1 Adversarial examples with only two pixels changed 12

2 Visual transformation example 15

3 Visualization of Asymptotic GNN Architecture 29

4 Graph of the unit step function $H(x, \delta)$ 41

5 Graph of unit step function $H(\mu, \delta)$ and ϵ 43

6 Graph of ϵ without the unit step function 45

7 Graph of effect of ϵ on the product 49

8 Plots of the asymptotic barrier function 53

9 Graph of the asymptotic barrier function when $\psi = 4$ and $\kappa = \frac{1}{e}$ . . . 54

10 Graph of ϵ with asymptotic barrier function (Γ). 56

11 Visualization of substructure mismatching in an image 58

12 Graph of the multiclass asymptotic barrier with two classes 62

13 Graph of the multiclass asymptotic barrier function with three classes 63

14 Adversarial examples produced by the MIM for the CNN baseline . . 87

15 Adversarial examples produced by the DFA for the CNN baseline . . 88

16 Adversarial examples produced by the PWA for the CNN baseline . . 88

17 Reproduction of two plots from Chapter 2 100

Appendices

Proof Sketch of Γ's Approximation of $H(x, \delta)$ 129
Implementation Details 132
Training Details 133
Hyperparameters 134

Notation and Symbols

a, b, c	scalars
$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$	vectors
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	set of strong predictors for class A, B, C
f, F	arbitrary function
G	graph
V	base (middle) level of graph
V^{Γ}	set of barrier nodes
S	substructure (lowest) level of graph
C	superstructure (highest; class) level of graph
E	an edge of the graph (not used)
$\boldsymbol{x}$	input image
$\hat{\boldsymbol{x}}$	sampled input image from denoising sampler
$\boldsymbol{z}_i$	latent vector for image patch i
$\boldsymbol{v}_i$	node vector (image patch) for node i
$\boldsymbol{v}_i^{\Gamma}$	barrier node vector for node i
$\overline{\boldsymbol{v}}$	aggregated node vector
$\widetilde{\boldsymbol{v}}$	probablistic node vector over K classes
$\boldsymbol{s}_i$	substructure vector (pixel values) for image patch i
$\hat{\boldsymbol{s}}_i$	image patch i from image sample $\hat{\boldsymbol{x}}$
c	superstructure (image class)

N^v	number of node vectors in V
N^s	number of substructure vectors in S
N^c	number of superstructure vectors in C (always 1)
θ	trainable parameters of a neural network
$\boldsymbol{W}, \boldsymbol{W'}$	trainable weights of a fully-connected neural network
$\boldsymbol{b}, \boldsymbol{b'}$	trainable biases of a fully-connected neural network
ϕ	substructure function (usually neural network)
ϕ_α	first latent image patch encoding step of ϕ
ϕ_γ	decoder of ϕ_α; used for training
ϕ_β	second step of ϕ that outputs a probabilistic vector over the K classes for each image patch
Γ	asymptotic barrier function
ψ	scaling hyperparameter for Γ
κ	hyperparameter for Γ that truncates, sets the maximum, and sets the initialization point for the multiclass Γ
K	number of classes in the multiclass Γ
Ω	learning branch
τ	learning branch trainable parameter
ζ	learning branch hyperparameter that flattens the local topology within a class
$\prod$	product or product aggregator
$H(x, \delta)$	the unit step function
δ	shift for unit step function (pp. 37-41; Ch. 4); distance (p. 62)
ϵ	change applied to a single node vector (Ch. 2, p. 106); change to the pixels in an adversarial example (Ch. 3-5)
μ	constant value when $v_i = v_j$
$\boldsymbol{\eta}$	noise vector added to the input image for the denoising autoencoder and denoising sampler
t	proximity threshold bounding values close to 0
t_h	horizontal threshold similar to t
t_v	vertical line where t_h intersects the graph of the function

Chapter 1

Introduction

Deep neural networks trained to classify images are known to be susceptible to adversarial examples: images that are modified slightly to produce an incorrect classification (Szegedy et al., 2013; Biggio & Roli, 2018). Although there is some evidence that adversarial examples can transfer to time-limited humans, it is generally accepted that humans with no time limit are extremely robust to adversarial examples (Elsayed et al., 2018). At minimum, simple visual inspection supports this idea (see Figure 1).

Numerous engineering solutions have been proposed to improve the adversarial robustness of machine learning models of all types (for review, see Biggio & Roli, 2018). In their original context, these solutions are most often designed to improve the security of machine learning algorithms (i.e., make them less susceptible to tampering through modifications to the input). The two most common techniques are to add adversarial examples during training and to detect when an input is adversarial so it can be handled appropriately. Problematically, neither of these solutions have any psychological plausibility nor were they intended to. For example, Schott, Rauber,

Figure 1: A random pick of adversarial examples created using the Pointwise adversarial attack on a baseline convolutional neural network. Only two pixels have been changed in each image. Human visual perception can easily identify these images correctly.

Bethge, and Brendel (2019) show that adding adversarial examples to the training set actually results in internal representations that are heavily distorted. It teaches the network to recognize examples that are very noisy rather than how to detect clean representations within the noise.

To date, there is no theory or model of cognition that explains how and why human visual information processing is not susceptible to these distortions that easily fool state-of-the-art (SOTA) neural networks. The closest example I could find is a neurocognitive theory and model of neural plasticity in perceptual learning (Dosher & Lu, 1998; Dosher, Jeter, Liu, & Lu, 2013; Petrov, Dosher, & Lu, 2005). Although there are a number of similarities between my proposed model and this work, the underlying focus of the two projects is different. The authors are interested in understanding perceptual learning (or modification) in adult life, while my focus is on the fundamental mechanisms required for any adversarial robustness. One would expect the latter to develop during early childhood, in direct contrast to the above mentioned research on neural plasticity in adult life. For example, incorporating training during the experiment is an essential component of the authors' methodology, but no training or preparation outside of one's existing lived experiences is required to correctly identify any of the images in Figure 1. As a consequence, I would speculate that these two tasks will have similar but distinct underlying mechanisms. A more detailed comparison with this research was left to future work.

Given the significant cross-pollination between cognitive neuroscience and artificial intelligence research (for a review, see Hassabis, Kumaran, Summerfield, & Botvinick, 2017), there is a growing interest in cognitive theories that can explain and inform the study of adversarial robustness (Elsayed et al., 2018). In what follows, I propose and evaluate a theory and computational model of adversarial robustness that takes inspiration from cognitive science—a first step towards a cognitive theory of adversarial robustness. In the next section I discuss some of the underlying theoretical

commitments of my approach.

1.1 Visual Nativism

My proposed approach takes as its core assumption a form of visual nativism. It leverages Chomsky's (1980) "poverty of the stimulus" argument and directly extends it to the visual domain. However, it is worth highlighting that there are many different interpretations and debate of Chomsky's work in cognitive science. The current description is my own interpretation applied to the problem of adversarial examples in machine vision.

In essence, Chomsky's argument is that the experiences of a child are not sufficiently rich to enable them to learn the full range of possible linguistic behaviour. Consequently, additional internal structure is required to fully specify the grammar of human languages. Extending this basic idea from Chomsky (1959, 1980) and similar theorists, I assume that visual experiences also fail to be rich enough to describe the full range of possible *visual* behaviour (i.e., perception). Pullum and Scholz (2002) describe Chomsky's argument in terms of five claims, which I directly map into the visual domain, below:

1. Every realistic visual transformation that takes as input one visual scene and returns as output another visual scene can be mapped to a set of transformational rules (i.e., a visual grammar).

2. For any given visual grammar, one can select data that would enable the separation of that grammar from any other grammar that is consistent with a given input.

3. One can show that the required data for a given grammar is missing from the experiences of children.

4. Yet, children are still able to acquire that visual grammar.

5. Therefore, the visual grammar must emerge from some property of the child.

Although I take these claims to be *assumptions* of the current work—hence, I will not defend them directly—it is worth considering a simple example that meets these requirements and is relevant to the current topic.

Consider the following simple example of a visual transformation: the optical distortion produced by a fish-eye lens. A fish-eye lens is a wide-angle lens that produces a pronounced convex distortion resembling a hemispherical image (Kumler & Bauer, 2000). It is reasonable to assume that this distortion will never have been experienced by many children. Yet, I would speculate that every child with a typical visual system will be able to perceive and interpret the objects within such a distortion. Although the empirical evaluation of this point is outside of the scope of this project, I encourage the reader to examine the picture on the next page to see if it requires experiential learning to enable recognition of the objects (assuming one already knows what the classes of objects are).

Over the course of this work, I will try to show that the problem of adversarial examples fundamentally hinges on these assumptions. If vision suffers from impoverished environmental stimuli, then the standard approaches for solving the problem of adversarial examples (e.g., by adding more training examples to the input) are *fundamentally* unable to address these concerns. There simply will never be enough examples to cover every possible visual transformation that humans can perceive.

One way to think about this problem is in terms of the constraints of a system. For

Figure 2: An example of a visual transformation that is unlikely to have been experienced by many children, especially given the additional distortion produced by the JPEG compression. Nevertheless, all of the broad classes of objects should be easily identifiable by any child with a typical visual system, although the specifics might be outside of their knowledge base. This is a candidate example in support of visual nativism that can be empirically evaluated.

any given system, there are certain things that system simply cannot do. Considering first a physical example with psychological implications, it is a generally accepted fact that the legs of a human have a much more constrained range of motion than the arms. Although this makes the arms much more flexible in their use, it also makes them more challenging to learn how to use. Evidence for this can be found in the motor development of young infants. For example, infants learn to engage with the world with their legs and feet *before* they use their arms and hands (Galloway & Thelen, 2004). Galloway and Thelen (2004) explicitly theorize that this developmental timeline is determined by differences in the mechanical constraints of these two limbs. The fewer mechanical constraints imposed by the system, the harder it is to learn.

Similarly, there are numerous examples of constraints in cognition. One classic example is Miller's (1956) "magical number seven," which initially defined the limits of human short-term memory. Neuroscience offers even more constraints. For example, the size, location, and connectivity of the human visual cortex imparts a number of unavoidable constraints to visual cognition (Hubel, 1982). Although the consequences of cognitive constraints are often challenging to determine (e.g., how would human visual cognition change if the locations and connectivity of the parietal and temporal lobes were swapped), the simpler physical example that I described above suggests that there *are* consequences. More importantly, the overwhelming stability of many of these biological constraints suggests that these consequences are *critically important*, at least to the survival of the species. What Chomsky adds to this discussion is that the biological constraints that exist prior to learning are also critical to cognition itself. In other words, they are critical to the very functions and capacities of the system.

It is my hypothesis that it is an absence *of* constraints that lead to adversarial examples. In an effort to make this discussion of constraints more concrete in the context of machine vision, it is worth considering a detailed example. This example

will inform much of the discussion that follows, including the proposed solution to adversarial robustness.

1.2 The Structure of CNNs

Convolutional neural networks (CNNs) have a long history and popularity in computer vision (LeCun et al., 1989; LeCun, Bottou, Bengio, & Haffner, 1998; Krizhevsky, Sutskever, & Hinton, 2012; Goodfellow, Bengio, & Courville, 2016). Their defining characteristic is the use of a special type of neural network layer that has become the default layer type for many models in this domain. The unique property of a convolutional layer relative to a standard fully-connected layer is that it has well-defined constraints on the relationships between different neurons between layers in a feed-forward neural network. Instead of allowing every input (e.g., pixel) to have a direct relationship with every other input—like a fully-connected layer—the convolutional layer constrains an input's direct relationships to its neighbours (e.g., a small image patch). To achieve this, the convolutional layer slides a small window with trainable weights across the entire image. This window or *kernel*, as it is often called, functions like a template that can be learned through training to identify complex features in the domain (e.g., cats, faces, airplanes).

The constraints imposed by the kernel of the convolutional layer has a number of interesting properties that relate to my discussion of visual nativism. The constraints are defined prior to learning and cannot be changed by the learning process. Yet, they directly affect learning by incorporating the researcher's assumptions into the model's architecture. For the convolutional layer, the main assumption is that visual processing is not affected by spatial translation. In other words, changing the location of an object in an image does not change the object. Battaglia et al. (2018) call this kind of assumption a "relational inductive bias" (RIB) to highlight that it is a constraint imposed by the researcher over the possible direct relationships that can occur between the inputs to a given layer of the model (for a detailed analysis of CNNs in terms of their inductive biases, see N. Cohen & Shashua, 2016). By contrast, all activation functions are inductive biases, but they are not *relational* as they do not

constrain the direct relationships between a layer's inputs. Throughout the rest of this work, I will focus on architectural constraints that are RIBs.

There are a number of key points stemming from my choice of RIBs as the central architectural constraints of interest. First, these constraints do not define any primitives. A convolutional layer does not specify what kinds of templates are acceptable or should be learned (i.e., the layer does not constrain the possible values of the parameters of the kernel; however, there are other inductive biases that do like dropout). These values are learned by the system rather than set ahead of time. Thus, one should restrain themselves from thinking of examples, like the psychological theory of geons or similar, that focus on constraints in the form of primitives (Biederman, 1987). This is not what I mean.

My second key point is that I am not suggesting that fully-connected neural network layers, or any other models for that matter, have no constraints. Fully-connected layers have a lot of constraints built into them (e.g., the number of input nodes, the number of hidden nodes in each layer, the number of output nodes, any non-linearities, initialization procedures). What I am suggesting is that a fully-connected neural network layer has fewer constraints over the acceptable direct relationships that can occur between inputs to the layer (Battaglia et al., 2018). One knows this to be true as a convolutional neural network can always be represented as a fully-connected neural network while the reverse is not possible. In the work that follows, a single, fully-connected neural network layer in isolation is the neural architecture with the fewest relational constraints that I consider as every input can be directly related to every other input and the outputs immediately follow from the inputs. But, it would be meaningless to say that this is the smallest set of relational architectural constraints *in general*.

My third point is that relational constraints are not restricted to the individual layers of neural networks. They can also be built into the organization of the layers

of a network. In this sense, a deep neural network with three layers that are all fully-connected has fewer constraints than a network where each layer is unique, has to occur in a specific order, etc. In general, a neural network with more relational constraints will be able to represent a smaller family of functions than their less constrained counterparts. However, I leave the specification of the size of the associated families to future work.

To summarize, my focus is on the pre-existing relational constraints that cannot be changed through learning and that incorporate a specific set of assumptions into the model. Each constraint that I am interested in has an associated RIB that defines which direct relationships are acceptable within the architecture. Each structure also has an associated invariance that is produced as a consequence of its constraints. For example, the convolutional layers that I discussed above are *invariant* to spatial translation (Battaglia et al., 2018). That is, they will correctly identify an object in an image no matter where it is located in that image.

Depending on the framing, there is some ambiguity in the literature around whether to use "invariance" or "equivariance." I strictly mean *in*variance in this work in the sense that the output class does *not* vary with a change in the location of the object being classified. By contrast, *equi*variance would occur in, for example, multiclass classification. In this setting, changes to the positions of the objects should produce equivalent changes to the labels of those objects (for a formal discussion, see Zaheer et al., 2017). That is, the labels and objects should vary *together*. Equivariance is not the focus of this work.

It is on the basis of these points that I will argue that there is no theory to date of adversarial robustness that describes the pre-existing *relational* constraints that produce adversarial robustness. I lay the groundwork for this objective by focusing on what I hypothesize are the necessary constraints, RIBs and associated invariances for adversarial robustness. This project begins with a discussion of neural network

layers that compute over graphs. These layers provide the foundation for my proposed constraint and RIB that I argue are necessary for adversarial robustness.

1.3 Graph Neural Networks

Graph neural networks (GNNs) are neural networks that can directly compute over almost any abstract collections of objects, associations, and the properties of those objects and associations (Gori, Monfardini, & Scarselli, 2005; Scarselli, Gori, Tsoi, Hagenbuchner, & Monfardini, 2008; Li, Tarlow, Brockschmidt, & Zemel, 2015). I have chosen to use the word "association" here instead of "relation" in order to differentiate between links within the content of the domain (i.e., associations) and the possible links between the inputs of a GNN layer (i.e., relationships). Although the two words are similar in common parlance, it is very important to keep them separate to prevent category errors in the current context. For example, I state later in this section that a GNN layer limits the possible relationships between its inputs, which can be nodes, edges, attributes, or some combination or aggregation thereof. These limits do not affect the edges encoded in the graph, which are defined by the domain. I will make an effort to be clear what type of link I am talking about throughout this chapter, but it is an easy slip to fall into conceptually. Thus, I respectfully request the diligence of the reader on this point.

This collection as a whole is often called a graph, which is formally defined in terms of its nodes (objects), edges (associations), and attributes (properties). I reserve my technical discussion of the graph formalism until the next chapter and focus on a more informal discussion for the rest of this section. An easy way to think about GNNs that avoids much of the abstraction is as a generalization of convolutional neural networks (CNNs). There are many other generalizations of CNNs (e.g., gauge-theoretic generalization, see T. S. Cohen, Weiler, Kicanaoglu, & Welling, 2019), so

this framing is not that unusual.

Just like CNNs, GNNs are defined by a special type of neural network layer (Battaglia et al., 2018). This GNN layer also has a well-defined set of relational constraints that constrain the direct relationships between the inputs to the layer to the local neighbourhoods of each input. What is special about the GNN layer—relative to the convolutional layer—is that it is able to compute over *arbitrary* neighbourhoods. For example, the inputs could be the objects in an image and their neighbourhoods could be between five and ten nearby objects with varying distances. This is very different from the regular grid of pixels that a convolutional kernal usually computes over.

There are three key differences between the arbitrary neighbourhoods of the GNN layer and the neighbourhoods of the standard convolutional layer. Each of these differences result from the fact that convolutional kernels compute over a *grid* of inputs. The first difference is that the grid fixes the position of each node in the neighbourhood so that a single, fixed kernel can be used across the entire image.[1] For example, in a 2D image, a 3-by-3 kernel always computes over the same nine positions (i.e., top-left, center-top, top-right, left, center, right, bottom-left, center-bottom, bottom-right) surrounding and including the center "source" pixel no matter where the source pixel is placed in the image.[2] By contrast, every node (e.g., pixel, object) processed by a GNN layer can potentially be connected to any other node (e.g., pixel, object) in an image. Second, the grid fixes the size of each neighbourhood to the size of the kernel. This means that, for a 3-by-3 kernel, every pixel in the image (excluding the edges) has a neighbourhood of nine pixels (including the source pixel).

[1]The diligent deep learning researcher will probably be thinking of all of the unique exceptions to the standard line, square, and cube convolutional kernels. In all but the rarest exceptions (for example, vision models with attention; see Parmar et al., 2018; Wu, Fan, Baevski, Dauphin, & Auli, 2019), my points will still stand. Additionally, many of the exceptions are proximate to or have even been formalized as GNNs in the literature. For example, see the generalization of visual attention models to the GNN framework by Wang, Girshick, Gupta, and He (2018).

[2]Again, I am simplifying here. The edges of an image are a special case, but these are implementation details that are irrelevant to the current discussion.

By contrast, every node (e.g., pixel, object) processed by a GNN layer can have any size of neighbourhood as well as a neighbourhood of a different size from every other node. Simpler and more regular setups are often used in practice. However, in the general case, the size of the neighbourhoods is completely arbitrary. And third, the grid fixes the size of the transitions between each 'step' of the kernel as it moves around the image. This means that a region in an image with lots of objects will still get an equal amount of processing as a region of an image that is mostly uniform or empty.[3] For a GNN, a denser region of an image can be encoded as a denser region of the graph (e.g., a region with nodes that are more closely packed together resulting in larger neighbourhoods). Thus, a GNN generalizes the basic structure of a CNN by enabling each node to have a neighbourhood that is more variable than the standard convolutional layer.

At this point, it might seem confusing to say that a GNN layer has a well-defined, pre-existing set of relational constraints *and* the relationships are more *arbitrary* than convolutional layers. That is, arbitrariness can easily be conflated with an absence of constraint. This is incorrect. A GNN layer still constrains the direct relationships of its inputs to a local neighbourhood *by design*. However, this neighbourhood is explicitly encoded by the edges between the nodes of the graph rather than by a fixed frame of reference like a grid. This means that the local neighbourhood of a node (e.g., object, pixel) can be defined in terms of the associations encoded in the edges that connect it to other nodes (e.g., objects, pixels). This allows the domain to inform what relationships are acceptable. For example, a dog (node 1) might be associated to a bone (node 2) and a bowl (node 3) in an image based on their label semantics. For each of these associations, an edge is connected between the pair of nodes (e.g., a dog-bowl edge, a dog-bone edge). Thus, the local neighbourhood of the dog contains three nodes and two edges. Depending on the GNN layer, either all or some subset (e.g.,

[3]There are exceptions. For example, Gorodissky, Harari, and Ullman (2018) describe a variable sampling approach for CNNs.

just the nodes) might be used to compute the resulting output of the layer. One of the harder challenges for GNNs is coming up with an implementation that satisfies the three requirements I outlined above. Luckily, other researchers have already solved this problem and GNNs have been successful in almost every learning paradigm (e.g., supervised, unsupervised, semi-supervised, reinforcement) and across many domains, including visual segmentation and classification (for an extensive review of GNNs, see Battaglia et al., 2018).

Although there are many different formulations of GNNs that are easily as varied as their many different use cases, all of them have two essential properties. First, they use distributed representations for the nodes, edges, and attributes—or whatever subset of these elements are relevant—for the domain. This usually means that the nodes, edges, or attributes are encoded as vectors of continuous values that are output from earlier computations of the GNN layer or, occasionally, neural network layers. The three most common initialization strategies for these distributed representations are by random sampling from a distribution, by computing over their attributes (e.g., classes or labels), and directly based on the content of the domain (e.g., pixels in an image). Second, they all include a permutation invariant *aggregator function*. I will first outline what an aggregator function is before addressing permutation invariance.

An aggregator function is a function that accepts a variable number of inputs (e.g., vectors of continuous values) and returns an output (e.g., a vector of zeros and ones, a vector of continuous values) that summarizes the inputs. Common aggregator functions in the literature include an element-wise sum, an element-wise average, and an element-wise maximum among others. What is special about all of these aggregator functions is that they 1) can compute over an arbitrary number of inputs, 2) are insensitive to the relative position of the inputs, and 3) do not require a grid to index what inputs need to be aggregated. These are exactly the three differences that I outlined between a convolutional layer and the GNN layer. Thus, the aggregator

function is the essential ingredient that enables a GNN layer to compute over arbitrary neighbourhoods as I defined them above. Before I move on, the second property warrants a little more attention.

Of the three key differences between GNN layers and convolutional layers, the most important difference is its insensitivity to the relative position of its inputs. This difference directly contributes to the core invariance for GNN layers—what is often called *permutation* invariance (Battaglia et al., 2018; Zaheer et al., 2017). Permutation invariance means that the layer will produce the same output for every possible permutation of the input nodes.[4] The commutativity (i.e., $f(a, b) = f(b, a)$) of the sum, average, and maximum aggregators produces the required permutation invariance. For example, if the object dog is represented by the vector [1, 0, 0], the object bone is represented by the vector [0, 1, 0], and the vector bowl is represented by [0, 0, 1], then the sum and maximum aggregators will produce the vector [1, 1, 1] while the element-wise mean will produce [1/3, 1/3, 1/3].[5] It does not matter what order these objects take (e.g., [dog, bone, bowl], [bowl, dog, bone]), each aggregator produces the same result for every permutation.

By contrast, the kernel of a convolutional layer is dependent upon the order of the inputs. Thus, for example, if the vectors are represented as stacked column vectors in a 3x3 matrix, the 2x2 kernel [[1, 0], [0, 0]] with a step size of 1 will produce the vector [[1, 0], [0, 1]] for [dog, bone, bowl] and will produce [[0, 1], [1, 0]] for [bone, dog, bowl]. These two outputs are not equal element-wise. Thus, the convolutional layer is not permutation invariant.

To summarize, the GNN layer generalizes the convolutional layer by defining arbitrary local neighbourhoods for each node in the graph. These local neighbourhoods are arbitrary in exactly three ways: they can be of variable size, they are invariant

[4] A formal proof of the necessary properties for permutation invariance is defined by Zaheer et al. (2017).

[5] I am using simple vectors like [0, 0, 1] for pedagogical reasons rather than to indicate that the vectors need to be categorical or one-hot.

to permutations of their nodes, and they do not require a grid to index their position relative to other local neighbourhoods. The GNN layer achieves these three properties through the use of aggregator functions (e.g., element-wise sum, average, maximum) that summarize the input to the layer in a permutation invariant way. The GNN layer constrains direct relationships between its inputs by controlling what inputs are acceptable to the layer—mainly, other nodes that are connected to the source node via an edge. This allows the domain to inform what direct relationships are acceptable via the local neighbourhoods defined by the edges of the graph. Battaglia et al. (2018) describe the GNN layer's relational inductive bias (RIB) as an *arbitrary* RIB to emphasize the arbitrary local neighbourhoods GNN layers can compute over. My proposed solution to adversarial robustness builds upon the arbitrary RIB and related permutation invariance of GNNs. However, before getting to the details of my solution, I first need to define the problem.

1.4 Adversarial Robustness

One of the core hypotheses of my research is that adversarial robustness necessitates a stronger relational inductive bias (RIB) than the graph neural networks (GNNs) discussed in the previous section. In order to make sense of this claim, I need a more precise definition of adversarial robustness.

Papernot, McDaniel, Wu, Jha, and Swami (2016) define the notion of adversarial robustness in terms of "adversarial gradients." They observe that many of the early methods for producing adversarial examples exploit the gradient of a neural network's output (e.g., its classification error) with respect to its input. These gradients are used to estimate the network output's sensitivity to the individual pixels of an image. They call these gradients "adversarial gradients" to emphasize their role in the creation of adversarial examples by these attacks. When adversarial gradients

are large, the network sensitivity is high and small perturbations to the input pixels produce large changes to the output predictions. Adversarial robustness can then be informally defined in counterpoint to these adversarial gradients: the network's "capability to resist perturbations" to its input (Papernot et al., 2016). That is, a robust network should 1) have good accuracy and 2) classify inputs consistently within the neighbourhood of a real sample from the domain (i.e., image).

Papernot et al. (2016) define the neighbourhood of a sample image as all of the other images that are within a specified distance of the original sample (i.e., image). To compute this, one selects a norm (i.e., notion of length or magnitude) with which to compare different input images. The Euclidean or L_2-norm is a common representative that is used in practice, but the L_0 and L_∞ norms are also used. I discuss all three in more detail in Chapter 3. Adversarial robustness is then measured by evaluating the average size of the neighbourhood around a set of input samples over which the output of the network is constant or near constant (for an even more formal treatment of this definition of robustness, see Fawzi, Fawzi, & Frossard, 2018). The larger the size of the neighbourhood, the more robust the model.

One of the challenges of this definition is that it is severely limited by the techniques used to produce samples within the chosen neighbourhood—in other words, by the adversarial attacks that are applied to evaluate robustness. For example, Papernot et al. (2016) originally proposed a form of network distillation to improve robustness. Their technique trains a simpler neural network based on the output of a more complex teacher network that has good accuracy on the classification problem. Despite having a larger neighbourhood on average around each sample in the test set and thus smaller adversarial gradients—indicating a more robust network than previous models—subsequent research showed that a simple trick that divides the logits (i.e., the pre-softmax output) of the network by a constant entirely mitigates this robustness (Carlini & Wagner, 2017).

This situation, where a more robust model is identified only to be invalidated by subsequent research, is so common that many of the central figures in the field have constructed a living document to encourage better research practices (Carlini et al., 2019). Critically, the authors propose a new mindset that is essential for working in the domain. In this mindset robustness status is awarded to models in an ad-hoc fashion after a concerted effort has been made to break the model. Although I am sympathetic to the authors' views and have implemented many of their proposed suggestions, I do not believe that this approach and mindset will ever help researchers understand what ultimately produces adversarial robustness in systems that have it (e.g., the human visual system). In fact, this approach and mindset directly conflicts with visual nativism by virtue of being enumerative. In other words, it proposes to define robustness as an ever-expanding set of as yet-to-be-defined grammars instantiated in the form of L_p norms and associated adversarial attacks.

What I propose in the research that follows is a definition of adversarial robustness as a constraint imposed on a sequential set of three neural network layers that restricts the terminal adversarial gradients of the network by leveraging a non-linearity that approximates the unit step function.

1.5 Asymptotic Hierarchies

I call my relational inductive bias (RIB) the asymptotic hierarchical RIB and the models that implement it asymptotic GNNs. The model and bias get their name from their use of a non-linearity with a vertical asymptote—a function that approaches infinity in finite time as the input decreases or increases (but not necessarily as it increases *and* decreases). The behaviour of this non-linearity is discussed in detail in the next chapter. However, the unique property of the asymptotic layer relative to the

GNN layer is that it constrains the gradients of the layers preceding the asymptotic layer in the network. One can think of this as a constraint preventing relationships between separated *levels* in a hierarchy. It is worth discussing an example to clarify what I mean.

Consider an image represented as a graph. One way to represent the image is with the pixels as the nodes. This was the representation I used to explain the differences between convolutional and GNN layers in Section 1.3. An alternative representation is to think of patches of contiguous pixels as the nodes, while the individual pixels are the substructure of those nodes (step 2 in Figure 3; green brackets). This substructure is a vector of pixels—which could be flattened from, for example, a square image patch—combined with a neural network. I will reserve the discussion of the structure and implementation of this network until Chapter 2. This substructure neural network (green arrows in Figure 3) outputs a distributed representation (e.g., vector of continuous values) of the associated node's state (base level; step 3 in Figure 3; rose brackets), which is slightly more complex than the examples from the previous section.

Figure 3: A visualization of an example layout of the asymptotic GNN. The first step (blue) is the initial image and the blue arrows are the initial pre-processing. The second step (green) is the substructure level and the green arrows are the substructure processing. The third step (rose) is the node state at the base level. Finally, the fourth step is the superstructure level (i.e., the final class prediction; dark yellow) and the dark yellow arrow is the combined asymptotic layer and aggregator function.

In order to incorporate the class of the image within the hierarchy of the model, the asymptotic GNN includes an additional *super*structure level (step 4 in Figure 3; dark yellow brackets). I define the superstructure level of an asymptotic GNN as a GNN layer that aggregates all of the image patch nodes to produce the final class output for the image (dark yellow arrow in Figure 3). This output is represented by the state of the superstructure node. Together, the substructure, base, and superstructure levels

make up the simplest hierarchy implemented with asymptotic GNNs (see Figure 3).

This 3-level hierarchy allows me to define the invariance associated with the asymptotic hierarchical RIB. I call this new invariance *substructure* invariance to emphasize that the superstructure is invariant to gradients from the substructure level *but not gradients from the base level*. Said another way, the network should be sensitive to changes in the state of the base level nodes and insensitive to any change in the substructure that does not produce a change in the state of the base level nodes. This simplifies the problem of adversarial examples by explicitly specifying where within the network architecture sensitivity is acceptable and where within the network architecture it is unacceptable.

For example, consider a network where the base level nodes can only be in two states: on or off. Any time there is a change to the state of one of the *base level nodes*, there is a corresponding change to the superstructure level (e.g., class prediction). Thus, there is a meaningful gradient from the base level nodes to the superstructure level. By contrast, the substructure level can only affect the superstructure level by changing the states of the base level nodes. Since there are far more than two states represented by the substructure (by design), the mapping between substructure and base level nodes cannot be one-to-one. That is, many changes to the pixels will not change the state of any of the base level nodes. Any change to the substructure that does not change the state of the nodes will not have gradients (by design). I then assume that any change to the substructure that does change the state of the nodes is meaningful *by design*. Thus, their gradients express sensitivity in the network that *must be there*. In sum, this means that the entire network no longer needs to be considered when evaluating the adversarial gradients. The substructure can be abstracted away so long as one can make the gradients between the substructure and base levels approach zero.

This perspective changes the nature of the problem of adversarial examples. It is

no longer a task of enumerating the adversarial attacks and norms that the network is robust to. Instead, it is the job of the researcher to find ways to engineer the nodes into meaningful organizations that capture natural categories within their substructure. As I stated above, when a node changes state as a result of a change in its substructure, that change should be meaningful *by design*. If it is not, then the node is too expansive: it includes too many changes (ultimately resulting in adversarial gradients) that are not relevant to the state change described by the node. One way to solve this is to make the node define an even more complex, non-linear hyperplane by, for example, increasing the representational power of the node's subnetwork and augmenting its training data (i.e., one of the currently accepted approaches that is used by Madry, Makelov, Schmidt, Tsipras, & Vladu, 2018 and discussed in Chapter 3). A much simpler approach that I am proposing here is to take that node and sub-divide it into another 3-level hierarchy like the one I described above. That is, one reduces the expression of the *node's state* to a set of sub-nodes. These sub-nodes have their own set of internal states that the node is sensitive to by design. This process can be repeated until the base domain is a simple separation of relevant and irrelevant patterns. The result is a compositional hierarchy with well-defined and intentionally selected regions of sensitivity and insensitivity.

Building the entire system that I described above is beyond the scope of the current project. It is also unproductive from a research perspective as too many decisions would need to be made at once. Instead, I have chosen to implement the minimal, 3-level hierarchy as a first step towards constructing and evaluating this larger system. However, it should be clear from my description above that this test will ultimately fail as there will be gradients within the substructure that cannot be abstracted away without further sub-nodes. This is a necessary failure as it will enable me to assess the important elements of the design that enable the desired functionality of my proposed system. Additionally, although I was critical of the existing literature on adversarial

examples, I still need some way to assess whether the gradients of my proposed system lead to adversarial examples. A pure mathematical approach to this assessment would be preferred, but is beyond the scope of the current project. As a consequence, I will continue to use the adversarial attacks as a weak proxy for the sensitivity of the network that points in the direction of future mathematical research that is likely to be productive. However, it is absolutely essential to recognize that my approach is in direct opposition to the enumerative approach that I described in the previous section. A longer list of adversarial attacks that my system is robust to will in no way advance my proposed research program beyond this initial suggestion of future productive research. With these project details in hand, the question that remains to be answered is how the asymptotic layer constrains the adversarial gradients between the sub- and superstructure levels.

The asymptotic layer of the asymptotic GNN is composed of two critical functions, which I call the asymptotic barrier and the learning branch. The asymptotic barrier is a function that constrains the gradients between the sub- and superstructure levels by approximating the unit step function via a tunable hyperparameter. This allows the base nodes to function like they have a finite, discrete set of internal states. The learning branch enhances the functionality of the asymptotic barrier by directly parameterizing the size of the gradients deriving from variation within the members of a given class. Together, these functions reduce the sensitivity of the superstructure to the substructure level while preserving its sensitivity to the base level nodes. The next chapter discusses the structure and dynamics of the asymptotic layer in detail.

1.6 Outline

Over the next four chapters, I propose and evaluate the first theory and computational model that explicitly describes the cognitive constraints—in the form of rela-

tional architectural constraints—that produce adversarial robustness. In Chapter 2, I provide a detailed description of the model and implementation of my theory of the asymptotic hierarchical relational inductive bias (RIB) and its associated substructure invariance. Chapter 3 provides a baseline comparison between my proposed model and a traditional convolutional neural network that is competitive on the MNIST dataset—a dataset of black and white images of the digits 0 through 9 (LeCun et al., 1998). Although MNIST is a simple dataset, current state-of-the-art (SOTA) deep learning models are still sensitive to simple adversarial examples of it. I use a collection of SOTA adversarial attacks and show that, for every evaluation of adversarial robustness I consider, the asymptotic GNN achieves higher scores than at least one of the other models. In Chapter 4, I empirically evaluate the essential structures of the asymptotic layer: the asymptotic barrier and the learning branch. The fifth chapter explores one of the SOTA adversarial techniques—the PointWise Attack (PWA)—in greater detail in order to better understand why the asymptotic GNN has minimal effect on the adversarial examples produced by this technique. The sixth and final chapter summarizes my findings and outlines future work.

My primary claim is that the asymptotic hierarchical RIB enables a form of adversarial robustness by constraining the adversarial gradients between the sub- and superstructure levels in a 3-level hierarchy. My key contributions are as summarized below.

1. A new theory and approach to adversarial robustness as a well-defined organization of regions of sensitivity and insensitivity that results in substructure invariance.

2. An improved model and implementation of adversarial robustness on the MNIST dataset.

3. A new RIB, new form of invariance, and new non-linearity for artificial neural

networks.

4. An implementation of the asymptotic barrier and learning branch functions.

Various secondary contributions that were not selected for but emerge as a consequence of the core theory are also highlighted throughout the work. The next section provides a detailed description and analysis of the essential structures of the asymptotic GNN and its associated asymptotic layer.

Chapter 2

Model and Implementation

I proposed that the asymptotic hierarchical relational inductive bias (RIB) and its associated substructure invariance are the missing ingredients required to achieve adversarial robustness. Chapter 1 outlined a definition of adversarial robustness as a constraint imposed on a sequential set of three neural network layers that restricts the terminal adversarial gradients of the network by leveraging a non-linearity that approximates the unit step function. In this chapter, I provide a detailed description of my proposed model and implementation. I begin with a general description of the core components of asymptotic GNNs. I then describe my asymptotic barrier as an approximation of the unit step function, which makes the adversarial gradients between the substructural and base levels approach zero. Once I have explained the basic behaviour of my model, I outline my formulation of the asymptotic barrier and show how its behaviour approximates three essential properties of the unit step function. The next section discusses the learning branch of the asymptotic layer followed by the implementation details of a simple instantiation of the asymptotic GNN.

2.1 Asymptotic GNNs

The core theory of my proposed model is that adversarially robust visual representations can be modeled by asymptotic Graph Neural Networks (GNNs). One of the essential functions of these networks is the asymptotic barrier (Γ), which uses a non-linearity with a vertical asymptote to make the adversarial gradients between a node (e.g., image patch) and its substructure (e.g., pixels) approach zero. I explain this mechanism along with the supporting learning branch later in the chapter. I describe the general organization of the information processing of the asymptotic GNN below.

The asymptotic GNN requires three distinct, hierarchical levels to meet the minimal requirements of its definition: a base level (V), a substructure level (S), and a superstructure level (C). One can define the resulting graph as a 4-tuple $G = (V, S, C, E)$. The $V = \{\boldsymbol{v}_i\}_{i=1:N^v}$ is the set of nodes and the size of the set is N^v. Each $\boldsymbol{v}_i$ is a vector encoding the state of the i^{th} node in the graph. In my example from Chapter 1, each $\boldsymbol{v}_i$ represented an image patch from a different location. The $S = \{\boldsymbol{s}_i\}_{i=1:N^v}$ is the set of substructures mapped by their index to their corresponding node $\boldsymbol{v}_i$. Each $\boldsymbol{s}_i$ is a vector of continuous values. In the example, these vectors represented the individual pixels within each image patch. The $C = \{\boldsymbol{c}_j\}_{j=1:N^c}$ is the set of superstructure nodes and the size of the set is N^c. Each $\boldsymbol{c}_j$ is a one-hot vector encoding an attribute about the entire graph G. In the previous example, I let $N^c = 1$ and let $\boldsymbol{c}_1$ represent the final output class for the entire image. Finally, E represents the edges between base nodes and the superstructure node. However, I focus on graphs where an edge exists between every base node and $\boldsymbol{c}_j$ (i.e., the graph is fully-connected). This means that every node is within a single neighbourhood that is permutation invariant. Consequently, I use the simpler graph representation $G = (V, S, C)$ throughout the rest of this research as the edges are uninformative. More complex edge layouts are left for future work.

In addition to this basic graph structure, asymptotic GNNs require the use of

three types of invariance: a weak or weaker invariance of a traditional neural network layer type (e.g., fully-connected, convolutional, recurrent), permutation invariance, and substructure invariance. The weaker invariance is isolated to the substructure level, the permutation invariance operates over the base level, and the substructure invariance operates over the superstructure level. It is this combination of 3-layer graph structure combined with this organization of RIBs that uniquely characterizes asymptotic GNNs as I have defined them. However, this structure is designed to be modular and stacked asymptotic GNNs, as I discussed in the previous chapter, are an active area of my current and future research.

The minimal requirements to construct an asymptotic GNN are as follows. The substructure level is instantiated with a fully-connected neural network (ϕ) that accepts the pixels of an image patch ($\boldsymbol{s}$) as input and it outputs the state of its associated image patch node ($\boldsymbol{v}$). I will leave the complete definition of ϕ to Section 2.7.1. Each node's state ($\boldsymbol{v}$) is represented as a probability distribution over each of the K image classes. The probabilistic vector values represent the relative *mis*match of the substructure of the node to the set of available classes in the superstructure node.[1] This will be explained in detail in Section 2.5.

At this point in the network, the output probabilistic vectors of the nodes are sensitive to the substructure values. In order to make sure that the superstructure node (output class prediction) is not sensitive to the substructure, one needs to make the adversarial gradients approach zero. My proposed method for asymptotic GNNs is to use an asymptotic barrier function (Γ) that I explain in detail in the next section. The output of Γ is the barrier node vector ($\boldsymbol{v}^{\Gamma}$)—the last representation that is sensitive to all of the variations in the substructure. After the asymptotic barrier, the adversarial gradients from the substructure approach zero and the nodes become graph-like as I describe below.

[1] The probability of a match is equal to $\mathbf{1} - \boldsymbol{v}$.

Recall that GNNs uniquely capture associations within arbitrary local neighbourhoods of nodes. To produce permutation invariance, all of the nodes within a single neighbourhood are aggregated via a permutation invariant function (e.g., element-wise sum, element-wise mean, element-wise maximum) into a single vector that summarizes the local neighbourhood (Battaglia et al., 2018). The summary of the neighbourhood is permutation invariant because the aggregation function computes the same output for different orderings of the input (i.e., it is commutative and $f(a, b) = f(b, a)$). For example, the element-wise sum of a collection of vectors produces the same output for every permutation of the input vectors. In a sense, the asymptotic GNN resembles a traditional GNN after the asymptotic barrier and with application of the aggregation function.

In the models that I consider in this work, every pair of base nodes are assumed to be connected. This means that they are a part of a single local neighbourhood that encompasses all of the base nodes. Consequently, the GNN aggregation step produces a single, summary barrier vector ($\overline{\boldsymbol{v}}$) summarizing all of the base nodes in G. This vector represents the class of the image. As I discuss in the next section, I have chosen to use the product function ($\prod$) to aggregate the barrier node vectors in order to facilitate the asymptotic barrier function. This operation is similar to the product in sum-product networks discussed by Poon and Domingos (2011) and convolutional arithmetic circuits discussed by N. Cohen, Sharir, and Shashua (2016).

The product is a permutation invariant function as a consequence of its commutativity (i.e., $a \cdot b = b \cdot a$).[2] This is a new contribution to the set of GNN aggregator functions discussed in the literature.[3] It also motivates my description of the model

[2]For the interested reader, Zaheer et al. (2017) provide a proof that a function is permutation invariant if it has the form $\rho(\sum_{x \in X} \phi(x))$. One can trivially map a product to this form by mapping an exponential into ρ (resulting in ρ_e) and the natural logarithm into ϕ (resulting in ϕ_{ln}) as follows: $\rho(\prod \phi(x)) = \rho(e^{\ln(\prod \phi(x))}) = \rho(e^{\sum \ln(\phi(x))}) = \rho_e(\sum \phi_{ln}(x))$.

[3]Since the product is such an elementary commutative function, I expect that it actually has been tried before by other GNN researchers. What is unique about it is its incredible instability with gradient-based learning techniques. Consequently, I would speculate that my actual contribution is the specification of a mechanism that allows for the stable use of products as aggregator functions.

as an asymptotic *GNN* instead of merely an asymptotic non-linearity—the asymptotic barrier function (Γ) and the GNN aggregator ($\prod$) are tightly coupled in my proposed system. For example, in preliminary experiments, replacing $\prod$ with any of the standard aggregator functions results in a model with very poor accuracy if it works at all.

The final step in this system is what I call a learning branch. This final function (Ω) incorporates a parameter (τ) that reduces the impact of adversarial gradients from within-class variation over the course of training the network. It is defined as follows. Unlike many similar 'temperature' parameters, the value of τ in the learning branch is actually learned by the system. That is, it exploits a natural dynamics in the system where the prediction accuracy improves as a consequence of reducing these adversarial gradients. This introduces a critical learning period during which the system can easily learn new things but is more susceptible to adversarial examples. It also means that the model is much more robust to adversarial examples after it exits its critical period (i.e., τ is at its maximum value), although it is also unable to learn new things. Thus, the learning branch function is another novel contribution of my approach that provides an important constraint that drives the desired effect (i.e., adversarial robustness).

After the learning branch, the state of the final class node approximates a categorical (i.e., one-hot) distribution over the available classes. In the next section, I begin developing a framework to understand how the asymptotic barrier function works.

2.2 A Framework For The Asymptotic Barrier

In Chapter 1, I proposed a new model—the asymptotic GNN—that extends the basic structure and permutation invariance of GNNs with a stronger asymptotic hierarchical

This is another example of the tight coupling between the asymptotic barrier and GNNs that I discuss below, as the stability emerges as a consequence of the asymptotic barrier function. However, as an observation that is lateral to the core of my thesis, I leave the proof of this point to future work.

relational inductive bias (RIB). The unique property of the asymptotic GNN and its associated asymptotic layer is that it makes the gradients between the substructural level and base level approach zero. In other words, the asymptotic barrier enables the barrier nodes ($\boldsymbol{v}_i^{\Gamma}$ for $i = 1 : N^v$) to behave like regular GNN nodes.

One of the essential components of the asymptotic GNN is the asymptotic barrier function. The asymptotic barrier exploits a non-linearity with a vertical asymptote to make the superstructure insensitive to variations in the substructural level while maintaining sensitivity to the base level. The result is what I call substructure invariance. I have chosen to analyze the behaviour of the asymptotic barrier function (Γ) in terms of a series of increasing approximations to the unit step function $H(x, \delta)$. However, I include the full equation for Γ below as a reference, the details of which will be explained over the course of this chapter.

$$\Gamma(\boldsymbol{v}, \psi, \kappa) = (-1)^{\psi} \cdot \ln^{\psi}(\boldsymbol{v} + \kappa) \tag{2.1}$$

where $\boldsymbol{v}$ is a probabilistic node vector for a single image patch, and ψ and κ are hyperparameters that are summed or exponentiated element-wise.

The unit step function, $H(x, \delta)$, is a good representation of the perfect *match* detector for a single class represented as a scalar value (Figure 4). I focus on the match detector instead of the mismatch detector from Chapter 1 to simplify many of the equations in this section. The analytic form of the unit step function shifted to the right by $1 - \delta$ is defined as follows (Venetis, 2014).

$$H(x, \delta) = \frac{1}{\pi} \cdot \left[\frac{3\pi}{4} + \arctan(x - 1 - (1 - \delta)) + \arctan\Big(\frac{(2 + (1 - \delta) - x}{x - (1 - \delta)} \Big) \right] \tag{2.2}$$

where x is in radians and δ is a small number (e.g., $\delta = 10^{-4}$).

When $H(x, \delta)$ is computed, the function is insensitive to changes in its input ev-

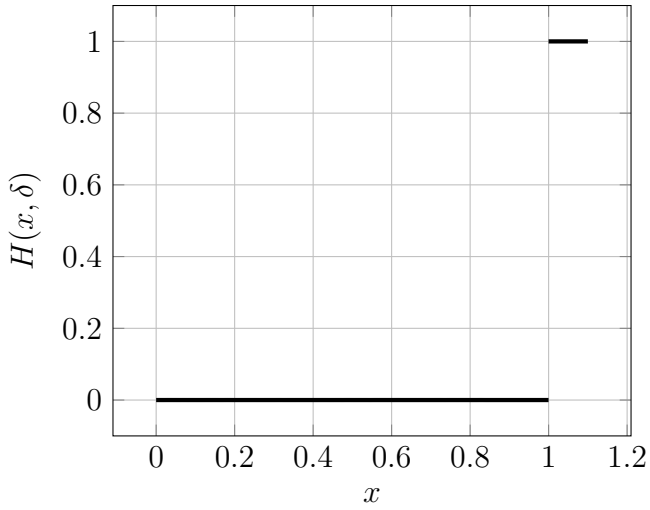

Figure 4: A graph of the unit step function $H(x, \delta)$ that approximates a perfect *match* detector with a continuous input space. The constant output over [0, 0.9999] together with the discontinuity at 0.9999 perfectly encapsulates the input values (x). Thus, $H(x, \delta)$ is a good model of the desired behaviour for substructure invariance. Problematically, the gradient of $H(x, \delta)$ is ill defined.

erywhere except at 0.9999. At 0.9999, the rate of change (i.e., gradient)[4] is undefined. This behaviour of the unit step function enables the strongest form of substructure invariance as the adversarial gradients are zero or undefined everywhere. Problematically, the jump discontinuity of $H(x, \delta)$ makes it intractable for gradient-based learning. Consequently, my asymptotic barrier will approximate the behaviour of the unit step function without introducing this discontinuity (i.e., by preserving the gradient). The rest of this section will try to characterize the behaviour of the unit step function in more detail.

In the general case defined by the asymptotic GNN the full range of the behaviour of $H(x, \delta)$ is quite complex. Consequently, I simplify the problem space to make the analysis easier to explain. A full formal analysis of the behaviour of the asymptotic barrier is left for future work.

[4]Technically, the rate of change is a scalar value so it is not really a gradient. However, for simplicity's sake, I will refer to the both the scalar and vector representations of the rate of change as the "gradient." I will rely on the context to differentiate which I am talking about (e.g., if the quantity is a bolded variable then I mean the traditional definition of a gradient).

Assume that there is only a single class to consider that is represented by a single scalar value. I will also assume that the node (v_i for $i = 1 : N^v$) and class output values (c) are scalar in the range [0, 1]. There are only $N^v = 4$ image patches in this simplification. I will represent the barrier node as v_i^Γ. Note that this means that, although v_i is a node, its values can still be affected by its substructure ($\boldsymbol{s}_i$). I will define a new term (ϵ) to be the change that is introduced to the value of *a single* v_i without changing the state of the corresponding barrier node v_i^Γ and hence needs to be zeroed by Γ. On the one hand, in the absence of the asymptotic barrier or with an inappropriate barrier, ϵ will directly affect c. On the other hand, the closer the barrier behaves to $H(x, \delta)$—in other words, the less ϵ affects c—the better my approximation of $H(x, \delta)$.

For my first approximation, I will let $v_i^\Gamma = H(v_i, \delta)$ and the summarized barrier node ($\bar{v}$) equal $1/N^v \sum_{i=1}^{N^v} v_i^\Gamma$. That is, the barrier is the unit step function and the aggregation function is the element-wise mean. There is no Ω mechanism in this approximation so c is equal to $\bar{v}$.

Instead of allowing all of the patches to vary independently, I will focus exclusively on the case when $v_i = v_j$ for all $i, j \in \{1, \ldots, N^v\}$. Although this simplification is very restrictive, it only minimally reduces the scope of my claims. To understand why, one only needs to realize that the output of the aggregator for variable input (i.e., when $v_1, \ldots, v_{N^v}$ are not equal or only some of them are equal) is equal to the output of the aggregator for some constant output for most aggregation functions. For the aggregator defined above, the average value (μ) of the nodes (v_i) will produce the same output.

$$c = \mu = \frac{1}{N^v} \sum_{i=1}^{N^v} v_i^\Gamma = \frac{1}{N^v} \sum_{i=1}^{N^v} \mu \tag{2.3}$$

This enables one to describe how the change (ϵ) in a single v_i affects c as a relationship between only two values: μ and ϵ. This relationship is much easier to understand as there are less moving parts than in the more general case. Consequently, to an-

alyze how ϵ affects a single μ, one simply expands the sum for one of the μ values, proportionally splits the weighting, and then adds ϵ to the initial value of μ.

$$c = \frac{1}{N^v}(\mu + \epsilon) + \frac{N^v - 1}{N^v}\mu \tag{2.4}$$

My approximation using μ can under- and over-estimate the acceptable range of ϵ for values of v_i close to their bounds. However, this is not really a concern as the focus is on the coarse grained behaviour of the function with small values of ϵ. Figure 5 provides a visualization of this equation.

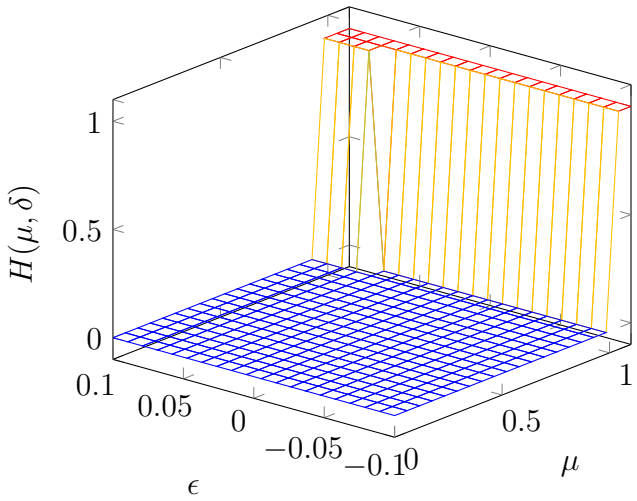

Figure 5: A graph of $H(\mu, \delta)$ when $\delta = 10^{-4}$ and $N^v = 4$. Critically, ϵ has no effect on $H(\mu, \delta)$ except when it crosses the discontinuity in H. Although ϵ can cause the discontinuity to occur slightly earlier, the gradient is otherwise flat. This is the behaviour that my asymptotic barrier function is trying to approximate. The transition in the graph across the discontinuity (vertical lines) is there to help with visualizing the behaviour. It is not representative of the actual function, which has a jump discontinuity along the boundary.

The key behaviour of this function with respect to substructure invariance is the limited effect of ϵ on c. A change in μ of a single image patch by ϵ will only cause a change in c if $\mu + \epsilon$ crosses $1 - \delta$. In every other situation, ϵ has no impact on c as almost all of the values are mapped to 0. If there is an output of 0 (or technically any constant) for a large set of input values, then changing those input values *cannot*

have an effect on c. Visually, this gets represented as a *flat* region in Figure 5 whose gradient is locally 0—a key topological property that I will discuss in a later section on the learning branch. This is a key behaviour of the unit step function that I want to capture in my asymptotic barrier function. I will call this behaviour the zero gradient (ZG) condition to highlight its significance throughout the rest of this section.

Further consideration of $H(x, \delta)$ reveals more behaviour that directly contributes to the constraint on ϵ. When a change in c occurs, the value of c jumps from 0 to 1 in a single (discontinuous) step. It is never proportional to ϵ as the gradient is undefined at that point. Consequently, even if one knows the exact position of the discontinuity *and* one knows perfectly the current state of all v_i for $i = 1 : N^v$—and neither of these things are easy to deduce in the general case—then ϵ can only communicate a single thing: one of the values changed by at least $(1 - \delta) - \mu$. This is what I want the asymptotic barrier to approximate.

To generalize this behaviour so that it functions on more of a continuum, which will allow Γ to approximate it, one can think of the discontinuous transition as an infinitely small change leading to an infinitely steep gradient. There are two key conditions to achieve this: the size of the window where the change occurs must be as small as possible and the steepness of the change must be as large as possible. Although there are interesting dependencies between these two conditions, I will ignore these dependencies for the sake of clarity. Thus, to better approximate $H(x, \delta)$ one wants to approximate both of these conditions. I will call these conditions the small transition window (STW) and large transition gradient (LTG) conditions.[5]

I now have a much clearer path for approximating the unit step function. To evaluate two candidate functions, one can compare them across each of the three approximation conditions: the zero gradient (ZG), small transition window (STW),

[5]The LTG could be explicitly measured using the second derivative. However, I have chosen to use a less formal analysis of this condition in an effort to establish a more intuitive grasp of the main concepts.

and large transition gradient (LTG) conditions. Although it is not clear which of these are more important relative to the others, if one function is better than another on all of them, then it is a better approximation of $H(x, \delta)$. To situate this style of analysis, I will now consider a very naive 'solution' to a major problem of the unit step function for gradient-based learning.

The problem with using $H(x, \delta)$ is that the jump discontinuity is intractable for gradient-based learning. But, if one removes the discontinuity introduced by the unit step function (i.e., by replacing v_i^Γ with v_i in Equation 2.3), one loses the desired invariance to the effect of ϵ on a single image patch (v_i; see Figure 6). One can evaluate that this is true by examining each of the approximation conditions.

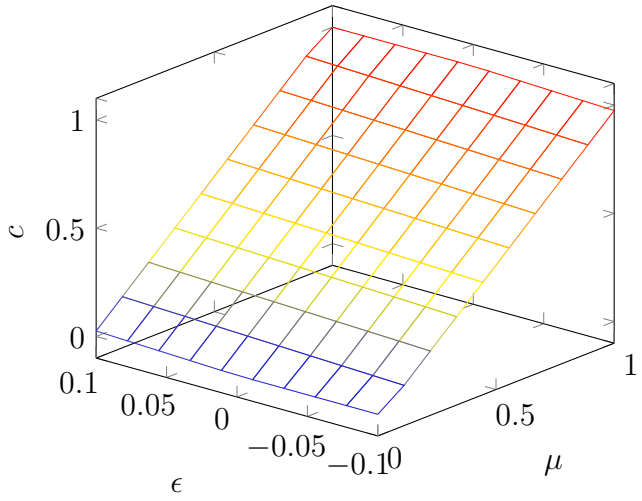

Figure 6: A graph of $c = \frac{1}{4}(\mu + \epsilon) + \frac{3}{4}\mu$. Changes to μ via ϵ can now directly change the value of c by $1/N^v$ at every point. Consequently, removing the unit step function also removes the desired substructure invariance and the related three approximation conditions: ZG, STW, and LTG. This is an example of the opposite of the desired behaviour for Γ. It is also the standard behaviour of a traditional GNN.

By naively removing $H(x, \delta)$, each of the approximation conditions becomes much worse. First, for the ZG condition, the removal of $H(x, \delta)$ causes the entire domain to be mapped to values greater than 0 except when the input is 0. In effect, removing $H(x, \delta)$ entirely removes the ZG behaviour of the function. Second, for the STW condition, the transition window now encompasses the entire range of values. That

is, the transition window is now at its maximum size. Consequently, removing $H(x, \delta)$ has entirely removed the STW behaviour of the function. Finally, the LTG condition is also at its worst state (i.e., the minimum of the set of valid gradients that respect the required bounds). If the gradient were smaller at any point, the function would not reach its upper bound. Consequently, removing $H(x, \delta)$ also entirely removes the LTG condition.

There are two key points that I want to conclude from this result. First, my three conditions are good indicators of the behaviour of $H(x, \delta)$ as the removal of $H(x, \delta)$ drives each of them to their worst (or opposite) state. Second, when one removes $H(x, \delta)$ one arrives back at the architecture of a traditional GNN. Thus, the traditional GNN acts as an excellent counter-point to $H(x, \delta)$. It defines the other end of the continuum of substructure invariance, which supports my hypothesis that standard GNNs are not sufficient for achieving this property.

At this point it is worth summarizing what I have accomplished thus far. First, I highlighted what my contribution was: a modification to GNNs that enables them to model substructure invariance via stronger architectural constraints. Second, I highlighted two major challenges of my approach: adequately characterizing substructure invariance and the intractable gradient of a perfect match-mismatch detector. Third, I simplified the problem domain so that I could adequately characterize the problem in terms of a set of constant node vectors (μ) where $v_i = v_j$ for all $i, j \in \{1, \ldots, N^v\}$ and a change (ϵ) to one of the node's values. This allowed a formulation and visualization of the ideal behaviour as a function that takes the element-wise mean of the unit step function ($H(v_i, \delta)$) and the worst-case behaviour as a function that takes the element-wise mean of v_i (i.e., the standard GNN behaviour). And fourth, I characterized the difference of these two functions in terms of three key approximation conditions: the zero gradient (ZG), small transition window (STW), and large transition gradient (LTG) conditions. Consequently, the unit step function and the

standard GNN behaviour provide a continuum of good to bad, respectively, behaviour for the asymptotic GNN.

One simple and small step to better approximate the unit step function (i.e., move from the bad, basic GNN approximation towards a better model of $H(x, \delta)$) is to change the aggregator function from the element-wise mean to my proposed product-based aggregator from the previous section.

2.3 The Product Aggregator

The product has a bunch of useful properties for capturing the behaviour I am interested in modeling. First, it does not need to be re-weighted to keep $\boldsymbol{c}$ within the bounds of 0 and 1. The product of N^v values between [0, 1] is also within [0, 1]. Thus, one does not need to include either $\frac{1}{N^v}$ or $\frac{N^v-1}{N^v}$ from Equation 2.4 to make sure that c is bounded by 0 and 1 when using a product aggregator. By contrast, the element-wise mean aggregator needs additional terms ($\frac{1}{N^v}$ and $\frac{N^v-1}{N^v}$) to keep the output within these bounds. In the absence of these additional terms, the effect of ϵ on c is even worse than the current basic GNN as each change in epsilon would result in an equal change to c. Instead of a $1 : \frac{1}{N^v}$ effect on c one gets a $1 : 1$ effect. Any change resulting from epsilon would pass directly on to the values of c (see Equation 2.4). Thus, the aggregator function (e.g., the element-wise mean) can also help reduce the impact of ϵ by re-weighting the relative contributions of each image patch. But, this is not true of all aggregator functions. For example, the element-wise sum is also an aggregator function (i.e., it is permutation invariant) but any change in v_i would have a $1 : 1$ effect on c. However, an aggregator that lets all of the signal through is even worse than the standard GNN at approximating the unit step function. Since removing the weighting removes the supporting role of the aggregator to the asymptotic barrier function, it is reasonable to expect that the preservation

of this behaviour by the product aggregator is important, if secondary to the key approximation conditions.

Similar to the element-wise mean, the product is also amenable to my simplification that assumes $v_i = v_j$ for all $i, j \in \{1, \ldots, N^v\}$ (i.e., the basic behaviour is amenable to the current structure of my analysis). Thus, one can simplify the formulation of c with the product to only include μ and ϵ. One change from the element-wise mean is that μ is no longer the average value of the image patches, it is the $(N^v)^{th}$ root. However, I will continue to use the Greek letter μ to clarify its role and representation of the value when $v_i = v_j$ for all $i, j \in \{1, \ldots, N^v\}$. This results in the following equations.

$$\mu^{N^v} = \prod_{i=1}^{N^v} v_i \tag{2.5}$$

$$\mu = \left(\prod_{i=1}^{N^v} v_i \right)^{\frac{1}{N^v}} \tag{2.6}$$

$$c = (\mu + \epsilon)\mu^{(N^v - 1)} \tag{2.7}$$

Conveniently, the n^{th} root of a number between [0, 1] is also between [0, 1]. A proof of this fact is outside of the scope of this project, but some simple experimentation can give one an easy intuition. For example, 0.1 to the 100^{th} root, or $0.1^{\frac{1}{100}}$, is approximately equal to 0.9772372 and $0.99^{(1/5)}$ is approximately equal to 0.9998995. Thus, the n^{th} root of a number between [0, 1] drives the value closer to 1 while the n^{th} product of a value (x^n) between [0, 1] drives the value closer to 0. This means that the value of μ when $v_i = v_j$ for all $i, j \in \{1, \ldots, N^v\}$ always exists and is always a larger number than the starting values (i.e., it is an upper bound on those values which only occurs when all of the starting values are equal).

Figure 7 shows how the product better resembles the unit step function than the element-wise mean. One can see why by considering each of the three conditions:

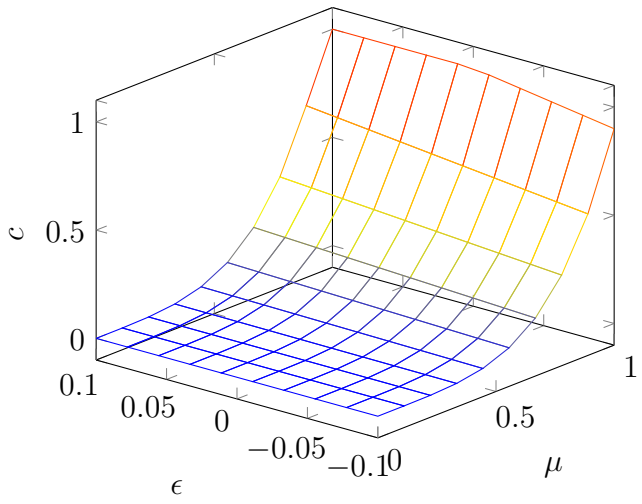

Figure 7: A graph of $c = (\mu + \epsilon)\mu^3$. Changes to μ via ϵ are now slightly larger when μ is larger. However, a larger range of values of μ are mapped to values of c close to 0. Overall, the graph better resembles the unit step function than the graph of the element-wise mean without $H(x, \delta)$.

zero gradient (ZG), small transition window (STW) and large transition gradient (LTG). For ZG, the visualization shows that a larger percentage of the values of μ are mapped close to 0. If one defines any input value that results in a c value less than a proximity threshold (t_h) of 0.01 (i.e. producing less than 1% of the maximum output of c) as "close to 0," then every μ value less than or equal to 0.31 (i.e., 31% of the input space) is close to 0. Although this threshold is somewhat arbitrary, the graph shows that for any fixed ϵ and any hypothetical $t_h > 0$, the product maps a larger set of values of μ to values of c less than t than the element-wise mean. Consequently, the product is always better than the element-wise mean on the ZG condition for a hypothetical $t_h > 0$.

One can perform a similar analysis for the STW condition. The key insight is that, for any fixed ϵ and hypothetical $t_h > 0$, the vertical line drawn where the function intersects with t_h will always have a larger value of μ that maps to the corresponding value of c. This value of μ will be called t_v. In effect, this means that the gradient of the product is increasing over time while the gradient of the element-wise mean is constant, which one knows by definition. This means that the product is always

better than the element-wise mean on the STW condition for hypothetical $t_h > 0$.

Finally, regarding the LTG condition, if one accepts that for every fixed ϵ and $t_h > 0$ the value of t_v is larger for the product function, then its gradient must also be steeper. One can deduce this as the end point of both functions is the same and yet, by virtue of t_v being greater for the product, the output of the product has farther to go to reach the final output value. Since one knows by definition that the gradient of the product is always increasing, the gradient of the product must be larger. This means that the product is always better than the element-wise mean on the LTG condition for hypothetical $t_h > 0$.

Although each of these conclusions are derived from very informal deductions, they are sufficient to illustrate the general improvement afforded by the product aggregator in a precise way.[6] On the basis of this analysis, I conclude that there are good reasons to believe that the product aggregator is a better approximation of the unit step function in the absence of my proposed asymptotic barrier function. By virtue of this approximation, one can also conclude that it imposes greater limitations on changes to a single image patch. I discuss the properties of this approach in the next section.

2.4 The Asymptotic Barrier

My asymptotic barrier function is designed to provide an even better approximation of the unit step function than the product aggregator. Of all of the design decisions discussed in this project, the final, multiclass asymptotic barrier function is the single-most important piece of the asymptotic GNN. It is the true source of substructure invariance in the model.[7] Thus, the asymptotic barrier function is similar to the convolutional kernel for convolutional neural networks (CNNs). Without it, there

[6] One might say that they have sufficient precision to be amenable to a more rigorous proof in future work.

[7] A fact that I empirically demonstrate in Chapter 4.

simply is no CNN. However, there are many other essential components that play a supportive role to the asymptotic barrier function, including the product aggregator discussed above and the learning branch discussed later in this chapter.

There are two quick caveats that I would like to make before I start my analysis. First, the asymptotic barrier function has some topical resemblance to the Gumbel-Softmax both in terms of its approximation of categorical output and its use of the negative of the natural logarithm. However, I want to emphasize that my preliminary research into applying the Gumbel-Softmax in place of my asymptotic barrier function resulted in models with very poor accuracy scores on normal images (e.g., 30%). Consequently, there is no trivial mapping between the two mechanisms. Second, in later research I tried replacing the natural logarithm in my proposed asymptotic non-linearity with the inverse of $\boldsymbol{v}$ yielding $(\boldsymbol{v} + \kappa)^{-\psi}$. My preliminary results were very similar to the results I report in Chapter 3. Future research will need to evaluate whether these two non-linearities are similar for the reasons discussed in sections that follow. However, this does provide preliminary evidence that it is the asymptotic behaviour of the non-linearity that is critical, not the specific implementation that I propose here. With these caveats in hand, I continue my analysis of my proposed asymptotic barrier function.

The key insight of my asymptotic barrier function is to use an asymptotic non-linearity to better approximate the unit step function. Since the input values to the asymptotic barrier function ($\boldsymbol{v}_i$ for $i = 1 : N^v$) range between 0 and 1, it needs to be defined over the domain [0, 1]. The natural logarithm is a function that is defined over the domain (0, 1] and has an asymptote at 0. That is, the limit approaches negative infinity as the input approaches 0 from the right. Critically, the unit step function operates over positive values. To better approximate this behaviour, I take the negative of the natural logarithm to flip the sign to positive infinity. In an effort to better satisfy the LTG, zero gradient (ZG), and small transition window (STW)

conditions, one can include a simple hyperparameter that exponentiates the resulting output as follows.

$$\Gamma(\boldsymbol{v}, \psi, \kappa) = (-1)^{\psi} \cdot \ln^{\psi}(\boldsymbol{v} + \kappa) \tag{2.8}$$

where ψ is a scaling hyperparameter of the asymptotic barrier function and κ is another hyperparameter that is explained below.

Figure 8 provides a visualization of four different plots of Γ with ψ set to integers 1 through 4 and when $\boldsymbol{v}$ is scalar (v). My formalization of Γ swaps back to its original characterization from Chapter 1 as a *mis*match detector (i.e., where larger values indicate a worse match with the class and correspond with a smaller output value). Although I will swap back to the match detector in a moment by using $1 - v$ as the input to Γ, I wanted to include the basic behaviour of the previous equation without adding too many steps at once.

Simple visual inspection of these figures should show that each of the approximation conditions improves as ψ increases. The lower half of each curve is flatter with increasing ψ leading to a larger range of input values falling below the horizontal dashed lines (t_h). This improves the ZG condition. The vertical line drawn where the function intersects with t_h is smaller—instead of being larger for the match detectors—which means that the transition window is smaller with increasing ψ.[8] This improves the STW condition. Finally, the gradient of the output values is larger when $v > t_v$ as ψ increases. Together, these properties suggest that Γ becomes a better approximation of $H(x, \delta)$ as ψ increases.

There are two key issues with this approach that are closely related. First, infinite values are problematic for gradient-based learning and, second, the unit step function has a bounded output. To resolve both of these problems, one can truncate the

[8]It might seem like I am cheating to compare different values of t_h for each value of ψ. However, if one were to normalize the output, as I do in Figure 9, t_h would be equal (specifically 0.5) for each graph. Consequently, it is simply the change in the upper bound as ψ increases that requires this change in the value of t_h.

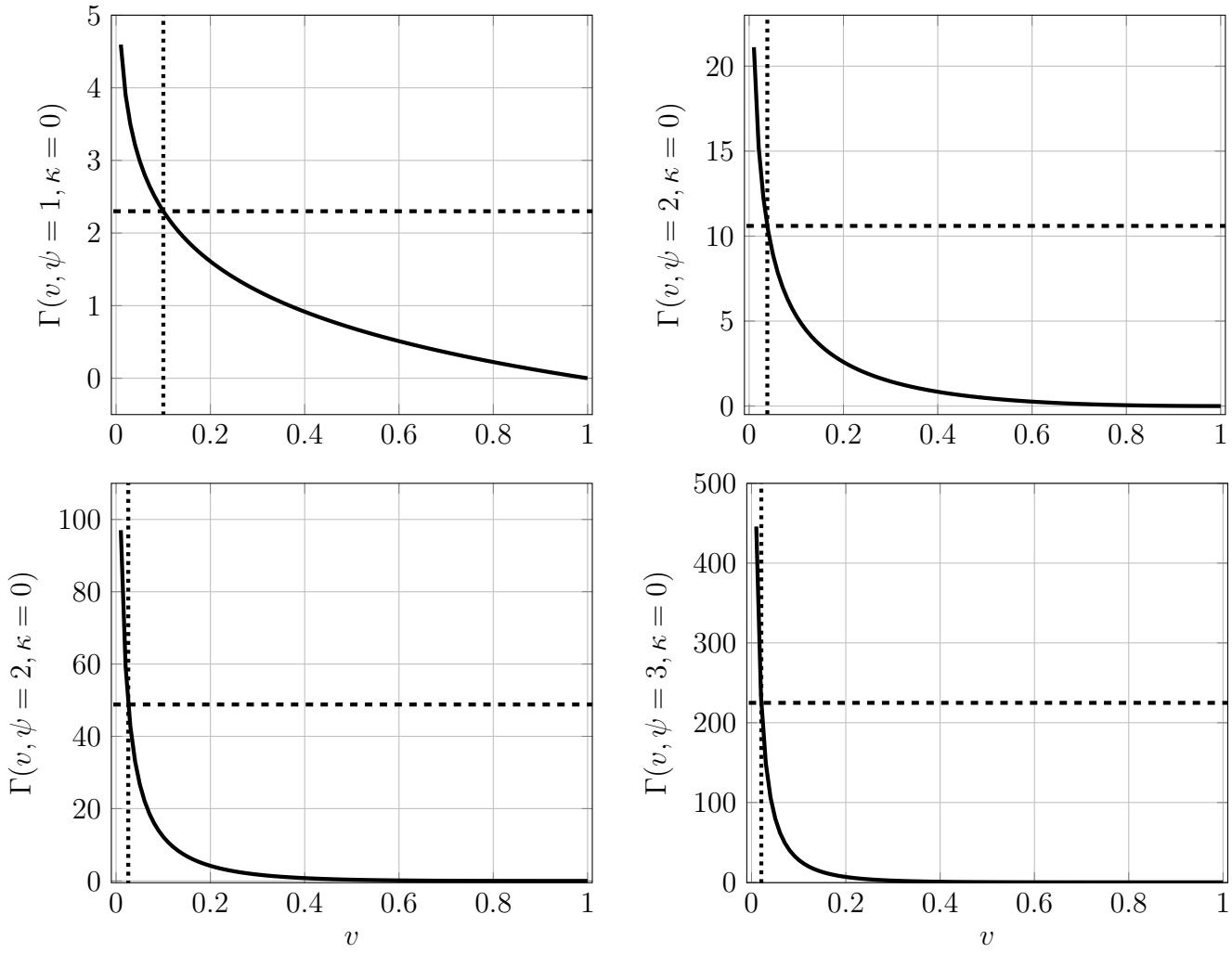

Figure 8: Four plots of the asymptotic barrier function with different values of ψ. The values are the integers from 1 to 4 from top-left to bottom-right. Each graph has one hundred equally spaced samples from within the domain [0, 1]. Consequently, the maximum value displayed for each graph has 0.01 as its input value for v. The horizontal dashed line is the central output value of Γ between the inputs $v = 1$ and $v = 0.01$. It models one possible approximation threshold (t_h) over the input domain. The vertical dotted line captures the intersection between t_h and the graph. It shows the value of μ (t_v) for the given t_h. Together, these dashed lines help one observe the effect of ψ on the approximation conditions: ZG, STW, and LTG. Critically, as ψ increases, t_v shifts to the left (i.e., gets smaller). This leads to an increasingly good approximation of $H(x, \delta)$.

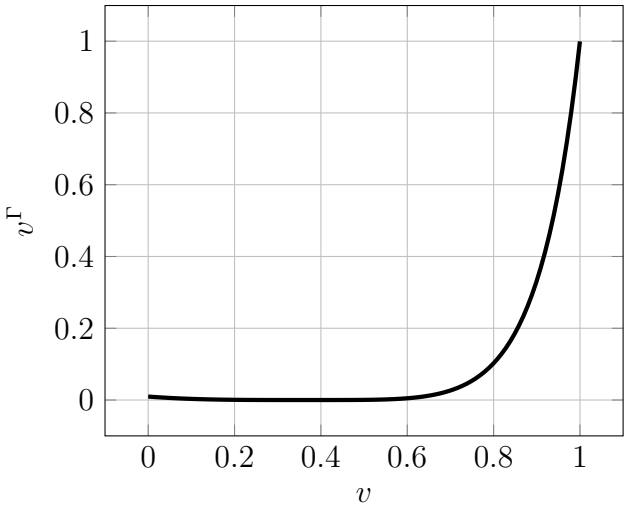

Figure 9: A graph of the asymptotic barrier function when $\psi = 4$ and $\kappa = \frac{1}{e}$. This sets the maximum value to be 1, which simplifies subsequent discussions. The graph has one hundred equally spaced samples from within the domain [0, 1].

output of Γ by adding a small constant (κ) element-wise to the input. As a bonus, κ acts to parameterize the desired maximum value of the output. Together, κ and ψ enable one to select increasingly good approximations of $H(x, \delta)$ while bounding the maximum value of the final output of the function. This makes it a very flexible non-linearity. For example, I consider the case when $\psi = 4$, $\kappa = \frac{1}{e}$ and the input is $1 - v$ (i.e., the strength of the match with the class instead of the mismatch) to simplify the rest of the analysis of the asymptotic barrier function's behaviour (see Figure 9).[9] This constrains the output of Γ to be in [0, 1].

Recall that the current objective is to use the asymptotic barrier function to better approximate the unit step function from Figure 5. As with the examples from the previous section—element-wise mean of $H(x, \delta)$, element-wise mean without $H(x, \delta)$, and product aggregator without $H(x, \delta)$—one can use the simplification (μ) where $\boldsymbol{v}$ is scalar and $v_i = v_j$ for $i, j \in \{1, \ldots, N^v\}$. However, since the change (ϵ) to the base

[9]The astute observer might notice that there is a small increase in the value of v^Γ as v approaches 0 and wonder how this might effect my results. Given that the results were the same when I used the inverse of $\boldsymbol{v}$ instead of the natural logarithm and $\boldsymbol{v}^{-1}$ is monotonically decreasing as $\boldsymbol{v}$ increases (except when it crosses zero), I can preliminarily conclude that this small increase is *not* driving my results.

values (v_i) occurs before the asymptotic barrier function, one needs to approximate μ by taking the inverse of $\Gamma(\mu, \psi, \kappa)$ (i.e., the value after μ after Γ). The equations for this inverse (Γ^{-1}), μ, and the class output c are defined below.

$$f(x, \psi) = ((-1)^{-\psi} \cdot x)^{\frac{1}{\psi}} \tag{2.9}$$

$$\Gamma^{-1}(x, \psi, \kappa) = e^{f(x,\psi)} - \kappa \tag{2.10}$$

$$\Gamma(\mu, \psi, \kappa)^{N^v} = \prod_{i=1}^{N^v} \Gamma(v_i, \psi, \kappa) \tag{2.11}$$

$$\mu = \Gamma^{-1}\left(\left(\prod_{i=1}^{N^v} \Gamma(v_i, \psi, \kappa)\right)^{\frac{1}{N^v}}, \psi, \kappa\right) \tag{2.12}$$

$$c = \Gamma(\mu + \epsilon, \psi, \kappa) \cdot \Gamma(\mu, \psi, \kappa)^{(N^v - 1)} \tag{2.13}$$

where x is equal to the output of Γ and f is equal to $\ln(\Gamma + \kappa)$. Although the formulation is more complex due to the required inverse of Γ, the final equation is just the product aggregator (Equation 2.9) from the previous section. The main change is that Γ is now modifying the signals of μ instead of v_i. If one represents the modified barrier node with the symbol v_ϵ^Γ and the generic, unmodified barrier node with symbol v_μ^Γ (to simplify the notation), one gets.

$$c = v_{\mu+\epsilon}^\Gamma \cdot (v_\mu^\Gamma)^{N^v - 1} \tag{2.14}$$

One can then visualize the output of this function for $\psi = 4$, $\kappa = \frac{1}{e}$ and an input of $1 - v$ in Figure 10.

Equation 2.14 gives a much better approximation of $H(x, \delta)$ than the normalized sum and product aggregators. To understand why, one can examine how the approximation conditions have changed with the inclusion of Γ. For the ZG condition, if the horizontal line is set to give c an upper bound of 0.01 ($t_h = 0.01$), then every v_i less than or equal to 0.89 (i.e., 89% of the input space) is "close to zero" relative to this

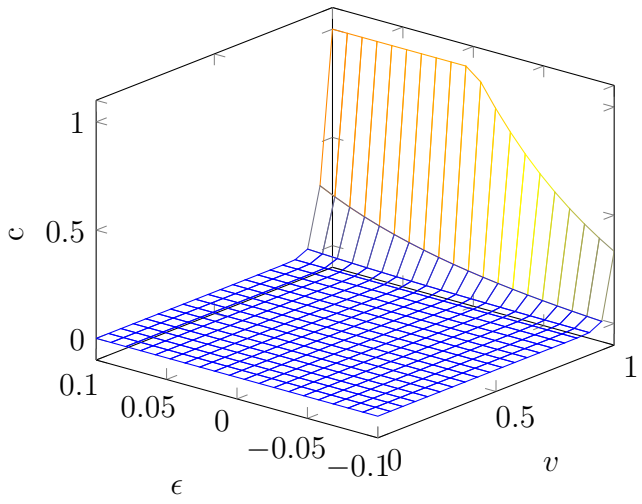

Figure 10: A graph of $c = v_{\mu+\epsilon}^{\Gamma} \cdot (v_{\mu}^{\Gamma})^{N^v-1}$ with $\psi = 4$, $\kappa = \frac{1}{e}$, and an input of $1 - v$ that is simplified to μ. The graph resembles the unit step function graph, but with a bounded asymptotic discontinuity near 1 instead of a jump discontinuity. This is the best approximation of $H(x, \delta)$ discussed thus far.

threshold. This is an approximate increase of 58% relative to the product aggregator on its own. For the STW condition, only 11% of the values fall within the transition time window for $t_h = 0.01$. The gradient is also much larger as the output needs to transition farther in a shorter amount of time. Consequently, the best approximation of the unit step function considered thus far is Γ. By extension, one can conclude that Γ is able to constrain the impact of ϵ on c by approximating $H(x, \delta)$.[10]

It is worth summarizing what I have accomplished thus far. Throughout this section I have been trying to approximate the unit step function ($H(x, \delta)$) with another function that has a tractable gradient. After making some simplifications to help with the analysis, I outlined three key conditions for approximating $H(x, \delta)$: the zero gradient (ZG), small transition window (STW) and large transition gradient (LTG) conditions. I then showed that removing the unit step function (i.e., using a traditional GNN) from the normalized sum aggregator almost entirely removes the behaviour of each of these conditions. This motivated my introduction of the product aggregator, which improved the ZG, STW, and LTG conditions relative to the

[10]For a sketch of a proof, see Appendix 1.

normalized sum. Finally, I introduced my asymptotic barrier function and showed how its hyperparameters can be used to improve my approximation of $H(x, \delta)$ along each of the three dimensions. This completes my analysis of Γ in the single class case. I will now extend this discussion to the multiclass case, which is essential for many classification tasks like MNIST.

2.5 The Multiclass Asymptotic Barrier Function

To formulate the multiclass case of the asymptotic barrier function with K classes, I need to incorporate three additional constraints. First, I constrain the base nodes to be a vector ($\boldsymbol{v}$) with exactly K elements. I will reference the individual elements of $\boldsymbol{v}$ by indexing them in the exponent position. For example, the m^{th} element of the vector $\boldsymbol{v}$ is the scalar value v^m. Second, as I outlined in the first section of this chapter, all of the elements of $\boldsymbol{v}_i$ must sum to 1 for each image patch $i = 1 : N^v$. They each define a probability distribution over the K classes for that image patch. Third, the output of c needs to be a one-hot vector. Before I describe the behaviour of the multiclass barrier function, it is worth considering the impact of these constraints on the model.

The most important result of these constraints is that, as a consequence of small values in the asymptotic barrier mapping to large values in the final output, the probability distribution describes the relative mismatch with a given class. For those who are familiar with the softmax function in its conventional usage, which I use to compute the probabilistic vector, this will seem very unintuitive if not completely incoherent. The softmax of the logits of a classifier is used to compute which class the input *belongs to* not which class it *does not* belong to. However, careful thought will reveal that these semantics are less than ideal when it comes to the substructure contained in an image patch of a larger image that one is classifying. For example,

the three pixel vertical line that can be found in a 3-by-3 image patch in the top-right position of a 5-by-5 pixel image could belong to every digit class *except* the '5' and '6' classes in their standard form (see Figure 11). Although one could say that there is zero probability of a match for classes 5 and 6 and an equal probability match for the other classes, this has some unusual implications. The foremost of which is that, the more classes that a substructure contributes to, the less it contributes to each of them. That is, if there were actually 9 different digits that used that image patch (or 5), the 3-pixel vertical line in the top-right corner of the image would somehow contribute less (or more for fewer digits) to that image.

Figure 11: A visualization of a 3-by-3 pixel image patch in the top-right corner a 5-by-5 pixel image. Green squares indicate agreement between the image patch and the corresponding digit while red squares indicate disagreement. The vertical line in the top-right position has perfect agreement with every digit except the '5' and '6' as shown by the red squares.

The statisticians in my audience will reply to this argument that *of course* the final probability will change if the match is with more (or fewer) output classes for the entire image. An image patch that is less common *should* contribute more to the classes it is used by. In response, I pose the following question: is this actually true in this domain? What is the most defining image patch of the digit '1'? What about the image patches of the digit '1' is "uncommon" in any statistical sense? The answer is self-evidently, "Nothing." The image patches of the digit '1' might well be the most common image patches for digits, at least for small images. Despite this fact, every single one of them is critically important to correctly identifying the digit. What about the digit '4'? One might say that the upper 'U' at the top of the four is distinctive. But, it is also contained in many other digits (i.e., it contributes to all circles; e.g., '9', '8', '6', '0'). This issue is even more critical once one considers my focus: adversarial examples.

Adversarial examples exploit sensitivity in the network at the boundary of class

divisions. In other words, the boundary of what constitutes a valid "match" with a given class. If a network is designed to compute the probability of a match between the image patch and each class, then an adversarial attack only needs to change the input enough to get the second highest match to be larger than the current maximum.[11] That is, the robustness of the network is strongly coupled to the difference between the largest and second largest match probabilities (for an undirected attack). If an aggregator is included in the network, then this changes to the aggregated difference between the largest and second largest probability across images patches (e.g., the element-wise average). By contrast, the mismatch probability has a very different semantics.

For the mismatch probability (my proposed mechanism), a given image patch can be a 'match' (i.e., near zero mismatch probability) with any number of classes. For a simple case of the example I outlined above, the three pixel vertical line might have a 0 probability mismatch with all of the classes except '5' and '6', which it has a 0.5 probability mismatch with. Thus, it is not the second *best* probability that matters in this case, it is the *worst case* probability. The adversarial attack needs to make the image patch resemble a '5' or a '6' by, for example, decreasing the intensity of the middle pixel. Obviously, this is a somewhat contrived case for this example, but the key point is that *all good matches will already be contributing to the output* of a given image patch.

In essence, my proposed technique is a somewhat complicated, engineering solution for counting zeros in the image patches. That is, each zero is a vote for a given class for a particular image patch. Each image patch is allowed to have a vote for any number of classes. The aggregator function then tallies the votes and selects the class with the most votes as the winner. By using a product aggregator, each vote contributes multiplicatively so that spurious matches quickly approach zero and

[11] Researchers familiar with the domain will know that this is a very common occurrence for many different adversarial attacks.

one class quickly dominates—the one that is consistent with the largest number of image patches. In this framing, the relative difference between the best matches in a single patch are mostly irrelevant as an additional zero in a different patch will overwhelmingly dominate the minor change (by design). Throughout the rest of this section, I describe the behaviour of the entire multiclass asymptotic barrier function in a lot more detail, in particular, where not every mismatch probability is either 0 or the maximum bad match. However, the above discussion hopefully helps orient the reader to some of the underlying motivation behind my design decisions, especially as they relate to these constraints and the use of the mismatch probability rather than the conventional one.

To implement the three constraints discussed above within the asymptotic barrier function, I change κ to be a smaller value (0.1) and, consequently, I allow c to have a larger maximum output value (approximately 624382). I then add a softmax non-linearity,[12] this produces a one-hot output vector of size K, where K is the number of classes.

To begin, I can define this new formulation of Γ with the following five equations that build off of the simplifications I defined in previous sections.

$$1 = \sum_{m=1}^{K} v_i^m \quad \text{for } i = 1 : N^v \tag{2.15}$$

$$c^m = \prod_{i=1}^{N^v} \left((-1)^\psi \cdot \ln^\psi (v_i^m + \kappa) \right) \quad \text{for } m = 1 : K \tag{2.16}$$

$$\text{softmax}_m(\boldsymbol{c}) = \frac{e^{c^m}}{\sum_{k=1}^{K} e^{c^k}} \quad \text{for } m = 1 : K \tag{2.17}$$

where the first equality is the constraint on the input values, c^m is the output class for class m of K classes, and the last equation computes the softmax output for each element in $\boldsymbol{c}$. One will note that I will no longer consider ϵ for the multiclass case as

[12]The softmax is a non-linearity that handles multi-class classification (i.e., a vector of output values) and makes the output sum to 1.

one should already be confident that Γ constrains its impact based on the analyses of the previous sections. To understand the behaviour of these equations, consider the following graph when $K = 2$, $N^v = 4$, $\psi = 4$ and $\kappa = 0.1$ (see Figure 12).

At first glance, the behaviour of this function does not resemble the unit step function I introduced earlier in the chapter. However, the mapping does exist. The missing detail is that the unit step function is actually not defined at $1 - \delta$ (or 0 in its standard formulation). Consequently, what value one chooses to pick at that indeterminate point is merely a matter of convention or specific use case. If one assigns the undefined value to be 0.5, then the unit step function becomes a piecewise-constant function with three constants: it can be in a 0, 0.5, or 1 state. Now, if one considers the case when there are two values that need to be approximated, then it is natural to see their interaction in terms of these three values. When one of the values are off, the softmax guarantees that the other value is on. When one of them is in the middle state (0.5), the other *must also* be in the 0.5 state. The two classes then flip as the first class turns on. This is exactly the behaviour of the function in Figure 12.

My proposed asymptotic barrier function loosely behaves like a piecewise-constant unit step function when $K = 2$. It has three key states (0, 0.5, 1). Each of these states still *locally* preserves the approximation conditions for $H(x, \delta)$. That is, they have zero gradient (ZG) for roughly a third of their input. They have small transition windows (STW) at every transition point and they have large transition gradients (LTG) at every transition point. The major change is that there is now a third value incorporated into the transition states (i.e., the initial unit step function exhibits slightly more complex behaviour). What is especially interesting is how this behaviour changes as K increases.

When $K = 3$, one can start to get a sense of the general behaviour of the function that counts zeros, which I described at the beginning of the section (see Figure 13).

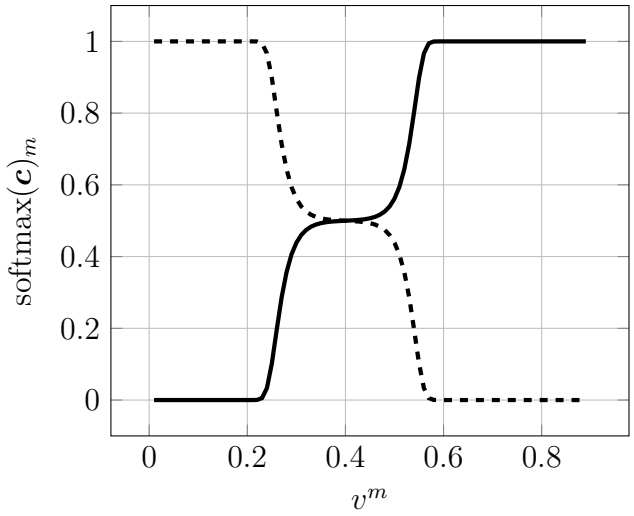

Figure 12: A graph of the softmax of the product of the asymptotic barrier function when $K = 2$, $N^v = 4$, $\psi = 4$ and $\kappa = 0.1$. The solid line is for $v^2 = 1 - v^1$ and the dashed line is for v^1. The graph has eighty-nine equally spaced samples from within the domain [0.01, 0.89]. In effect, the resulting output approximates a piecewise-constant step function with three pieces (e.g., when the value at the discontinuity does not equal 0 or 1) with an indeterminate class output when $x = 0.5 - \kappa$.

However, the behaviour is a lot more nuanced now. The key insight is that the third class acts as a bound on the maximum value that can be given to any other class. This is a direct consequence of the use of the final softmax, which forces the output values to sum to one, and the fact that small values of v^m are mapped through Γ to large values of c^m. For example, if the third class is 0.4, then there is only another $1 - 0.4 = 0.6$ left from the total output that can be allocated to the other two classes. However, for sufficiently small values (e.g., 0.01) any number of classes can be mapped to that value. The only requirement is that there is one class left over to absorb whatever remaining output is left to make the final sum equal 1. Understanding this behaviour is essential for understanding the final version of my proposed asymptotic barrier function and thus asymptotic GNNs so I will spend a little time unpacking it.

In a sense, the semantics of the last softmax states that there always must be some classes which are a bad match for the current input. Either one class can be very, very bad (i.e., $v^m = 1$) and any number of other classes can be perfect (i.e.,

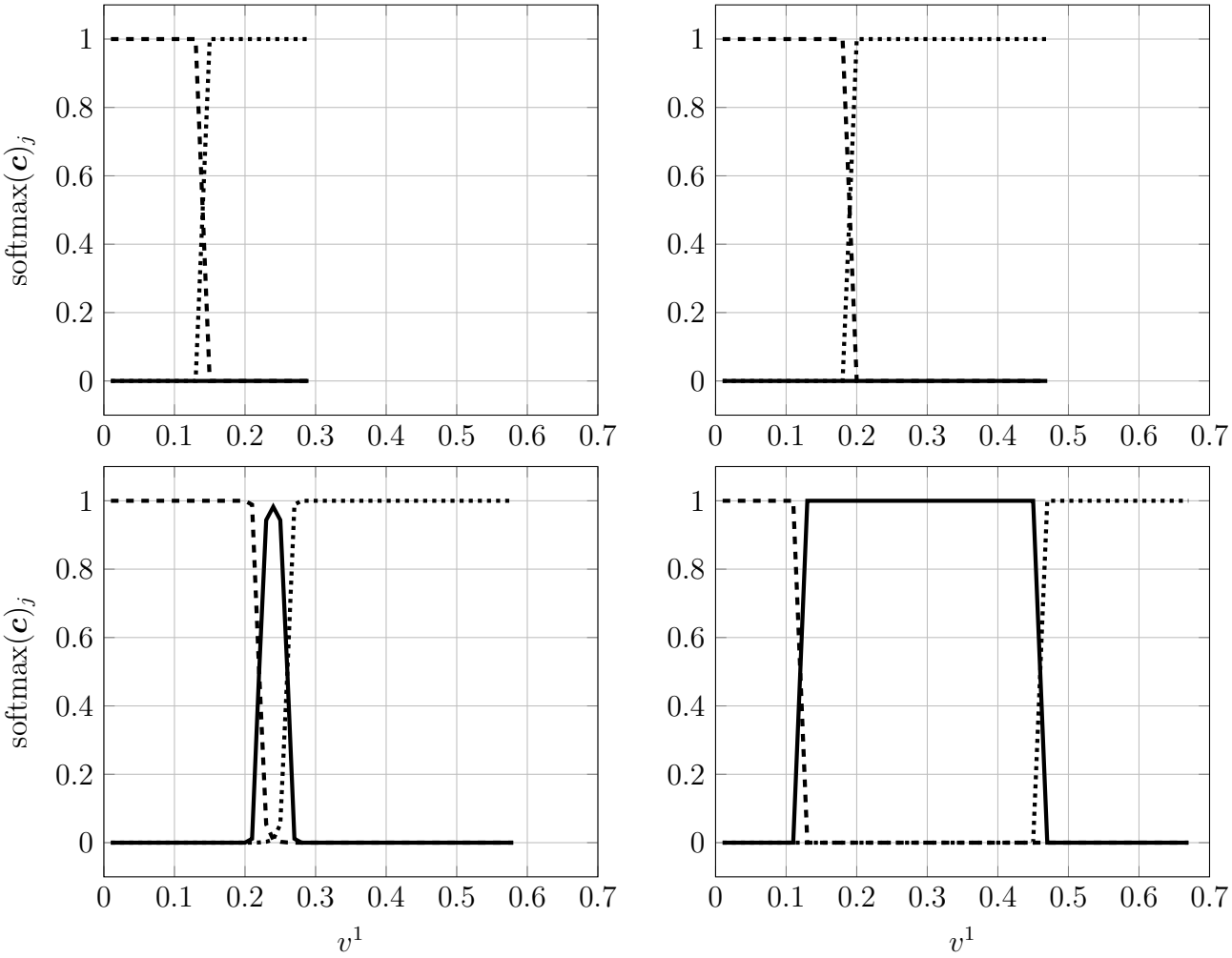

Figure 13: A graph of the softmax of the product of the asymptotic barrier function (Equation 2.21) when $K = 3$, $N^v = 4$, $\psi = 4$ and $\kappa = 0.1$. The solid line is for the third class v^3, the dashed line is for v^1, and the dotted line is for $v^2 = 1 - v^1 - v^3$. The values of v^3 from top left to bottom right are 0.42, 0.32, 0.22, 0.12 plus κ. All of the graphs have a sampling step size of 0.01. Each graph truncates when $v^1 + v^2 + v^3 > 1$. The key insights are that v^3 constrains the maximum values of v^1 and v^2 and the system is least like the unit step function (i.e., most volatile) when $v^3 = \frac{1}{K} - \kappa$.

$v^k = 0$ for all $k \neq m \in \{1, \ldots, N^v\}$). Or, M (where $M < K$) classes can be equally bad such that $v^m = \frac{1}{M}$ for $m = 1 : M$ with the remaining $(K - M)$ classes at zero. Although there are an infinite number of other combinations, the general pattern is the same: the softmax is designed to apply constraints on the classes that do not match the input well. This is even more important as most of the input space maps to small values (e.g., when $\kappa = 0.1$, $v^m = 0.2$ maps to $c = 62$ while $v^m = 0.05$ maps to a c that is greater than 60,000). Even relatively small values like 0.2 are still a bad match in terms of their output values.

Figure 13 helps visualize these properties of Γ, especially in terms of how they affect the transition between states in the system. For example, when the value of the third class is large (top row; large value means low match), then the first two classes must be small as I described above (high match). In this state, the function strongly resembles $H(x, \delta)$ in terms of each of the approximation conditions. That is, the entire space up to the transition window is mapped *to* 0 (ZG). The transition occurs over a very small time window (STW): less than 0.04 for $v^3 = 0.42 + \kappa$ and less than 0.09 for $v^3 = 0.32 + \kappa$. And, the transition changes in effectively two quick steps (i.e., there is a lead up of very small changes, then the values jump to 0.5 and then 1). As the value of the third class approaches an equal split in the classes (e.g., $v^3 = \frac{1}{3} - \kappa$ for $K = 3$; bottom row), the third class is introduced and the transitions between the classes smooth out. At this point, the function is least like $H(x, \delta)$, though it is still very similar—the true transition window (i.e., one that does not require a proximity threshold t_h) is less than 0.17 while the thresholded single class value for Γ was only slightly smaller (0.11). The result is that a change occurs in the system at the point of greatest volatility. At this point, the behaviour switches from a piecewise-constant function with two pieces to one that has three pieces. Finally, as the value of the third class continues to get smaller (high match), the volatility decreases and the transitions between each of the values become very steep. The

result is that the system reverts back to closely approximating $H(x, \delta)$.

There are three important properties about this approach. First, increasing the value of K does not change the overall behaviour of the system. For example, if $K = 4$, then the fourth value just acts as a bounded form of the behaviour when $K = 3$. That is, the fourth value just reduces the maximum values that can be allocated to the other classes. It pushes the graphs from Figure 13 so that all of the values are closer to zero. Although it is beyond the scope of the current project, it seems reasonable that this pattern could be extended by induction to any K.[13] Second, κ now has a direct impact on the behaviour of the system beyond setting the maximum value. By setting κ to $\frac{1}{K}$, one can use κ as a bias that places the system within the transition point where it is *least* like the unit step function (i.e., the point of greatest volatility). This could enable the system to more easily learn the division of unit step like behaviour that is relevant in the current domain.[14] At very least, it assumes the least about the final configuration that best approximates a given domain. Consequently, a κ of 0.1 or $\frac{1}{10}$ is ideal for a system with 10 classes like MNIST as it exactly matches this pattern. Thus, my choice for κ was never arbitrary. It just took a while to have enough context to understand why I picked it. Third, increasing N^v only increases the value of ψ by a multiple. This merely increases the steepness of the asymptotic barrier function, which better approximates the unit step function. The only caveat to this point is that if N^v is too large (relative to a given ψ), the dynamics can be so similar to the unit step function that the system becomes unstable (i.e., it functions like a true discontinuity). I found $N^v = 9$ (i.e., a 3-by-3 grid of image patches) to be a good value that works in practice.

It is worth briefly summarizing this multiclass extension of Γ. I initially stated that there were three key constraints for handling the multiclass case: the base node vectors

[13] I leave the proof of this fact for future work.

[14] My preliminary analyses of κ suggested that *most* values of κ do not work for the model. More importantly, the values that do work are either near $\frac{1}{K}$ or near $\frac{0.9}{e}$. I leave a detailed analysis of the properties of κ to future work.

($\boldsymbol{v}$) must have the same number of elements as the number of classes (K), the elements of $\boldsymbol{v}$ should sum to 1, and the output should be approximately one-hot. I explained that the standard match-based semantics of the probability vectors used by classifiers had unintentional and undesirable consequences for the system. I then explained how my mismatch-based semantics better aligned with the current problem and domain, especially in the context of adversarial examples. I then proposed to implement these constraints by setting κ to a smaller value (0.1) and introducing a softmax non-linearity on the output of Γ. By using a series of figures of the behaviour of the resulting function as the number of classes increases, I revealed that the multiclass case *better* approximates a piecewise-constant function with three pieces in terms of the three approximation conditions: ZG, STW, and LTG. In effect, the multiclass setup locally functions like a collection of individual unit step functions with special volatility points when the number of good candidates increases. My selection of κ was then revealed to be deeply informed by this insight. When $\kappa = \frac{1}{K}$ the system starts its training within the point of greatest volatility for the model, which assumes less about the underlying data domain. This completes my discussion of my asymptotic barrier function, which is the single-most important component of the asymptotic GNN. In the next section, I discuss the details of my learning branch (Ω), in brief.

2.6 Learning Branch

The goal of the current section is to motivate and outline the functionality of what I call the learning branch (Ω). To understand the underlying motivation, one needs to consider the broader implications of using a function like the asymptotic barrier as it applies to classification. However, before I go into these details, it is worth emphasizing that the learning branch is the second-most important design decision of my asymptotic GNNs. Together, the asymptotic barrier function, the learning branch,

and the hierarchical organization of the system outlined in Figure 3 are the essential structures of my asymptotic GNN. Almost everything else is an implementation detail. For this reason, it is worth it to consider how the asymptotic barrier functions more generally.

In Chapter 1, I outlined the notion of adversarial robustness defined by Papernot et al. (2016) as the ability to 1) have good accuracy and 2) classify inputs consistently within a neighbourhood of a real sample from the domain. I then explained how the authors define a neighbourhood as all the images within a specified distance of the original sample image, given some norm (e.g., L_2). However, they leave open how to construct this consistency.

The asymptotic barrier supports this consistency in the output by making one of the output dimensions asymptotically dominate. By constraining the base node vector to have the same number of dimensions as the final output vector, the structure of the network combined with the optimization algorithm (as opposed to the asymptotic barrier function) encourages the class dimension to be the one that dominates. In effect, this increases the difference between samples from different classes (in so far as they are correctly classified). And, in turn, this makes it *less* likely that samples from different classes share a neighbourhood. One can think of this as a barrier or boundary that separates samples from different classes with a very large change in the gradient of the output at each dimension of the respective classes. This is exactly what I showed in Figure 13. However, this effect is limited to changes in the dominant class. The asymptotic barrier function *does not constrain* the non-dominant classes unless they are about to transition to the dominant class. To constrain the non-dominant class values, I need another mechanism: the learning branch.

The purpose of the learning branch is to constrain the non-dominant classes *within* a given class (i.e., away from the asymptotic boundary). To consider this problem, I can no longer rely on the gradient of a scalar output for a single class. I must

consider the gradient with respect to all of the output classes (i.e., an output vector), which produces a matrix of first-order partial derivatives with respect to each of the input dimensions (the columns) and output dimensions (the rows). This matrix is commonly called the Jacobian matrix. In its simplest form, the goal is to make the gradient of each non-dominant class output in the Jacobian matrix equal to zero when comparing samples from the same class. For class $k \in [1, K]$, one can express this as follows:

$$\frac{\delta c^j}{\delta (v^\Gamma)^i} = 0 \quad \text{for } i = 1 : N^v,\ j = 1 : K,\ \text{and } j \neq k \tag{2.18}$$

where c^j is the output class for class j of K classes and $(v^\Gamma)^i$ is the input for the barrier node vector ($\boldsymbol{v}^\Gamma$) at element i. Critically, this does not necessarily mean that the output of the other class dimensions needs to be zero (i.e., that there is no general organization to the classes; e.g., that '4's are more similar to '9's than they are to '8's).

One can think of the action of the learning branch as a generalization of the zero gradient (ZG) condition that I discussed previously with respect to the asymptotic barrier function. The focus of the ZG condition was on the dominant class, which was constrained to be 'flat' for large regions of the input. Similarly, the learning branch constrains all of the non-dominant classes to be 'flat' in a higher-dimensional space described by the Jacobian matrix. Together, the asymptotic barrier function and the learning branch constrain the entire Jacobian matrix to approach zero everywhere except along the edge of the class boundaries. I describe how the learning branch approaches this state below.

The learning branch function (Ω) occurs prior to the final softmax in Equation 2.17. It is defined as follows:

$$\Omega(\overline{\boldsymbol{v}}, \tau, \zeta) = \underbrace{(1-\tau)\overline{\boldsymbol{v}}}_{\text{left branch}} + \underbrace{\tau \ln(\text{softmax}(\overline{\boldsymbol{v}}) + \zeta)}_{\text{right branch}} \tag{2.19}$$

where $\overline{\boldsymbol{v}}$ is the aggregated output of the asymptotic barrier function, ζ is a hyper-parameter,[15] and $\tau \in [0, 1]$ is a trainable parameter. I explain the action of ζ and τ in the learning branch throughout the rest of this section. However, the basic idea behind the learning branch is to use the right term in the sum $(\ln(\text{softmax}(\overline{\boldsymbol{v}}) + \zeta))$ to make the non-dominant class values approach zero. I will call this term with τ the *right branch* and the left term $((1-\tau)\overline{\boldsymbol{v}})$ the *left branch*. I will focus on the behaviour of the right branch first before discussing the weighted average resulting from the parameter τ.

As I discussed in the previous section, the output of the asymptotic barrier function $(\text{softmax}(\overline{\boldsymbol{v}}))$ is approximately one-hot with a one in the position of the dominant class and zeros for every other value. After adding a small constant (ζ), the natural logarithm of this output is a vector with ζ in the dominant class and $\ln(\zeta)$ for every other value. If ζ is a small value between zero and one, then $\ln(\zeta)$ is a negative number. Consequently, as a whole, the right branch in Equation 2.19 subtracts a constant from all of the non-dominant values of the left branch. This is the first part of how the learning branch makes the non-dominant class values approach zero (technically, $\ln(\zeta)$, which is negative). The second part leverages the weighted average resulting from τ.

To understand the action of τ in the learning branch, one first needs to understand that the *softmax* of the right branch in Equation 2.19 is approximately equal to the softmax of the left branch (ignoring the effect of τ). One can express this as follows.

$$\text{softmax}(\boldsymbol{x}) \simeq \text{softmax}(\ln(\text{softmax}(\boldsymbol{x}) + \zeta)) \quad \text{as } \zeta \to 0 \tag{2.20}$$

Consequently, the final class prediction of the two branches is approximately equal. What changes is how the structure of the final prediction is expressed and the effect of that expression on the non-dominant class rows in the Jacobian matrix.

[15] I found $\zeta = 10^{-10}$ to work well in practice.

The asymptotic GNN always starts with τ set to 0 and the value of τ increases when the conditions are right. In my research, the asymptotic GNN always eventually increases the value of τ and always does so monotonically (i.e., the value of τ never decreases over the course of training). However, preliminary work integrating the asymptotic barrier and learning branch into a CNN architecture did not produce the same behaviour. In particular, the value of τ never increased beyond zero for these models. I leave a more detailed exploration of this behaviour to future research.

With the asymptotic GNN by contrast, there are four different states that are expressed by the learning branch as the value of τ changes: when $\tau = 0$, when $0 < \tau < 1$ and every value in the left branch is greater than $|\ln(\zeta)|$, when $0 < \tau < 1$ and at least one value in the left branch is less than $|\ln(\zeta)|$, and when $\tau = 1$. When $\tau = 0$, the learning branch lets the output of the aggregated asymptotic barrier function pass through the left branch without the right branch affecting it. As the value of τ increases, the state changes and two things happen in parallel: a larger percentage of the right branch is subtracted from the left branch and a smaller percentage of the left branch contributes to the final output. Together, these two effects allow the network to proportionally increase (via the weighted average) how close to zero the non-dominant rows in the Jacobian matrix should be. This behaviour can be explained as follows. Over the course of training, the model will accurately classify more of the training set. As more of the training samples are assigned the correct dominant class, the relative impact on the error signal that is caused by decreasing the non-dominant classes on correctly classified training samples will be more beneficial than the impact of decreasing the correct class for the incorrectly classified training samples. When this happens, the model will increase τ. This completes the first two states of the learning branch. I continue the discussion of the other two states below.

At some point during training, the right branch will start to dominate (i.e., at least one of the values in the left branch will be less than $|\ln(\zeta)|$). When this happens—the

precise time of which is largely dependent on the value of ζ and the maximum value of the elements of the output of the aggregated asymptotic barrier function ($\overline{\boldsymbol{v}}$)—one can think of the action of the left branch as modifying the right branch. There are two reasons why it is helpful to change one's perspective in this way. First, the value that is less than $|\ln(\zeta)|$ will now be negative, so it is awkward to say that the right branch is subtracting from some quantity in the left branch.[16] Second, the relative contribution of the left branch to the final output is a lot less significant. In sum, in the third state of the learning branch, the final output more closely resembles the right branch ($\ln(\text{softmax}(\overline{\boldsymbol{v}}) + \zeta)$) than the left. As I stated above, the final state of the right branch is a vector with a zero in the dominant class value and $\ln(\zeta)$ in all the non-dominant class values. Consequently, it can be useful to think of the action of the left branch in the third state as a modification *away* from the output of the right branch. In the final state, when $\tau = 1$, the learning branch lets the right branch pass through without the left branch effecting it. In this state, the Jacobian matrix is approximately zero everywhere for all samples that are correctly classified.

Together, the asymptotic barrier function and the learning branch create the required classification consistency of inputs within a neighbourhood of each sample by doing the following. The asymptotic barrier prevents inputs from different classes from sharing the same neighbourhood while the learning branch makes all members of the same class share a *single* neighbourhood. It is my hypothesis that the result of these two functions produces adversarial robustness in machine vision.

Although the final state of the learning branch might seem extreme as it effectively 'freezes' the internal state of the network, the basic idea of the learning branch strongly resembles a common phenomenon in the neural development of humans and animals. This phenomenon is called a "critical period" of learning (Berardi, Pizzorusso, & Maffei, 2000). Many organisms experience periods during their devel-

[16] As a reminder, the values of the left branch are always positive as the product of the output of the asymptotic barrier function is always positive.

opment when a particular behaviour, cognitive or otherwise, is typically learned and after which it usually cannot be learned. For example, the output of neurons responsible for sound localization in the barn owl can be modified through intervention up to around two hundred days (Brainard & Knudsen, 1998). After this critical period, the neural patterns crystallize into a stable pattern of behaviour for the rest of their life—a fundamental constraint of their neurobiology. In many ways, the action of τ in the learning branch produces the same effect. When τ is small, the network is able to learn new organizations of the classes of the training samples. As τ approaches and ultimately achieves the value of '1', the network stabilizes and 'freezes' into a terminal state where gradient-based learning is no longer possible and, more importantly, where the network is a lot less susceptible to gradient-based adversarial examples. In both cases, this is achieved by making the values of the Jacobian matrix approach zero.

The goal of this section was to motivate and outline the functionality of the learning branch (Ω). To motivate Ω, I returned to the definition of adversarial robustness by Papernot et al. (2016) as a consistent classification of the inputs within a neighbourhood of real samples from the domain. I then described how the asymptotic barrier function contributes to this objective by making the assigned class output dimension asymptotically dominate and thereby decreasing the chances that samples from different classes share a neighbourhood. This led to my discussion of the learning branch as a means to constrain the non-dominant class values for samples that share their dominant class. I defined this constraint as an action that makes the non-dominant class rows of the Jacobian matrix approach zero. This action resembles the ZG condition produced by the asymptotic barrier function over large regions of the gradient of the dominant class, but in a higher dimensional space describing interactions between the non-dominant class values.

From there, I defined the equation for Ω and described the behaviour of the

equation in terms of two branches—a left branch that provides the raw signal of the aggregated output of the asymptotic barrier function and a right branch that leverages a hyperparameter (ζ) to make the non-dominant class values approach zero. These branches are modulated by a trainable parameter (τ) of the learning branch that decreases the impact of the left branch while increasing the impact of the right branch over the course of training. As τ increases, four separate states are produced by the learning branch that take the network from a standard Jacobian matrix to one that approaches zero everywhere. Although the action of the learning branch is extreme, I argued that it is consistent with a neurobiological constraint that is common in animals and humans, the critical learning period, which also freezes the state of the system such that learning is no longer possible. This completes my description of the core mechanisms of the asymptotic GNN. In the next section, I cover the details of the substructure processing (ϕ).

2.7 Substructure Processing

In Section 2.1, I stated that the minimal requirement for an asymptotic GNN was a function (ϕ) modeled by a fully-connected neural network. This function accepts the pixels of an image patch ($\boldsymbol{s}_i$ for $i = 1 : N^s$) as input and outputs a vector of probabilities of its associated image patch node ($\boldsymbol{v}_i$ for $i = 1 : N^v$). These probabilities represent the relative mismatch of the substructure of the node to the set of K available classes as I described previously. Although this approach meets the minimal requirements, it is not easy to train for image classification. Additionally, by virtue of being the minimal requirement, it makes few theoretical commitments that are interesting to evaluate. Consequently, I propose to use a slightly more intricate model of ϕ that lends itself to training and more interesting theoretical commitments. This model builds off of many standard GNN setups (for example, see Gori et al., 2005;

Duvenaud et al., 2015; Zaheer et al., 2017).

One important caveat is that the techniques described in this section are not a contribution in and of themselves. Every element of the substructure processing builds off of existing techniques from the machine learning literature. As a consequence, I will only explain the basic details in brief with sufficient references that the interested reader can easily look up the required information. My justification for this approach is that each of the techniques is only relevant as an expression of the underlying inductive biases that they implicitly implement. Note that I do not say *relational* inductive biases as some of the modifications that I describe do not restrict the direct relationships between the inputs to the neural network layer. However, if another technique produces the same inductive bias, then the technique I discuss should be entirely replaceable without affecting my conclusions. That is, the importance of the substructure processing for the current work is that it enables an evaluation of the relative contribution of different architectural constraints to the structure of asymptotic GNNs.

At its core, the basic setup of the substructure processing is designed to provide enough structure for me to be able to evaluate the impact of the asymptotic barrier function and the learning branch (i.e., the asymptotic layer). Simultaneously, the substructure is also designed to be simple so that there are a minimal number of architectural constraints directly influencing the results. Consequently, I do not use any of the standard tricks of the trade in deep learning (e.g., drop-out, batch normalization, residual connections, attention, large multilayer networks, ensembling). This motivates my basic setup of ϕ.

2.7.1 Basic setup

The basic setup is to define ϕ as the composition of three simpler functions: ϕ_α, ϕ_β and average pooling. The first step (ϕ_α) transforms the input image patch ($\boldsymbol{s}_i$ for

$i = 1 : N^s$) into a vector of latent variables ($\boldsymbol{z}_i$ for $i = 1 : N^s$). The second step (ϕ_β) accepts $\boldsymbol{z}_i$ as input and outputs a probabilistic vector ($\widetilde{\boldsymbol{v}}_i$ for $i = 1 : N^s$), resulting from a softmax non-linearity. The third step applies average pooling, which reduces the size of N^s to N^v for the asymptotic barrier function (Γ).

Although this formulation is not exactly the same as my original description of the model where each image patch is paired with a single image patch node (i.e., $N^s = N^v$), one can think of the current description as a generalization of that simpler case when $N^s \neq N^v$. For a 28-by-28 pixel image with my chosen window (5-by-5) and step sizes (3-by-3), each image patch node ($\boldsymbol{v}$) receives input from 3 of the 9 image patch probabilistic vectors (for edge nodes) or 4 of the 9 image patch probabilistic vectors (for the center nodes).

Both ϕ_α and ϕ_β are implemented as fully-connected neural networks with a single hidden layer. This gives the following three equations.

$$\phi_\alpha(\boldsymbol{s}_i, \theta_\alpha) = \boldsymbol{z}_i = \tanh(\text{relu}(\boldsymbol{s}_i \boldsymbol{W}_\alpha + \boldsymbol{b}_\alpha)\boldsymbol{W}'_\alpha + \boldsymbol{b}'_\alpha) \tag{2.21}$$

$$\phi_\beta(\boldsymbol{z}_i, \theta_\beta) = \widetilde{\boldsymbol{v}}_i = \text{softmax}(\text{relu}(\boldsymbol{z}_i \boldsymbol{W}_\beta + \boldsymbol{b}_\beta)\boldsymbol{W}'_\beta + \boldsymbol{b}'_\beta) \tag{2.22}$$

$$\phi(\boldsymbol{s}_0, \cdots, \boldsymbol{s}_{N^s}, \theta_\alpha, \theta_\beta) = \frac{1}{N^s} \sum_{i=1}^{N^s} \phi_\beta(\phi_\alpha(\boldsymbol{s}_i, \theta_\alpha), \theta_\beta) \tag{2.23}$$

where $\theta_\alpha = \{\boldsymbol{W}_\alpha, \boldsymbol{W}'_\alpha, \boldsymbol{b}_\alpha, \boldsymbol{b}'_\alpha\}$ and $\theta_\beta = \{\boldsymbol{W}_\beta, \boldsymbol{W}'_\beta, \boldsymbol{b}_\beta, \boldsymbol{b}'_\beta\}$.

The training of ϕ is also broken into two steps. The first step trains ϕ_α as the encoder network of an autoencoder by training a decoder network (ϕ_γ), which is an approximate inverse of ϕ_α as follows.[17]

$$\phi_\gamma(\boldsymbol{z}, \theta_\gamma) = \widetilde{\boldsymbol{s}} = \tanh(\boldsymbol{W}'_\gamma \text{relu}(\boldsymbol{W}_\gamma \boldsymbol{z} + \boldsymbol{b}_\gamma) + \boldsymbol{b}'_\gamma) \tag{2.24}$$

where $\theta_\gamma = \{\boldsymbol{W}_\gamma, \boldsymbol{W}'_\gamma, \boldsymbol{b}_\gamma, \boldsymbol{b}'_\gamma\}$. I then use the standard, unregularized reconstruction

[17]It is an approximate inverse as it is not guaranteed to preserve scale (see Goodfellow et al., 2016).

loss for autoencoder training over the image patches (for details, see Bourlard & Kamp, 1988; Hinton & Zemel, 1994). In place of stochastic gradient descent, I use the Adam optimizer (Kingma & Ba, 2014).

The advantage of pre-training the encoder network is that it simplifies comparisons between variations of the asymptotic GNN. In effect, a single encoder can be trained once and shared across all of the model variations. This has a bunch of computational advantages (e.g., speeds up training), but it also reduces undesirable variation between the models (e.g., stochasticity in weight initialization, stochasticity in data shuffling). This increases the power of my experiments and analyses.

After training ϕ_α (and ϕ_γ, although it is not used in subsequent processing) to completion, I fix θ_α and perform supervised training over ϕ_β on the full images and through the full asymptotic GNN. I use the multiclass cross-entropy loss over the values of $\boldsymbol{c}$ with Adam. This completes the basic setup and my description of the asymptotic GNN model.

2.8 Summary

In this chapter, I provided a detailed description of the general formulation and organization of my asymptotic GNN. This included a description of the four basic steps of the model: the substructure network (ϕ), the asymptotic barrier function (Γ), the product aggregator ($\prod$), and the learning branch (Ω). This organization is one of the essential design decisions of the asymptotic GNN.

My description of the organization of the model led to a discussion of the framework for my analysis of the asymptotic barrier function's behaviour. I discussed how the asymptotic barrier approximated the unit step function in terms of three key conditions: the zero gradient (ZG), small transition window (STW), and large transition gradient (LTG) conditions. After examining the basic behaviour of the unit

step function as a perfect match detector, I introduced the basic GNN layer as a counterpoint to the desirable properties of the unit step function.

My first improvement on the basic GNN layer was the product aggregator that I proposed at the beginning of this chapter. I showed how the product aggregator had a number of nice properties (e.g., trivial re-weighting) and achieved a better approximation of the unit step function than the basic GNN in terms of the three key conditions.

The next section discussed my formulation of the asymptotic barrier function in detail. I started this discussion with the single class case and showed that my formulation of the asymptotic barrier had a convenient parameterization that enabled better and better approximations of the unit step function. My analysis completed with a demonstration of the improvements of Γ over the product aggregator alone on each of the three conditions: ZG, STW, LTG.

After my initial introduction of the single class version of my asymptotic barrier function, I showed how the multiclass version better approximated the desirable behaviour of the unit step function. As the number of classes increased, this trend became even more stable. I explored some of the interesting dynamics of Γ then highlighted the importance of my choice of κ in the multiclass version of the asymptotic barrier. This completed my discussion of the single most essential structure for the asymptotic GNN.

The next section discussed the learning branch (Ω). I outlined how the learning branch supported the asymptotic barrier function in producing a consistent classification of the inputs within a neighbourhood of real samples from the domain. I directly outlined how the asymptotic barrier function focused on the assigned (dominant) class output values while the learning branch focused on all the interactions of the non-dominant class values for each class. The branching function included two parameters, the hyperparameter ζ and the parameter τ that is trained with the network,

that together drive the non-dominant class rows of the Jacobian matrix towards zero. I closed the section with some parallels with critical learning periods—a common neurobiological constraint in humans and animals. This completed my discussion of the second most important design decision for asymptotic GNNs.

The final section outlined the basic setup of the substructure processing as a pair of fully-connected neural networks each with a single hidden layer. The first network encoded the image patches into distributed representations while the second converts these image patch representations into probabilities over the K image classes. The final step of this substructure processing pooled the patches to the required node count for processing by the asymptotic barrier function, product aggregator and learning branch downstream. This concludes my discussion of the asymptotic GNN model. In the next chapter, I introduce the framework, evaluations, comparisons, and overall experimental methodology as well as my first set of baseline experiments.

Chapter 3

Baseline Experiments

In the previous chapter, I gave a detailed description of the essential design decisions of asymptotic graph neural networks (GNNs) to situate the experiments and analyses that occur over this and the next two chapters. There are two primary goals of this chapter. The first is to introduce the evaluations, techniques, and general methodology of all of the experiments that follow. The reasoning for this is that it simplifies the explanation of all subsequent experiments. The second goal is to evaluate the simplest instantiation of GNNs that could produce results that are suggestive. The minimum requirements to demonstrate this are that the asymptotic GNN outperforms at least one of the models on each adversarial attack. This elementary objective makes it easier to evaluate what the impact of the asymptotic layer is without incorporating every state-of-the-art (SOTA) modification that has ever been used in this domain.[1] I see the growing trend of applying this "kitchen soup" approach to modeling—where all of the latest techniques from the field are applied in combination often without any ablation studies—as a major shortcoming of the field. If different model elements are combined without serious consideration of their effects, then it is unknowable what is producing the observed effect. Consequently, each of the experiments and analyses that occur over the next three chapters are intentionally incremental. Although this means that I will accomplish less over the next three chapters, my assumption is that

[1] My underlying assumption here is that I could get better results by incorporating many of these modifications. Preliminary experimentation did support this idea.

these accomplishments will be more meaningful and thereby more impactful.

I will compare my asymptotic GNN to the results of four models from the literature. The first model is a multilayer convolutional neural network (CNN) classifier that can achieve nearly SOTA results on the the MNIST dataset despite being relatively simple and uniform in architecture. I borrow the basic architecture of the CNN from Schott et al. (2019) who also used this architecture as a baseline for their adversarial robustness results on the MNIST dataset. I also compare the asymptotic GNN to three other models that have achieved SOTA results on the MNIST dataset with the evaluations of adversarial robustness that I use in this chapter. As a set, these four models will act as a reference point for how my asymptotic GNN fits into the wider literature on adversarial robustness in machine vision.

In what follows, I discuss each of the models that are compared to my asymptotic GNN. I then outline the technical details of three SOTA evaluations of adversarial robustness, all of which have been applied to the MNIST dataset (Schott et al., 2019). After describing the evaluations, I discuss two quantitative techniques that I will use to compare the results of the models on each of the evaluations of adversarial robustness. I then outline the methods and results of my experiments. I close the chapter with a discussion of these results and a brief summary of my findings. My hypothesis is that the asymptotic GNN will get better results than at least one of the other models on each of the evaluations. The next section discusses the other models used in my comparisons.

3.1 Baselines and SOTA Models

There are four models that I use throughout this chapter. The first model is a multilayer CNN defined by Schott et al. (2019). This is the only model that I re-implemented. The second is another multilayer CNN that leverages a data augmen-

tation process during training to improve adversarial robustness (Madry et al., 2018). The other two models are ensembles of variational autoencoders (VAEs) that use an analytic technique to compute a 'best guess' for each noisy input image (Schott et al., 2019). Although I will refer the interested reader to the associated papers for the details of these models, I want to highlight the architectural constraints and assumptions assumed by each of them.

Both the non-augmented CNN that I implemented based on Schott et al. (2019) and the augmented CNN that Madry et al. (2018) implemented are convolutional. Consequently, both of the models incorporate the RIB of convolutional layers and the associated invariance to spatial translation that I discussed in Chapter 1.

At the organizational level, the non-augmented CNN has a relatively uniform, all-convolutional architecture with exponential linear units (ELUs) and batch normalization (BN) at every layer (except the last layer for BN). Although the decision to use ELUs and BN does influence the model, these two techniques do not constrain the possible relationships between the inputs like a CNN layer. Consequently, they are not claims about RIBs (Battaglia et al., 2018). Although the filter number, kernel size, and step size of each layer is relatively unique and has a specific order, there is little theoretical insight that can be derived from these decisions without a detailed mathematical analysis. Such an analysis is outside of the scope of the current project.

The augmented CNN by Madry et al. (2018) also has a relatively uniform architecture. The only constraint is the choice of a fully-connected layer for the final layer, which is a standard practice in the literature. The layer immediately before the output enables the model to make any and all relationships between the internal representations—produced by the earlier CNN layers—that will help select the correct class. However, the authors' choice to use a data augmentation approach to resolve adversarial robustness makes this model an ideal counterpoint to my nativist position. To understand what I mean by this last point, I will re-map the argument of

Madry et al. (2018) into the language of Chomsky's poverty of the stimulus argument.

As a quick caveat, it is worth highlighting that I think the work by Madry et al. (2018) has merit. Thus, although I am critical of the paper by Madry et al. (2018) for the next two paragraphs, my intended target is neither the authors nor their project. My target is a set of cultural assumptions that are unquestioned and pervasive in the literature. With that caveat, I will continue with my critique as it helps motivate the contributions of my project.

Madry et al. (2018) make the following four claims. First, the authors define a set of possible grammars over the problem of adversarial examples. The authors use an augmented empirical risk minimization formulation[2] to define the set of grammars (i.e., the set of available models of the problem domain). Second, Madry et al. (2018) select one of the possible grammars to model. The authors choose to model the set of image perturbations in terms of the L_∞ ball. I explain the related L_∞ norm in the next section. Third, the authors optimize their model to better handle the perturbations afforded by their grammar. Fourth, the authors find that the memory of the model (i.e., its storage capacity) is essential for accurately handling adversarial examples. I am going to go through this argument one more time to make what is happening perfectly clear.

First, a claim is made that *assumes* the problem of adversarial examples is a problem of inadequate *experience*. To be fair, given the learning theory slant of the paper, this is a very reasonable assumption for the authors to make. My issue is that the assumption is *inconsistently* and *uncritically* applied. If one is assuming a pure nurturist position, then it is a mistake to use a convolutional layer *or* one needs to strictly specify how that operation informs the model. To do both without even remarking on the fact is problematic. Second, the authors select the types of experiences that they think are missing. Third, they (effectively) design a model

[2]See Vapnik (1992).

that can learn those experiences. And fourth, they conclude that it is the ability to hold more experiences that is essential for resolving the problem. This has all the trademarks of a nurturist argument.

What is especially interesting, given this context, is the follow up paper by Schott et al. (2019). In effect, Schott et al. (2019) demonstrate that the experiences that lead to the robustness of the augmented CNN used by Madry et al. (2018) entirely misses the point of adversarial robustness. Instead of learning how to model the data domain, the augmented CNN learns to model noise. That is, it learns to store all the various perturbations that Madry et al. (2018) defined as being relevant to the domain. One of the ways Schott et al. (2019) demonstrate this point is by incorporating a large number of evaluations across multiple different grammars (i.e., representations of magnitude; e.g., L_0, L_2, and the L_∞ norms). Schott et al. (2019) then propose a pair of models—which just so happen to be the other two models that I use in this chapter—that are exactly the opposite of the augmented CNN. Instead of being nurturist, these models are overwhelmingly nativist.

The models proposed by Schott et al. (2019) assume that *a lot* of architectural constraints pre-exist learning and cannot be modified by learning. For example, the Analysis-By-Synthesis (ABS) models—which come in binary and non-binary forms—are effectively an ensemble of variational autoencoders (VAEs). By design, each VAE in the ensemble learns to uniquely represent a single class of digits from the MNIST dataset. After training, classification reduces to an encoding step and a comparison step. The encoding step transforms the image into an optimal internal representation that is computed analytically from a set of candidate samples. This representation is "optimal" in the sense that it produces the highest possible output score that can be achieved for a given VAE for that image. The comparison step uses a modified softmax to determine which of the optimal representations is the best guess across all of the VAEs in the ensemble. I discuss some of the relational inductive biases (RIBs)

of these strong architectural constraints after considering some of the similarities between the ABS and my asymptotic GNN.

The ABS setup should sound very familiar. The encoding step uses the encoder of a variational *autoencoder* to represent the image just like ϕ for the asymptotic GNN. The ABS trains this encoder with a regularized, autoencoder loss function.[3] My basic setup of ϕ for the asymptotic GNN also uses an autoencoder loss to train ϕ, although one that is not regularized. In the comparison step, the ABS uses a modified softmax that is very similar to my learning branch: it includes a term like ζ^4 and a multiplicative term like τ.

Although there are a lot of similarities between the two models, the underlying reasoning and related RIBs are often very different. First, the autoencoders of the ABS models encode entire images instead of image patches like the asymptotic GNN. This entirely skips the arbitrary RIB and associated permutation invariance provided by the GNN layer. Second, the ABS model incorporates primitives into its structure by making each class its own VAE. Although these primitives can have a lot of variation within each VAE's representation of the class, the structure of the model *implicitly* specifies the kinds of values (or distributions of values) that are acceptable for each VAE. By contrast, the convolutional layer, the GNN layer, and the asymptotic layer do not provide any constraints on the values of their kernels, nodes, or substructures, respectively. Consequently, although the separate VAEs assume strong architectural constraints, these constraints are very different from the asymptotic GNN.

This approach is also problematic from a psychological perspective. In effect, one would have to conclude that black-and-white representations of numbers are separately and independently encoded in the brains of humans. These representations would also need to be distinct from color representations of digits, styles of writing

[3]This is the standard setup for VAEs. For additional details, see the original paper by Kingma and Welling (2013).

[4]The authors describe this behaviour as "converging towards a uniform distribution," which is an accurate probabilistic interpretation of the behaviour (Schott et al., 2019).

and probably even different simple transformations of digits. This situation quickly deteriorates into a combinatorial explosion, which is not biologically nor neurally plausible.

Beyond the encoder networks, there are also a number of differences between the modified softmax proposed by Schott et al. (2019) and my learning branch. At base, the objectives are radically different. The intent of their ζ-like parameter is to flatten the values of the dominant class as it often exhibits sharp transitions between high and low probability inputs. These sharp transitions in the class distribution partially result from training each VAE on a single class. By contrast, ζ enables the learning branch to flatten the non-dominant values—a completely different objective from the ζ-like parameter of the ABS models. Similarly, the ABS parameter like τ enables Schott et al. (2019) to compare their model to models with a different loss function. Specifically, models that use cross-entropy instead of the evidence lower-bound (ELBO) (Schott et al., 2019). This radically departs from τ's role in the learning branch, which regulates the action of ζ to produce a critical learning period for the model.

To summarize, the four models—non-augmented CNN, augmented CNN, ABS, and binary ABS—each entail different sets of assumptions. All four models use convolutional layers. The augmented CNN implicitly takes a nurturist perspective, eschewing architectural constraints for an augmented training regime. Finally, the ABS models make very strong architectural claims that are ultimately differ from my asymptotic GNN despite superficial similarities. Consequently, the experiments that follow enable the following three theoretical comparisons:

1. The difference between substructure invariance (asymptotic GNN) and spatial translation invariance (all other models).

2. The difference between using architectural constraints (asymptotic GNN) and using data augmentation (augmented CNN).

3. The difference between using architectural constraints (asymptotic GNN) and using primitives (ABS and binary ABS).

In light of these comparisons, my hypothesis in this chapter assesses whether and to what extent architectural constraints *in general* contribute to adversarial robustness and whether and to what extent substructure invariance *in particular* contributes something unique to adversarial robustness relative to spatial translation invariance. This completes this section. The next section describes three adversarial attacks for three different 'grammars' (i.e., distance norms) that are all SOTA on the MNIST dataset.

3.2 Evaluations of Adversarial Robustness

All of the evaluations that I discuss in this section are different techniques for perturbing an image such that it is incorrectly classified by the models. The perturbed image is called an "adversarial example" while the techniques are called "adversarial attacks" (Szegedy et al., 2013). Each adversarial attack minimally accepts as input the image that will be perturbed, the correct class of that image, and a target model that should incorrectly classify that image. The attack then uses a well-defined procedure to modify the pixels in the image to produce the adversarial example.

I discuss the basic procedures of three different adversarial attacks below—one for each of the three common L_p norms used in the literature. The attacks are the Momentum Iterative Method (MIM) attack for the L_∞ norm (Dong et al., 2018), the DeepFool Attack (DFA) for the L_2 norm (Moosavi-Dezfooli, Fawzi, & Frossard, 2016), and the PointWise Attack (PWA) for the L_0 norm (Schott et al., 2019). All three of these techniques are implemented in the Foolbox v1.8 library, which I use for all my comparisons (Rauber, Brendel, & Bethge, 2017). I explain each of the norms and associated attacks below.

The Moment Iterative Method (MIM) is a gradient-based attack that won the NIPS 2017 adversarial attack challenge (Dong et al., 2018). Where gradient-based learning algorithms use the gradient to minimize a loss function by changing the parameters of a model,[5] gradient-based attacks use the gradient to maximize the loss by changing the input values while the model parameters stay fixed. For example, the iterative Fast Gradient Sign Method (IFGSM) examines the gradient to determine whether increasing (positive gradient) or decreasing (negative gradient) the value of each pixel would increase the loss. It then changes all of the pixels by a small quantity (ϵ)[6] in the direction of the sign (positive/negative) of the gradient (Kurakin, Goodfellow, & Bengio, 2016).

Figure 14: A random pick of adversarial examples created using the MIM adversarial attack on the non-augmented CNN. The L_∞ distance varies between 0.13 and 0.31. All of these images were incorrectly classified by the non-augmented CNN model.

The MIM attack builds off of the IFGSM, but adds a momentum term that scales the step size based on the gradient of the past t iterations. It minimizes the L_∞ norm, which scores the largest normalized difference (i.e., values range between 0 and 1) on any one pixel between the original and adversarial images. Abstractly, one can think of the MIM attack as changing the original image so it is less like the correct class *in general* (i.e., higher overall loss) while limiting the maximum change to any single individual pixel. Figure 14 provides example output of this attack.

The DFA is a gradient-based attack (Moosavi-Dezfooli et al., 2016) that minimizes the L_2 norm (i.e., Euclidean distance; scores range between 0 and $(N^s)^{\frac{1}{2}}$, where N^s is the number of pixels) that also shows improvements over the FGSM family of attacks using a different approach than MIM. The DFA computes the tangent of the gradient local to the original image then greedily changes the image in the direction to the

[5]For example, the weights and biases of a neural network can be changed to minimize the cross-entropy loss between the output class values and the true class values of an image.

[6]I use the Greek letter ϵ to stay consistent with the literature. This ϵ has no relation to the ϵ in my analysis of the asymptotic barrier function in Chapter 2.

closest hyperplane separating the classes. In effect, this changes the original image so that it resembles an image from the incorrect class that it is most similar to at each time step. For example, a three might be changed into an image that is more eight-like by partially closing the left openings. Figure 15 provides example output of this attack.

Figure 15: A random pick of adversarial examples created using the DFA adversarial attack on the non-augmented CNN. The L_2 distance varies between 1.55 and 1.56. All of these images were incorrectly classified by the non-augmented CNN model.

The PWA is a decision-based attack (i.e., it does not rely on gradient information) that minimizes the L_0 norm in a greedy fashion (Schott et al., 2019). The L_0 norm is a count of the number of pixels different from the original image, which ranges between 0 and N^s. This attack starts by adding salt and pepper noise until the image is misclassified. It then (greedily) searches through the perturbed pixels for noise that can be removed while preserving the model's misclassification. The attack ends when none of the pixels can be reset to the original image. The authors suggest re-running the attack ten times before picking the result with the smallest number of pixels changed. I use the same approach. Figure 16 provides example output of this attack.

Figure 16: A random pick of adversarial examples created using the PWA adversarial attack on the non-augmented CNN. The L_0 distance varies between 12 and 13. All of these images were incorrectly classified by the non-augmented CNN model.

For each of these three attacks, I also perform a transfer attack. This simple additional evaluation takes the adversarial examples created using one of the models (e.g., the non-augmented CNN) and tests whether or not the images are also adversarial examples for another model (e.g., asymptotic GNN). Although this attack does not contribute to my core hypotheses in this chapter as the results are much harder to interpret, the transfer attack provides additional theoretical insights by evaluating

the relative strength of the adversaries and the relative robustness of the associated models. To make the evaluation more informative, I restrict each transfer attack to images that both models could correctly identify before they were perturbed. An image that is incorrectly identified by a model is considered to be a trivial 'adversary,' so the attacks will not modify it further. The end result is that transfer attacks vary between 0 and 100% regardless of the original accuracy of the model.

Although there are many more adversarial evaluations, these four provide a good sampling of the state-of-the-art across three different norms. One important caveat to all the results that follow is that I only preliminarily explored the impact of different advesarial attack 'hyperparameters.' It is probable that extreme changes to these values could produce different results. However, the default parameters are really the only ones used in the literature to date and thus my results and conclusions are still strongly suggestive.

3.3 Quantitative Comparisons

There are two quantitative evaluations that I use to compare each of the models. Both of them measure differences in the ability of the models to correctly classify perturbed images produced by each of the evaluations. They are called the "median adversarial distance" and the "bounded perturbation accuracy" (Schott et al., 2019). Both of these quantitative comparisons are used on the test set of images.

The median adversarial distance is the perturbation size at which a model achieves 50% accuracy. To perform this comparison, the distance of every adversarial example from its input image is computed for a single model on the images from the test set. If a given evaluation could not produce an adversary (i.e., no perturbation led to a misclassification by the associated model), its distance is set to infinity. If the image was already misclassified, the distance is set to 0. After sorting the distances from

lowest to highest, I then select the smallest valid distance that puts at least 50% of the images above it. Consequently, if I were to exclude all of the perturbations of that distance or higher, the model would correctly classify at least 50% of the images correctly (i.e., all of the images with perturbations greater than that distance).[7]

The bounded perturbation accuracy is similar to the median adversarial distance except, instead of picking the distance based on the number of correctly classified images, a bound is selected *a priori*. The standard conventions for the bounds are 12, 1.5, and 0.3 for the L_0, L_2, and L_∞ norms, respectively. I have chosen to use the same values as they enable comparisons with previous research. However, the median adversarial distance is generally a more nuanced evaluation and should be preferred throughout the rest of the work.

3.4 Experiments

I compare my model to a CNN that was used by Schott et al. (2019) as a baseline for their experiments (see Appendix 2 for implementation and Appendix 3 for training details). All four attacks are run against the non-augmented CNN and the basic asymptotic GNN. I also report the results of three SOTA models from Schott et al. (2019) on all of the attacks except the transfer attack. I report the median adversarial distance and the bounded perturbation accuracy for each attack.

3.5 Methodology

I train the CNN and asymptotic GNN a single time after hyper-parameter tuning (see Appendix 4 for a list of all the hyper-parameter values). I use the standard train-test split for the MNIST database of black-and-white handwritten digits (LeCun et

[7]This is mainly a concern for the PWA attack and the L_0 norm, which outputs discrete values. For this attack, the number of images is greater than 50%.

al., 1998). It has 50,000 training images (5,000 per digit) and 10,000 test images (1,000 per digit). The autoencoder of the asymptotic GNN is trained for 20 epochs with Adam then the parameters are frozen.[8] The rest of network is then trained for another 20 epochs using a different Adam optimizer. The non-augmented CNN is trained for 5 epochs with its own Adam optimizer. All of the evaluations are applied to the asymptotic GNN and the non-augmented CNN on the test images, which have never been seen by either model.

3.6 Results

The results support my hypothesis: for each adversarial attack, the asymptotic GNN outperformed at least one of the models (see Table 1). I have included the scores of the L_∞ defense by Madry et al. (2018), the binary ABS model, and the regular ABS model as reported by Schott et al. (2019). The asymptotic GNN outperforms all of the models on the DFA and MIM. The asymptotic GNN outperforms the augmented CNN on the PWA and performs slightly worse than the non-augmented CNN on the bounded perturbation accuracy of the PWA. The asymptotic GNN is also robust to the DFA transfer attack from the non-augmented CNN (see Table 2). The asymptotic GNN is modestly robust to the other transfer attacks. The non-augmented CNN is also robust to the adversaries produced by the asymptotic GNN on the PWA. Finally the asymptotic GNN has the lowest clean accuracy (3.2% less than the augmented CNN).

[8] See Appendix 3 for additional training details.

	CNN	Madry et al.	Binary ABS	ABS	GNN
Clean	98.9%	98.8%	99.0%	99.0%	95.7%
L_2-metric ($\epsilon = 1.5$) DeepFool	1.23/29.6%	9.0/91%	*1.4/41%	*2.4/83%	∞/95.7%
L_∞-metric ($\epsilon = 0.3$) MIM	0.08/0%	0.34/90%	*0.46/85%	*0.26/8%	∞/95.7%
L_0-metric ($\epsilon = 12$) PointWise Attack 10x	11.0/47.1%	4.0/0%	36.5/82%	22.0/78%	11.0/43.9%

*A gradient does not exist for these models so an estimate of the gradient was used by the original authors.

Table 1: Results for the baseline, non-augmented CNN, asymptotic GNN and three models reported in Schott et al. (2019) on the three adversarial attack evaluations. The median adversarial distance is reported on the left of the slash of each cell and each model's percent accuracy for thresholds $L_2 = 1.5$, $L_\infty = 0.3$, and $L_0 = 12$ are reported on the right for the relevant evaluation. The asymptotic GNN outperforms all of the models on the DFA and MIM. The asymptotic GNN only outperforms Madry et al. (2018) on the L_∞ defense for the PWA. These results are consistent with my hypothesis that the asymptotic GNN would outperform at least one SOTA model on each of the adversarial attacks.

	Original Model	CNN	GNN
Clean	N/A	98.9%	95.7%
DFA	CNN	0%	94.3%
MIM	CNN	0%	89.8%
PWA	CNN	0%	88.0%
	GNN	97.8%	0%

Table 2: Results when transferring the adversarial examples learned on one model to another model. The percent correct is reported for the source model (left) and the target transfer model (top) with no threshold on the attack. The asymptotic GNN is robust to the DFA transfer attack. It is also modestly robust to the other two attacks. The CNN can correctly classify almost every adversarial example produced by the PWA with the asymptotic GNN as the source model.

3.7 Discussion

Recall that the main goal of this section is to get a rough sense of the effectiveness of the asymptotic GNN. To this end, I proposed the modest hypothesis that the asymptotic GNN would outperform at least one SOTA model on each of the adversarial attacks. This hypothesis was trivially achieved for all of the core adversarial attacks in Table 1. The real significance of these experiments is not in the minor differences between the results of these adversarial attacks. *Instead*, the central importance of these evaluations is in understanding how these results inform the underlying relational inductive biases (RIBs) built into the different architectures of the models.

Towards this effect, I outlined three key theoretical comparisons in Section 3.1 of this chapter. I repeat them here for convenience:

1. The difference between substructure invariance (asymptotic GNN) and spatial translation invariance (all other models).

2. The difference between using architectural constraints (asymptotic GNN) and using data augmentation (augmented CNN).

3. The difference between using architectural constraints (asymptotic GNN) and using primitives (ABS and binary ABS).

Although the results of the experiments in this chapter are very preliminary, these comparisons are coarse enough to derive insights from them.

3.7.1 Substructure VS Spatial Translation

Each of the models other than the asymptotic GNN incorporate convolutional layers into their design. Thus, an overwhelming architectural pattern that is common to every model other than the asymptotic GNN is an invariance to spatial translation.

By contrast, the essential architectural constraints of the asymptotic GNN is its substructure invariance. The difference between these two constraints can be summarized as follows.

The spatial translation invariance tries to learn patterns that can occur at any position within its fixed frame of reference supplied by the pixel grid. These patterns occur at each layer in the network, resulting in a simple compositional architecture. However, the architecture assumes that translation invariance defined over a fixed grid is the best invariance at every level in a uniform model (e.g., the non-augmented CNN from this chapter). Alternatively, if the model incorporates multiple different types of layers without sufficient analyses, then the behaviour of the network as a whole is ambiguous.[9]

By contrast, the asymptotic GNN takes a fundamentally different approach. Within the substructure of a node anything goes. However, the inclusion of the asymptotic layer gives the researcher and the model some nice guarantees. It allows them to ignore variation in the substructure to the extent that the asymptotic layer can get the Jacobian matrix between the substructure and base levels approach zero.

What does all of this mean for the current evaluations and adversarial robustness? The results in Table 1 support my hypotheses. The gradient-based adversarial techniques cease to function when the Jacobian matrix is sufficiently close to zero. This is all the more impressive given how simple and small my proposed model is. The basic asymptotic GNN only has 169,043 parameters. By contrast, the non-augmented CNN has 249,228 parameters—almost 33% more than the asymptotic GNN. Additionally, in preliminary research I was able to get decent results with less than a quarter of the current number of parameters. Future research will need to explore this phenomenon in more detail.

To summarize, the results show support for the importance of asymptotic layers

[9]For additional evidence that spatial translation invariance is not preserved by stacked convolutional layers, see Zhang (2019).

for adversarial robustness to gradient-based attacks. Additionally, spatial translation invariance in whatever form it manifests in these models is orthogonal to adversarial robustness. With that, I will reserve further discussion of the difference between substructure invariance and spatial translation invariance for my analyses in Chapter 5. The next theoretical comparison evaluates the nativist and nurturist approaches to adversarial robustness.

3.7.2 Architecture VS Data Augmentation

From the second section of Chapter 1, I have made my theoretical commitments to visual nativism overt and explicit. However, this analysis provides the first opportunity to actually compare the empirical results between two models that endorse different sides of this division: my asymptotic GNN and the augmented CNN from (Madry et al., 2018).

Since I spent a good portion of Section 3.1 critiquing the assumptions of this model and approach, I will refrain from repeating my points here. The key takeaway is that the inconsistent and uncritical application of assumptions to the modeling process is problematic.

The results show a trend: on every evaluation the augmented CNN performed worse than the asymptotic GNN. And, this is despite the fact that *not a single adversarial perturbation* was included in the training data.[10]

3.7.3 Architecture VS Primitives

The purpose of this section was to compare the results between the ABS models and my asymptotic GNN. One of the main differences between these two models—despite their many similarities—is the presence of primitives in the ABS models. I

[10]I evaluate a variation of my asymptotic GNN that incorporates Gaussian noise in the training process in Chapter 5.

argued in Section 3.1 of this chapter, that training a unique VAE for each class was equivalent to including pre-existing primitives into the architectural constraints of the model. Thus, this evaluation tries to differentiate the relative impact of choosing to use primitives instead of these constraints.

In this case, the results are not as interpretable as they were for the other theoretical comparisons. However, the key difference between the ABS models and every other model is their ability to get very good results on the PWA. In this regard, they outperform both the asymptotic GNN and all of the other models.

One interpretation for these results is that the separation of the VAEs into class-specific networks (hence, primitives) simplifies the internal representations of each model. Instead of modeling the joint distribution across all of the classes, each network can focus on the marginal distribution of a single class. This is almost always an easier distribution to model. Additionally, small perturbations like salt and pepper noise are less likely to be able to hit interesting transition points that exist within the larger representation space defined by the ensemble. Even if one pixel perturbs one of the VAEs in the ensemble it is unlikely to perturb all of them. In this sense, the ensemble acts almost like a normalization term: it reduces the relative impact of each pixel added by the PWA. This is an empirically testable hypothesis that I leave for future work.

In terms of the broader theoretical impact, the results are not surprising. If one picks the right primitives they will substantially improve the model. However, if one picks the wrong primitives, then they might never find a good model—at least until they change their primitives. The relative change in scores across the two ABS models (e.g., the binary ABS is much better at the MIM while the regular ABS is much better at the DFA) suggests that selecting the right primitives in this domain is a non-trivial problem. This completes my analysis of the basic asymptotic GNN.

3.8 Summary

There were two key goals of this chapter. First, I provided a detailed description of my experimental methodology. This included a description of the models I compared the asymptotic GNN to, the techniques for evaluating adversarial robustness, and the quantitative comparisons for evaluating the differences between the models for each adversarial attack. The second goal was to evaluate whether the asymptotic GNN could successfully outperform at least one SOTA model on each of the core evaluations of adversarial robustness. The asymptotic GNN trivially met these requirements.

On the basis of these results I derived three key theoretical conclusions. First, I showed that substructure invariance is a better indicator of adversarial robustness than translation invariance. Second, I showed that data augmentation is not the only means for achieving adversarial robustness. Finally I discussed how the ABS models demonstrate the main strength and weakness of using primitives. The right primitives are often very helpful but the wrong primitives can entirely block progress on the problem. The next chapter evaluates the relative contribution of the asymptotic barrier function and learning branch to adversarial robustness.

Chapter 4

Evaluating the Essential Structure of Asymptotic GNNs

In the previous chapter, I evaluated the basic behaviour of the asymptotic GNN. The primary goal of this chapter is to evaluate the relative contribution of the asymptotic barrier function (Γ) and the learning branch (Ω). At present, it is not clear how the different components of the model are contributing to the results.

Preliminary research exploring the viable modifications to the asymptotic barrier and learning branch revealed that most changes result in the catastrophic failure of the model. For example, if one replaces the product aggregator with a normalized sum, the model ceases to work at all. The same is true if the natural logarithm is replaced with a non-asymptotic function, including the Gumbel-Softmax—a state-of-the-art (SOTA) approximation of the categorical distribution (Jang, Gu, & Poole, 2017). Consequently, the set of viable ablations of the model was very limited.

In what follows, I discuss the details of the ablation of the asymptotic barrier and the ablation of the learning branch of the asymptotic GNN. I emphasize the theoretical implications of these changes and focus on two key theoretical comparisons. I then provide a brief review of the experimental design and methodology before reporting the results. I close the chapter with a discussion of my findings, especially as they relate to the two key theoretical comparisons. I then provide a brief summary before closing the chapter.

4.1 Ablation Models

Recall that the central role of the asymptotic barrier function is to make the superstructure level insensitive to changes in the substructural level while maintaining sensitivity to the base level. The result is what I called substructure invariance. The way I chose to model this was by making the asymptotic barrier function approximate the behaviour of the unit step function $H(x, \delta)$. To enable the approximation of the unit step function I specified three of its key behaviours in terms of three approximation conditions: the zero gradient (ZG), short transition window (STW) and large transition gradient (LTG) conditions. I will provide a quick review of each of these conditions from Chapter 2 to help clarify the theoretical impact of my chosen ablation of the behaviour of Γ.

The ZG condition described how most of the domain of the input to the unit step function was mapped to 0. This produced a large, flat region over almost all variations to the input that did not cross $1-\delta$. The other two approximation conditions emerged from the following observation of the unit step functions behaviour: the discontinuous behaviour of $H(x, \delta)$ can be thought of as an infinitely small change leading to an infinitely steep gradient. To approximate this behaviour, the size of the window where the transition occurs must be as small as possible (STW) and the gradient of the change over that short transition window needs to be as large as possible (LTG).

When I initially introduced my formulation of Γ, I highlighted two hyper-parameters: ψ and κ. The value of κ truncated the asymptotic non-linearity of the natural logarithm and assigned the maximum value of Γ's output. In my analysis of the multiclass behaviour of Γ, I then expanded the functional role of κ to include selecting the starting point of the multiclass dynamics. This enabled me to place the system into its most volatile state at the beginning of learning.

By contrast, the function of ψ was a lot more straightforward. Increasing ψ by a

whole number improved Γ's approximation of $H(x, \delta)$.[1] That is, it flattened the tail of the function (ZG), decreased the size of the transition window (STW) and increased the size of the gradient (LTG). I then added a caveat that approximating $H(x, \delta)$ too closely could lead to the same problems as the original unit step function—mainly, an intractable gradient. The value that I settled on for ψ in my implementation was 4. In my ablation of the asymptotic barrier function in this section, I decrease the value of ψ to 1. I reproduce the graphs of Γ from the single class section of Chapter 2 for each of the values of ψ, below, to illustrate how the behaviour changes (see Figure 17).

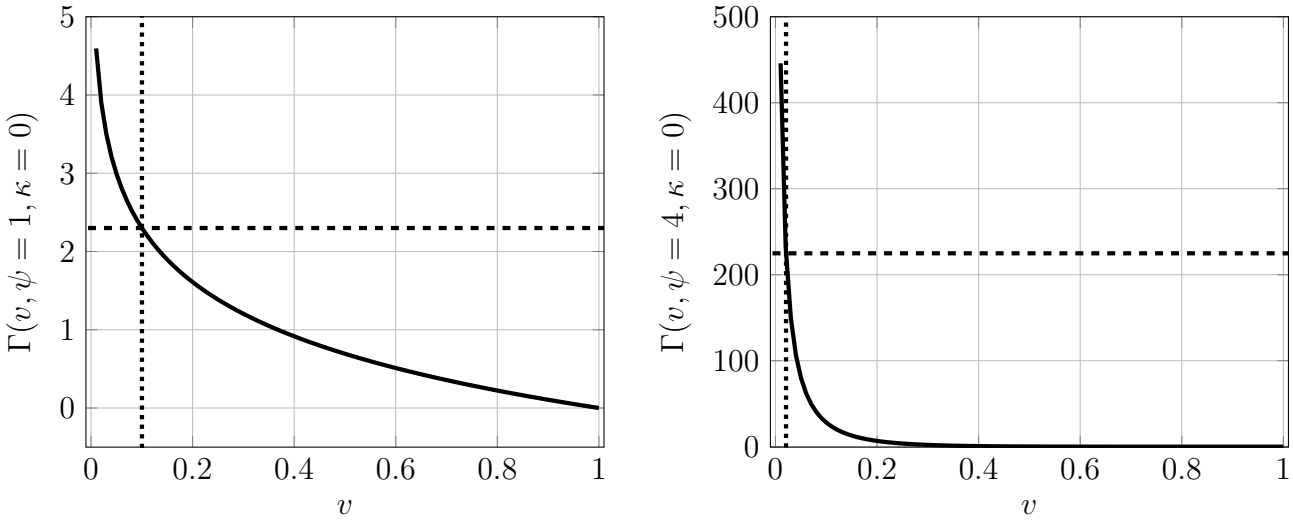

Figure 17: Reproduction of two of the plots of the asymptotic barrier function when ψ equals 1 (left) and 4 (right). The basic asymptotic GNN uses $\psi = 4$ while the ablation uses $\psi = 1$. Each graph has one hundred equally spaced samples from within the domain [0, 1]. Consequently, the maximum value displayed for each graph has 0.01 as its input value for v. The horizontal dashed line is the central output value of Γ between the inputs $v = 1$ and $v = 0.01$. It models one possible approximation threshold (t_h) over the input domain. The vertical dotted line captures the intersection between t_h and the graph. It shows the value of μ (t_v) for the given t_h. Together, these dashed lines help one observe the effect of ψ on the approximation conditions: ZG, STW, and LTG. Critically, as ψ increases, t_v shifts to the left (i.e., gets smaller). This leads to an increasingly good approximation of $H(x, \delta)$.

The change that results from decreasing ψ to 1 is further exacerbated by the

[1] I outline a sketch of a proof in Appendix 1.

product aggregator ($\prod$). In my final implementation of Γ, I have a 3-by-3 grid of nodes (i.e., $N^v = 9$). This means that, within the product aggregator, ψ gets multiplied by N^v to produce the final *exponent*. Consequently, the change in the maximum output values is substantial, specifically from 1.10×10^{13} to 1.82×10^3 for $\kappa = 0.1, N^v = 9$. Thus, even though I do not technically ablate (i.e., remove) the asymptotic barrier function, this small change to ψ radically changes the behaviour of Γ in a way that is theoretically interesting. That is, it makes Γ more similar to the traditional GNN model while preserving much of the basic structure of the asymptotic GNN. This makes it a theoretically rich incremental change.

The second ablation for the learning branch is much simpler to understand. I set the maximum value of τ to 0, so that the right branch never gets used (i.e., the critical period never ends). Although this ablation is not as complicated as the modification to the asymptotic barrier function, it still enables me to evaluate what the learning branch uniquely adds to the behaviour of the asymptotic barrier function.

Although it might not seem like it, these two ablations are the 'true' test of all of my theoretical work on adversarial robustness. If my theory of the asymptotic hierarchical relational inductive bias (RIB) and its related substructure invariance is true, then a weaker approximation of $H(x, \delta)$ should severely impair the model. And, this should be reflected in the results of the adversarial attacks. If it is not, this result would falsify one or both of the central theories of my project. Consequently, my hypothesis is that the asymptotic barrier function ablation will severely impair the adversarial robustness of the asymptotic GNN. As the robustness of the model was strongest for the gradient-based adversarial attacks in Chapter 3, my hypothesis will focus on an impairment in these results, specifically.

To understand the underlying theoretical comparisons for the learning branch, one needs to recall my theory of its behaviour. Recall that in Section 2.6 I defined the learning branch as a means to constrain the non-dominant class values for samples

that share their dominant class. In particular, the learning branch makes the gradient of each non-dominant class output in the Jacobian matrix approach $\ln(\zeta)$ as τ increases over the course of learning. This effect is then differentiated by the current evaluations. In particular, how the DFA and MIM gradient-based attacks function.

Recall that the MIM attack perturbs the input image so it is less like the target class *in general* (i.e., larger overall error). By contrast, the DFA makes changes in the direction of the most similar image from a different class. That is, the behaviour of the DFA and MIM differs on whether they engage with the dominant or non-dominant class. The MIM focuses on dominant class by increasing the overall error while the DFA focuses on the non-dominant class by making the input resemble it. What this means, in light of the behaviour of the learning branch, is that the DFA should be less effective when the learning branch is being used than when it is not. When the learning branch makes the non-dominant class rows of the Jacobian matrix approach zero, the DFA will have no signal to indicate the direction of the nearest separating hyperplanes. Thus, my second hypothesis is that while both the DFA and MIM should be negatively affected by ablating the asymptotic barrier function, only the DFA should be affected by ablating the learning branch.

4.2 Experiments

I compare the basic asymptotic GNN to the asymptotic barrier function and learning branch ablations discussed above. All three attacks (MIM, DFA, PWA) are run against the ablation models, which are then compared to the asymptotic GNN's results from Chapter 3. My evaluation of the transfer attack compares the ablation models, the asymptotic GNN, and the baseline CNN results. I report the median adversarial distance and the bounded perturbation accuracy.[2] The bounds for the latter follow the standard conventions of 12, 1.5, and 0.3 for the L_0, L_2, and L_∞

[2]For a description of these quantitative comparisons, see Chapter 3 Section 3.3.

norms, respectively.

4.3 Methodology

I train each of the models a single time after hyper-parameter tuning (see Appendix 4 for a list of all hyper-parameter values for each of the ablation models). I use the standard train-test split for the MNIST database of black-and-white handwritten digits (LeCun et al., 1998). It has 50,000 training images (5,000 per digit) and 10,000 test images (1,000 per digit). All of the ablation models use the same encoder network (ϕ_α) that was trained for the asymptotic GNN in Chapter 3. All of the evaluations are applied on the test images, which have never been seen by any of the models.

4.4 Results

The results support both of my hypotheses (see Table 5). The asymptotic barrier function ablation has the lowest scores on the DFA and MIM of any model considered thus far. The learning branch ablation is robust to the MIM attack but scores poorly on the DFA attack. The asymptotic GNN and the asymptotic barrier function ablation have similar scores on the PWA while the learning branch ablation does slightly worse.

The transfer attacks show similar results (see Table 6). All of the models do equally well on the DFA CNN transfer. The asymptotic GNN is the most robust to all of the transfer attacks from the ablation models while the ablation models are more robust to the transfer attacks from the CNN. In general, the learning branch ablation does similar or worse than the asymptotic barrier function ablation on transfer attacks from the CNN and the asymptotic GNN.

The accuracy of both ablations is slightly higher than the asymptotic GNN for clean images.

	GNN	$\psi = 1$	$\tau = 0$
Clean	95.7%	97.2%	96.1%
L_2-metric ($\epsilon = 1.5$)			
DeepFool	∞/95.7%	0.91/0.10%	0.74/0.04%
L_∞-metric ($\epsilon = 0.3$)			
MIM	∞/95.7%	0.06/0.00%	∞/96.1%
L_0-metric ($\epsilon = 12$)			
PointWise Attack 10x	11.0/43.9%	10.0/44.1%	8.0/32.0%

Table 3: Reported results from Chapter 3 for the asymptotic GNN and current results for the ablation models, where $\psi = 1$ is the asymptotic barrier function ablation and $\tau = 0$ is the learning branch ablation. The median adversarial distance is reported on the left of the slash of each cell and the model's percent accuracy for thresholds $L_2 = 1.5$, $L_\infty = 0.3$, and $L_0 = 12$ are reported on the right for the relevant evaluation. The asymptotic barrier function ablation has the lowest scores on the DFA and MIM. The learning branch ablation is robust to the MIM attack but scores poorly on the DFA attack. The asymptotic GNN and the asymptotic barrier function ablation have similar scores on the PWA while the learning branch ablation does slightly worse.

	Original Model	GNN	$\psi = 1$	$\tau = 0$
Clean	N/A	95.7%	97.2%	96.3%
DFA	CNN	94.3%	94.6%	94.7%
	$\psi = 1$	86.7%	0%	67.4%
	$\tau = 0$	95.0%	86.3%	0%
MIM	CNN	88.0%	92.6%	94.0%
	$\psi = 1$	87.8%	0%	8.8%
PWA	CNN	88.0%	91.0%	88.3%
	GNN	0%	85.3%	81.5%
	$\psi = 1$	77.2%	0%	60.9%
	$\tau = 0$	88.7%	80.0%	0%

Table 4: Results when transferring the adversarial examples learned on one model to another model, where $\psi = 1$ is the asymptotic barrier function ablation and $\tau = 0$ is the learning branch ablation. The percent correct is reported for the transfer model (top) with no threshold on the attack. All of the models do equally well on the DFA CNN transfer. The asymptotic GNN is the most robust to all of the transfer attacks from the ablation models while the ablation models are more robust to the transfer attacks from the CNN. In general, the learning branch ablation does similar or worse than the asymptotic barrier function ablation on transfer attacks from the CNN and the asymptotic GNN.

4.5 Discussion

Recall that the main goal of this section is to evaluate the relative contribution of the asymptotic barrier function (Γ) and the learning branch (Ω) to the results of the asymptotic GNN. Towards this effect I created two hypotheses. The first was essential for my theory of the asymptotic hierarchical RIB, substructure invariance, and its link to adversarial robustness. The second hypothesis related to my theory of the learning branch. I discuss each of them in turn, below.

4.5.1 The asymptotic barrier ablation

The results support my hypothesis that the ablation of the asymptotic barrier function would severely impair the asymptotic GNN on the gradient-based attacks. The asymptotic barrier ablation has the worst scores on the MIM and DFA attacks of any model compared thus far. This suggests that the asymptotic barrier function—through its approximation of $H(x, \delta)$—is ultimately driving the robustness effects that I observed in Chapter 1. This makes a lot of sense in the context of classification tasks: if the predicted dominant class is not robust to perturbation, then it really does not matter how robust the non-dominant classes are. It will be easy to produce adversarial examples by perturbing the input to be less like the dominant class. This leads to my discussion of the results of the learning branch ablation.

4.5.2 The learning branch ablation

My hypothesis that the learning branch would have a separable effect on the DFA evaluation was supported by the results of this chapter. When the learning branch was ablated, the asymptotic barrier function was able to maintain its adversarial robustness to the MIM attack. But, the asymptotic barrier function alone was not sufficient for maintaining robustness to the DFA attack. Interestingly, the learning

branch seemed to have no positive effect in the absence of the asymptotic barrier function, despite the fact that τ did reach its maximum value (i.e., there was enough of an error gradient to warrant increasing τ during training).

4.6 Summary

The primary goal of this chapter was to evaluate the relative contribution of the asymptotic barrier function and the learning branch to the results of the asymptotic GNN from the last chapter. First, I motivated my two ablation models with a review of the key insights from Chapter 2 concerning the asymptotic barrier function and learning branch. I then gave a detailed description of the two ablation models and coupled them with two essential hypotheses for the current work. My first hypothesis was that the asymptotic barrier function ablation would not be robust on either of the gradient-based attacks. My second hypothesis was that the learning branch would only impair the DFA attack. Both of these hypotheses were supported by my experiments.

Chapter 5

Modeling the Inductive Bias of PWA

In the last chapter, I evaluated to what extent the asymptotic barrier function and the learning branch were producing the adversarial robustness of the asymptotic GNN. The experiments supported my proposed solution for adversarial robustness as well as many of the supporting theories related to this work. The primary goal of this chapter is to determine what inductive bias is related to the PWA evaluation of adversarial robustness.

Throughout all of the analyses of the previous two chapters, only the ABS models have been adversarially robust to the PWA. What is even more interesting is that the the ablations from the last chapter had minimal effect on the PWA results. Thus, the asymptotic hierarchical relational inductive bias (RIB) of the basic asymptotic GNN seems to capture something largely orthogonal to the PWA evaluation. At present it is not clear what property this dimension is capturing. However, guided by my theory of adversarial robustness, I have chosen to focus on modifications to the encoder of the asymptotic GNN (ϕ_α). My reasoning is as follows.

If the asymptotic GNN does in fact model substructure invariance by making the gradient between the substructure and base levels approach zero, then the measurable effect of the PWA on the asymptotic GNN *must* occur due to changes in the states of the base level nodes. As one will recall from Chapter 2, even the unit step function

will still change state if the modification (ϵ) is large enough that the value crosses $1 - \delta$. This is the fail case that I reflected on in Chapter 1, where certain gradients cannot be abstracted away without sub-dividing the base level nodes into sub-nodes made from additional stacks of 3-level hierarchies. In the absence of these additional architectural structure, these perturbations to the input will change the final class state through changes to the state of the intermediary nodes. This occurs when enough nodes change state in response to the perturbation.

The PWA evaluation uses the L_0 norm, which measures the distance between two images by counting the number of pixels that were changed. As a consequence, every change introduced by the PWA evaluation always sets the input to its maximum (salt) or minimum (pepper). If one assumes that images approximate a multivariate normal distribution over the pixels, then every modification to the image is a low probability (i.e., unlikely) event. Thus, an aggregate of a bunch of low probability events *should* perturb the system. Although this makes sense theoretically, the fact remains that humans can still easily identify images even when a lot of noise is added to them. So, what could be happening?

One theory is that most of this noise is filtered out *before* it even gets processed by the asymptotic barrier function. If this were the case, then the PWA attack really is orthogonal to the core research program of the asymptotic hierarchical relational inductive bias (RIB). The behaviour of interest would happen *within* the substructure processing (i.e., within ϕ). This is the main hypothesis that I evaluate in this chapter.

In what follows, I discuss two key design choices for ϕ that are good candidates for producing adversarial robustness to the PWA. I then provide a review of the experimental design and methodology before reporting the results. I close the chapter with a discussion of the main findings followed by a brief summary.

5.1 Alternative Design Choices for ϕ

I propose to add two additional design choices for ϕ in this section. Although these design choices affect the substructure processing instead of the core structures of the asymptotic GNN, the underlying hypothesis is essential for the core theory of the research program. If there is no reasonable solution or at least a viable path to a solution that relies on the substructure processing instead of the asymptotic barrier function or similar, then my theory of adversarial robustness could be seriously flawed and might even need to be abandoned in the worst case.

All of the design choices that I discuss below change the way the substructure processing represents information within the domain. That is, all of these modifications change or replace the basic encoder (ϕ_α) used by the asymptotic GNN. The first inductive bias that I consider adds noise to the input during training. This changes the reconstructive autoencoder loss to a denoising reconstructive loss. It is also the simplest change that could produce a tolerance to noise in the input. The second adds a denoising sampling step that is similar in broad strokes to Markov Chain Monte Carlo (MCMC) techniques. This is a slightly more complicated approach that has a few commonalities with the variational autoencoders (VAEs) used by Schott et al. (2019). I consider the details of each of these design choices, below.

The first and simplest modification to the basic asymptotic DNN that meets the current requirements is to replace the unregularized autoencoder implementation of ϕ_α with a denoising autoencoder (DAE; for details, see Vincent, Larochelle, Bengio, & Manzagol, 2008; Bengio, Yao, Alain, & Vincent, 2013). The DAE changes the basic structure of the autoencoder from a system that reconstructs to a system that corrects. To incorporate this corrective component, the model adds noise ($\boldsymbol{\eta}$) to the input image patch ($\boldsymbol{s}$) before it is input to the network. The reconstruction loss is then computed relative to the original image without noise. I sample this noise from a normal distribution with a mean of 0 and a standard deviation of 0.1 (i.e.,

$\boldsymbol{\eta} \sim \mathcal{N}(\mu = 0, \sigma = 0.1)$). Gu and Rigazio (2015) showed that both adding noise to the input and, specifically, using a denoising autoencoder improved robustness of models on the MNIST dataset, so this is a very reasonable choice for my modification to the asymptotic GNN. This completes my description of the DAE.

The second model filters noise by replacing the basic ϕ_α with an approach that resembles the MCMC technique known as Gibb's sampling.[1] The basic idea is to sample from the latent distribution of the unregularized autoencoder by adding noise (i.e., $\boldsymbol{\eta} \sim \mathcal{N}(\mu = 0, \sigma = 0.1)$) to the image before it gets processed by the autoencoder. The resulting perturbed image is then sent back through the encoder. This effectively re-samples a new image from the latent space that resembles the coarse structure of the original input.

For simplicity, I use a single sampling loop to sample a new image as follows.

$$\hat{\boldsymbol{x}} = \underset{i=1:N^s}{\text{aggregate}}(\phi_\gamma(\phi_\alpha(\boldsymbol{s}_i + \boldsymbol{\eta}, \theta_\alpha), \theta_\gamma)) \tag{5.1}$$

where ϕ_α and ϕ_γ are the encoder and decoder, respectively, from Section 2.7, $\hat{\boldsymbol{x}}$ is the full size, sampled image (as opposed to an image patch $\boldsymbol{s}$) and $\underset{i=1:N^s}{\text{aggregate}}$ is an operation that reconstructs the image patches into the original image while averaging—for simplicity—the overlapping parts of the image patches. Conveniently, Tensorflow and PyTorch have functions for constructing this aggregation operation (e.g., `extract_image_patches`, `unfold`). The interested reader can review the associated documentation for additional details. One can then compute ϕ relative to this new sample.

$$\phi(\hat{\boldsymbol{s}}, \theta_\alpha, \theta_\beta) = \phi_\beta(\phi_\alpha(\hat{\boldsymbol{s}}, \theta_\alpha), \theta_\beta) \tag{5.2}$$

[1] For a discussion of the parallels and differences between denoising autoencoders and Gibb's sampling, see Bengio et al. (2013).

where $\hat{\boldsymbol{s}}$ is an image patch from the image sample ($\hat{\boldsymbol{x}}$).

This approach, which I call a denoising sampler, has some similarities with the "best guess" approach used by the ABS models from Chapter 3 (Schott et al., 2019). Both approaches re-sample from the latent distribution to improve the representation of the image. The main difference is that the ABS model adds noise to the latent distribution ($\boldsymbol{z}$) instead of to the image.[2] Given the success of the ABS models on the PWA, this overlap makes the denoising sampler a strong candidate.

5.2 Experiments

I compare the basic asymptotic GNN to the DAE and the denoising sampler discussed above. All three attacks (MIM, DFA, PWA) are run against the DAE and the denoising sampler, which are then compared to the asymptotic GNN's results from Chapter 3. My evaluation of the transfer attack compares the DAE, denoising sampler, the asymptotic GNN, and the baseline CNN results. I report the median adversarial distance and the bounded perturbation accuracy.[3] The bounds for the latter follow the standard conventions of 12, 1.5, and 0.3 for the L_0, L_2, and L_∞ norms, respectively.

5.3 Methodology

I train each of the models a single time after hyper-parameter tuning (see Appendix 4 for additional details). I use the standard train-test split for the MNIST database of black-and-white handwritten digits (LeCun et al., 1998). It has 50,000 training images (5,000 per digit) and 10,000 test images (1,000 per digit). The DAE was trained from scratch in two steps like the basic asymptotic GNN. The denoising sampler uses the

[2]The ABS model uses variational autoencoders so they *always* sample from the latent distribution using noise. For further details, see Kingma and Welling (2013).

[3]For a description of these quantitative comparisons, see Chapter 3 Section 3.3.

encoder from the basic asymptotic GNN from Chapter 3. It incorporates the new resampling loop with the original parameters.

5.4 Results

The results support my main hypothesis (see Table 5). Both the DAE and the denoising sampler outperform the basic asymptotic GNN on the PWA evaluation. Both of the models also outperform the basic asymptotic GNN on all of the transfer attacks (see Table 6). Finally, all of the models perform similarly on the clean images.

	GNN	DAE	DS
Clean	95.7%	95.4%	94.9%
L_2-metric ($\epsilon = 1.5$) DeepFool	∞/95.7%	∞/95.4%	∞/94.9%
L_∞-metric ($\epsilon = 0.3$) MIM	∞/95.7%	∞/95.4%	∞/94.9%
L_0-metric ($\epsilon = 12$) PointWise Attack 10x	11.0/43.9%	13.0/54.4%	30.0/73.1%

Table 5: The denoising autoencoder (DAE) and the denoising sampler (DS) both outperform the basic asymptotic GNN on the PWA. The median adversarial distance is reported on the left of the slash of each cell and the model's percent accuracy for thresholds $L_2 = 1.5$, $L_\infty = 0.3$, and $L_0 = 12$ are reported on the right for the relevant evaluation.

5.5 Discussion

Recall that the main goal of this chapter was to evaluate what inductive bias was most strongly related to the PWA evaluation of adversarial robustness. My main hypothesis for this chapter was that adversarial robustness to the PWA could be achieved purely by making changes to the substructure processing of the asymptotic

	Original Model	GNN	DAE	DS
Clean	N/A	95.7%	95.4%	94.9%
DFA	CNN	94.3%	96.8%	97.4%
MIM	CNN	89.8%	94.3%	98.4%
PWA	CNN	88.0%	93.2%	96.1%
	GNN	0%	92.9%	97.5%
	DAE	24.0%	0%	95.9%
	DS	31.2%	43.5%	0%

Table 6: Results when transferring the adversarial examples learned on one model to another model. The percent correct is reported for the transfer model (top) with no threshold on the attack. Both the denoising autoencoder (DAE) and the denoising sampler (DS) outperform the asymptotic GNN.

GNN. As a consequence of this, I proposed two simple design choices that would only affect the substructure processing: the denoising autoencoder (DAE) and the denoising sampler. Both of the models outperformed the asymptotic GNN on all of the PWA evaluations, including the PWA transfer attacks from each model. In fact, the denoising sampler achieved the second highest score on the median adversarial distance of the PWA across every model that I evaluated in this project. The previous highest and second highest median adversarial distance scores on the PWA were the Binary ABS at 36.5 and the ABS at 22.0. This lends additional support to my hypotheses.

For those who are interested in mapping this work to that of Gu and Rigazio (2015), the average distortion[4] on the test set of the DS, DAE, and GNN is 0.171, 0.128, and 0.119, respectively. All of these results are larger than their best reported result of 0.107. A larger average distortion is better as it shows that a larger neighbourhood around the test samples is consistently classified (i.e., the second condition of an adversarially robust network outlined by Papernot et al., 2016). However, in the original work the authors used the L-BFGS attack, which is a gradient-based attack

[4]The average distortion was originally proposed by Szegedy et al. (2013) and is calculated as $\sqrt{\frac{\sum(x'-x)^2}{n}}$, where x' is the adversarial example, x is the original image, and n is the number of pixels (e.g., 784 for MNIST).

originally proposed by Szegedy et al. (2013). My asymptotic GNN is not susceptible to this attack, so the PWA is at best a proxy for the average distortion on the L-BFGS. The results are still very promising. Finally, the most interesting result comes from the denoising sampler modification.

Recall that the core insight of the denoising sampler is that the basic autoencoder can be used to sample from the learned latent space. By adding noise from the normal distribution to the input, the model is able to find similar samples that resemble the original image but with less local variation. The reason there is less variation is that overlapping regions of the image patches are averaged together during the sampling process. This enables the full structure of the autoencoder, which already existed in the original asymptotic GNN, to be used to stochastically filter noise out of the input. Models that employ this kind of stochasticity are known to impair gradient-free attacks like the PWA, but there is no theory to date that explains why this is so (Carlini et al., 2019). I would speculate that the recurrent loop used by the denoising sampler can be used to make the input images more prototypical which reduces the impact of the low probability events modeled by the PWA. This result is not that surprising as feedback loops have already been shown to improve adversarial robustness (Orchard & Castricato, 2017). This completes my discussion of the inductive biases of the PWA.

5.6 Summary

The primary goal of this chapter was to determine the inductive bias that was most strongly related with the PWA. First, I leveraged insights from my theoretical work to make an informed guess about the most relevant avenues of research for answering this question. This led me to limit my consideration to design decisions that only affected the substructural processing of the model. My main hypothesis was that adversarial

robustness for the PWA could be achieved with design changes that only affected the substructure processing. This hypothesis was supported by my experiments.

Chapter 6

Conclusion

In the preceding five chapters, I outlined a new theory and computational model that explicitly describes the architectural constraints that produce adversarial robustness. My theory can be described as follows.

Adversarial robustness is an exemplar of a particular relational inductive bias (RIB), which I call the asymptotic hierarchical RIB. This RIB constrains the set of direct relationships across levels in a three-level hierarchy (substructure, base, superstructure) by introducing the asymptotic layer, which ultimately makes the Jacobian matrix approach zero between substructure and superstructure levels. By constraining where within the network architecture sensitivity is acceptable and where it is unacceptable, the details of the substructure can be abstracted away. In other words, the network is able to function as though its behaviour is fully defined by the states of the base level.

The asymptotic hierarchical RIB produces an invariance called substructure invariance, which functions similar to the spatial translational invariance and permutation invariance of the convolutional and GNN layer structures, respectively. But, unlike those other invariances, substructure invariance is defined over three-layer compositional hierarchies. In particular, it means that the superstructure level is insensitive to variations in the substructural level, but is still sensitive to changes in the base level. By extension, this means that the superstructure level is *only* sensitive to

changes in the substructural level that lead to changes in the states of the base level nodes.

My experimental evaluations over the past three chapters demonstrated three major findings.

1. The asymptotic GNN is able to produce state-of-the-art (SOTA) results on the MNIST dataset on two SOTA evaluations of adversarial robustness.

2. My ablations demonstrated that it is the asymptotic barrier function's approximation of the unit step function that uniquely contributes to adversarial robustness in general while the learning branch uniquely contributes to adversarial robustness on the DeepFool Attack.

3. Finally, the denoising sampler contributes to an orthogonal, stochastic form of robustness that is unrelated to substructure invariance, but is captured by the PointWise attack.

These findings support my theory of adversarial robustness by providing concrete examples of the key concepts in action.

Over the course of the past five chapters I have outlined a number of key definitions, three of these I will highlight explicitly below.

1. The **asymptotic hierarchical RIB**: The asymptotic hierarchical RIB constrains direct relationships between levels in a three-level hierarchy. This constraint is produced via the application of an asymptotic layer, which controls the flow of information from the substructure to the superstructure.

2. **Substructure invariance**: Substructure invariance is defined over a three-level hierarchy. It is produced by the asymptotic layer as it drives the values of the Jacobian matrix towards zero.

3. The **asymptotic non-linearity**: The asymptotic (truncated) non-linearity is a function defined over [0, 1] that approaches positive infinity as the input approaches 0. In its standard form it has two hyper-parameters that control how well it approximates the unit step function (ψ) and control its upper bound (κ). The asymptotic non-linearity is one of the key components of the asymptotic barrier function (Γ).

These definitions directly contribute to the field of deep learning, graph neural networks (GNNs), and analyses of inductive biases in neural networks.

I have made an effort to make my descriptions detailed and explicit enough to make re-implementing many of my models easy on contemporary deep learning platforms. However, I have implementations of the asymptotic and learning branchs in PyTorch that will be released with all of my code and documentation on Docker after publication of my results.

6.1 Limitations

There are a number of important limitations to this work. Four major ones are: the use of a single, black-and-white dataset, the reduced accuracy of the GNN, the effect of numerical instability and gradient masking, and the effect of the distillation hack on the ablation of the asymptotic barrier function. I address each of these below.

Most of the current research on adversarial examples is focused on CIFAR10 and ImageNet. These are much more complex datasets both in terms of content (e.g., a dog is more complex that the digit '5') and input (e.g., 3-channel colour images vs single channel black-and-white images). Consequently, it is a major limitation of the current work that my findings have only been shown on a single, black-and-white dataset that is extremely simple. In fact, there is some preliminary evidence that the MNIST dataset is amenable to a "closed-form" solution that focuses on binarization

of the input (Tramèr et al., 2017), so there is no guarantee that my findings will generalize. However, I do believe that there are some unique opportunities for applying the framework of the asymptotic GNN, especially once stacking is possible, to colour images. Each colour channel could, for example, be handled by a separate asymptotic GNN, the output of which is then combined by a higher-order asymptotic GNN. But, stacking the 3-layer network is, at present, a non-trivial engineering problem. Future research will need to explore many of these ideas in more detail.

Another major issue is that the asymptotic GNN never quite achieved state-of-the-art (SOTA) results on clean (i.e., non-adversarial) input. The base model achieved an accuracy of 95.7%, which is quite a bit lower than the standard 99% that is common for SOTA results on the MNIST dataset. More importantly, the ablation models had better results on the clean data: 97.2% for the asymptotic barrier ablation and 96.1% for the learning branch ablation. This suggests that the different parts of the asymptotic layer are actually impairing the accuracy of the model. One caveat to this is that I deliberately chose not to use any of the standard tricks of the trade in deep learning (e.g., batch normalization, drop out, ensemble methods) and I tried to keep the number of parameters as small as possible. Reducing these constraints could very well bring the model back to SOTA tier. But, they could also interact with the asymptotic layer in unexpected ways. Future research will need to unpack exactly what the interaction is between overall accuracy and robustness for this model.

One major limitation is that, due to a numerical instability in the gradients of the model, the Jacobian matrix is *strictly* zero for the right branch of the learning branch. This means that, once τ reaches its maximum value of 1, there is *no* gradient information. Although this behaviour strongly resembles the intended function of the learning branch—which is supposed to drive the non-dominant rows of the Jacobian matrix towards zero—there was supposed to be some gradient information still present in the model. Gradient masking is a common issue in the literature, so much so that

the living document written by Carlini et al. (2019) outlines a number of techniques for identifying when it is happening. The main concern with gradient masking is that the model might not actually be as robust to the underlying attack as it appears. However, the transfer attacks across all three chapters are suggestive.

The transfer attacks suggest that the asymptotic GNN is more than merely a complex form of gradient masking. For example, the asymptotic GNN only incorrectly classifies 10.2% (968 test samples) and 5.7% (545 test samples) of the CNN's adversarial examples (that both models correctly classified to start with) were incorrectly classified by the asymptotic GNN. In both cases, the average distances values increased. On the MIM attack, for example, the average L_∞ distance for the CNN was 0.171 while for the average distance was 0.220 for the 968 adversarial examples that the asymptotic GNN could not correctly classify. A similar pattern exists for the DFA, but with 1.22 for the average L_2 distance for the CNN and 1.49 for the 545 adversarial examples the asymptotic GNN could not correctly classify. The increased average distances suggest that the model is still more robust beyond the absence of gradient information and this makes sense. True masking only occurs at the end of the critical learning period when τ achieves its maximum value of 1. Throughout the earlier stages of the model's training, the asymptotic GNN is still learning to work with a Jacobian matrix that increasingly approaches zero *while a gradient is still present* from the left branch. That is, there is still an internal dynamics that is unique to the asymptotic GNN and appears to contribute to its robustness. A natural follow up question is what is happening on the learning branch ablation, as the model is still perfectly robust to the MIM attack despite the fact that the learning branch does not get used (i.e., no gradient masking).

More detailed inspection of the learning branch ablation revealed that it suffered from the same issue as the distillation network proposed by Papernot et al. (2016). In effect, the asymptotic barrier function makes the learning gradients too steep, which

confuses the MIM. A simple fix proposed by Carlini and Wagner (2017) is to simply divide the logits by a large number. In my initial research, I was using numbers similar to the authors (e.g., around 100-10,000) to no effect. However, after trying a much more extreme version of this approach—dividing the logits by 10,000,000—I was able to get the MIM to work. Although this does weaken my conclusions about the impact of the asymptotic barrier function, the transfer attacks are suggestive once again. For example, I show in Chapter 4 that the learning branch ablation is the most robust to MIM transfer attacks from the CNN, which still aligns with my hypotheses. For both this limitation and the limitation related to gradient masking, much more research needs to be done to tease apart the complex effects of my asymptotic layer on adversarial robustness.

6.2 Future Work

This research opened far more questions than it closed. These questions can be broken into four main groups: analyses into different model variations of the asymptotic GNN, formal analyses, and evaluating the model on more complex datasets. I discuss the first two of these questions in more detail below.

There are a number of model variations that are natural extensions of the current research. Many of these variations focus on how to expand the scope and impact of the asymptotic barrier function and learning branch. For example, can the three layers of the asymptotic GNN be converted into a stackable block to make deeper networks with multi-level substructural invariances? Can these blocks be arbitrarily incorporated into existing neural network structures to add substructural invariance where it is needed? Similarly, it is still an open question how one can incorporate more complex edge layouts in wider GNN models. Finally, the asymptotic GNN was much more parameter efficient than the non-augmented CNN. It is not clear what

the limits of this trend are or what other dependencies are coupled to the parameter count. These are the main model variations that are directly entailed by this research.

A substantial component of this research provided a detailed formal analysis of the asymptotic barrier function. Consequently, a lot of future work is warranted expanding and deepening these and similar analyses. In particular, it would be ideal if there were formal proofs that the product aggregator and asymptotic barrier function are better than the alternatives. It would also be nice to have a proof that the asymptotic barrier function stabilizes the product aggregator, which is extremely unstable in the general case. The hyperparameter κ deserves a great deal more formal analysis and an inductive proof of my claim that increasing the number of classes (K) does not change the behaviour of the multi-class asymptotic barrier function is definitely achievable.

References

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., ... others (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Bengio, Y., Yao, L., Alain, G., & Vincent, P. (2013). Generalized denoising auto-encoders as generative models. In *Advances in neural information processing systems* (pp. 899–907).

Berardi, N., Pizzorusso, T., & Maffei, L. (2000). Critical periods during sensory development. *Current opinion in neurobiology*, *10*(1), 138–145.

Biederman, I. (1987). Recognition-by-components: a theory of human image understanding. *Psychological review*, *94*(2), 115.

Biggio, B., & Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, *84*, 317–331.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, *59*(4-5), 291–294.

Brainard, M. S., & Knudsen, E. I. (1998). Sensitive periods for visual calibration of the auditory space map in the barn owl optic tectum. *Journal of Neuroscience*, *18*(10), 3929–3942.

Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., ... Kurakin, A. (2019). On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*.

Carlini, N., & Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)* (pp. 39–57).

Chomsky, N. (1959). Review of bf skinner, verbal behavior. *Language*, *35*(1), 26–58.

Chomsky, N. (1980). Rules and representations. *Behavioral and brain sciences*, *3*(1), 1–15.

Cohen, N., Sharir, O., & Shashua, A. (2016). On the expressive power of deep learning: A tensor analysis. In *Conference on learning theory* (pp. 698–728).

Cohen, N., & Shashua, A. (2016). Inductive bias of deep convolutional networks through pooling geometry. *arXiv preprint arXiv:1605.06743*.

Cohen, T. S., Weiler, M., Kicanaoglu, B., & Welling, M. (2019). Gauge equivariant convolutional networks and the icosahedral cnn. In *International conference on machine learning (icml)*.

Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., & Li, J. (2018). Boosting adversarial attacks with momentum. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 9185–9193).

Dosher, B. A., Jeter, P., Liu, J., & Lu, Z.-L. (2013). An integrated reweighting theory of perceptual learning. *Proceedings of the National Academy of Sciences*, *110*(33), 13678–13683.

Dosher, B. A., & Lu, Z.-L. (1998). Perceptual learning reflects external noise filtering and internal noise reduction through channel reweighting. *Proceedings of the National Academy of Sciences*, *95*(23), 13988–13993.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., & Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems* (pp. 2224–2232).

Elsayed, G., Shankar, S., Cheung, B., Papernot, N., Kurakin, A., Goodfellow, I., & Sohl-Dickstein, J. (2018). Adversarial examples that fool both computer vision

and time-limited humans. In *Advances in neural information processing systems* (pp. 3911–3921).

Fawzi, A., Fawzi, O., & Frossard, P. (2018). Analysis of classifiers' robustness to adversarial perturbations. *Machine Learning*, *107*(3), 481–508.

Galloway, J. C., & Thelen, E. (2004). Feet first: object exploration in young infants. *Infant Behavior & Development*, *27*, 107–112.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 ieee international joint conference on neural networks, 2005.* (Vol. 2, pp. 729–734).

Gorodissky, H., Harari, D., & Ullman, S. (2018). Large field and high resolution: Detecting needle in haystack. *Journal of Vision*.

Gu, S., & Rigazio, L. (2015). Towards deep neural network architectures robust to adversarial examples. *ICLR*.

Hassabis, D., Kumaran, D., Summerfield, C., & Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, *95*(2), 245–258.

Hinton, G. E., & Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems* (pp. 3–10).

Hubel, D. H. (1982). Exploration of the primary visual cortex, 1955–78. *Nature*, *299*(5883), 515–524.

Jang, E., Gu, S., & Poole, B. (2017). Categorical reparametrization with gumble-softmax. In *International conference on learning representations 2017*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *International Conference of Representational Learning*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Kumler, J. J., & Bauer, M. L. (2000). Fish-eye lens designs and their relative performance. In *Current developments in lens design and optical systems engineering* (Vol. 4093, pp. 360–370).

Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541–551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2015). Gated graph sequence neural networks. *International Conference On Learning Representations*.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *Proceedings of the international conference on learning representations.*

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, *63*(2), 81.

Moosavi-Dezfooli, S.-M., Fawzi, A., & Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2574–2582).

Orchard, J., & Castricato, L. (2017). Combating adversarial inputs using a predictive-estimator network. In *International conference on neural information processing* (pp. 118–125).

Papernot, N., McDaniel, P., Wu, X., Jha, S., & Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 ieee symposium on security and privacy (sp)* (pp. 582–597).

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, Ł., Shazeer, N., Ku, A., & Tran, D. (2018). Image transformer. *Proceedings of the 35th International Conference on Machine Learning*.

Petrov, A. A., Dosher, B. A., & Lu, Z.-L. (2005). The dynamics of perceptual learning: an incremental reweighting model. *Psychological review*, *112*(4), 715.

Poon, H., & Domingos, P. (2011). Sum-product networks: A new deep architecture. In *2011 ieee international conference on computer vision workshops (iccv workshops)* (pp. 689–690).

Pullum, G. K., & Scholz, B. C. (2002). Empirical assessment of stimulus poverty arguments. *The linguistic review*, *18*(1-2), 9–50.

Rauber, J., Brendel, W., & Bethge, M. (2017). Foolbox v0. 8.0: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80.

Schott, L., Rauber, J., Bethge, M., & Brendel, W. (2019). Towards the first adversarially robust neural network model on mnist. *ICLR*.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., & McDaniel, P. (2017). Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*.

Vapnik, V. (1992). Principles of risk minimization for learning theory. In *Advances in neural information processing systems* (pp. 831–838).

Venetis, J. (2014). An analytic exact form of the unit step function. *Mathematics and Statistics*, *2*(7), 235–237.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103).

Wang, X., Girshick, R., Gupta, A., & He, K. (2018). Non-local neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 7794–7803).

Wu, F., Fan, A., Baevski, A., Dauphin, Y. N., & Auli, M. (2019). Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). Deep sets. In *Advances in neural information processing systems* (pp. 3391–3401).

Zhang, R. (2019). Making convolutional networks shift-invariant again. In *Icml*.

Appendix

Proof Sketch of Γ's Approximation of $H(x, \delta)$

One way to approach a proof that the asymptotic barrier function (Γ) can arbitrarily approximate the unit step function $H(x, \delta)$ is to consider the definite integral of the two functions over the input domain. For $H(x, \delta)$, the definite integral over the domain [0, 1] when $\delta = 0$ is equal to 0. Thus, if Γ has the same start and end points, *and* the definite integral of Γ approaches 0 as ψ increases, then Γ can arbitrarily approximate $H(x, \delta)$ via ψ.

Consider the definite integral $\int_{\frac{1}{e}}^{1} \ln^{\psi}(x)\ dx$.[1] Using integration by parts, one arrives at the following:

$$\Big[x\ln^{\psi}(x)\Big]_{\frac{1}{e}}^{1} - \int_{\frac{1}{e}}^{1} \psi\ln^{(\psi-1)}(x)\ dx$$

This recursive formula keeps expanding until ψ is reduced to 1 in the exponent of ln. There are two cases that need to be considered to complete the definite integral: when $x = 1$ and when $x = \frac{1}{e}$. I consider each in turn below.

For the case when $x = 1$, the recursive formula cancels each of the first terms as $x\ln^{\psi}(1) = 0$ for any ψ. Only the integer constant at the end of the recursive formula contributes to the final result (i.e., when the integral becomes $\int 1\ dx = x$). For each recursive step in the second term, the value of the final term increases by a multiple

[1] I have shifted the input domain from [0, 1] to $[\frac{1}{e}, 1]$ to account for $\kappa = \frac{1}{e}$. I did not increase the upper bound by $\frac{1}{e}$ as the difference in the output between 1 and $1 + \frac{1}{e}$ approaches 0 as ψ increases. It also simplifies the proof sketch.

and the sign changes. For example, the output when $\psi = 3$ is:

$$\begin{aligned} & x\ln^3(x) - 3(x\ln^2(x) - 2(x\ln(x) - x)) \\ & = 0 - 3(0 - 2(0 - 1)) \\ & = (-3)(-2)(-1) \\ & = -6 \end{aligned}$$

Careful inspection of this pattern for different ψ reveals that the pattern reduces to $(-\psi)!$ for all ψ.

For the case when $x = \frac{1}{e}$, the recursive formula also cancels but in a slightly more complex way. One knows that $\ln(\frac{1}{e}) = -1$ and that the power of -1 only changes the sign. Thus, $x\ln^{\psi}(x) = x(-1)^{\psi}$ for any ψ when $x = \frac{1}{e}$. Consider the example when $\psi = 3$ once more:

$$\begin{aligned} & x\ln^3(x) - 3(x\ln^2(x) - 2(x\ln(x) - x)) \\ & = -\frac{1}{e} - 3(\frac{1}{e} - 2(-\frac{1}{e} - \frac{1}{e})) \\ & = 16 \cdot \frac{1}{e} \end{aligned}$$

If one pulls out the $\frac{1}{e}$ and the sign, the recursive formula looks like this:

$$\begin{aligned} & 3 \cdot (2 \cdot (1 \cdot 1 + 1) + 1) + 1 \\ & S_n = n \cdot S_{n-1} + 1 \quad \text{for } n \in \{0\} \cup \mathbb{N} \end{aligned}$$

where $S_0 = 1$. Careful inspection reveals that this can be simplified to:

$$\sum_{k=0}^{\psi} \frac{\psi!}{k!}$$

This then needs to be expanded to include the sign and $\frac{1}{e}$ as follows:

$$\frac{(-1)^{\psi+1}}{e} \cdot \sum_{k=0}^{\psi} \frac{\psi!}{k!}$$

The full definite integral is thus:

$$(-\psi)! - \frac{(-1)^{\psi+1}}{e} \cdot \sum_{k=0}^{\psi} \frac{\psi!}{k!}$$

The sign of the two values in the difference are always opposite, so the definite integral is actually a simple difference between the terms.

$$\psi! - \frac{1}{e} \cdot \sum_{k=0}^{\psi} \frac{\psi!}{k!}$$

This can be simplified to reveal the core difference as follows.

$$\psi!\Big(1 - \frac{1}{e} \cdot \sum_{k=0}^{\psi} \frac{1}{k!}\Big) \sim 1 - \frac{1}{e} \cdot \sum_{k=0}^{\psi} \frac{1}{k!}$$

Since 1 is a constant positive term and $\frac{1}{e} \cdot \sum_{k=0}^{\psi} \frac{1}{k!}$ keeps growing (if by smaller and smaller amounts), it is trivial to see that the difference approaches 0 as ψ approaches infinity. This completes my proof sketch that Γ arbitrarily approaches $H(x, \delta)$ as ψ approaches infinity.

Implementation Details

Baseline CNN

The baseline CNN is made up of four convolutional layers. Their kernel sizes are 5, 4, 3, and 5. Their stride sizes are set to 1, 2, 2, 1. Their filter sizes are set to 20, 70, 256, and 10. All of the layers have an exponential linear unit (ELU) non-linearity, and every layer except the last is followed by a batch normalization layer with the default PyTorch parameters.

Asymptotic GNN

The first "encoder" encoder (ϕ_α) has three layers with the following sizes and types: 64 (input), 1024 (hidden) and 8 (output). The decoder (ϕ_α^{-1}) has the opposite configuration. The hidden units always use rectified linear units and the output units always use the hyperbolic tangent.

The second network (ϕ_β) also has three layers but with different sizes and types: 8 (input), 1024 (hidden) and 10 (output). The hidden layer has a rectified linear activation after it, but the output does not.

All of the weights are initialized with the Glorot/Xavier normal distribution (Glorot & Bengio, 2010).

Training Details

All of the models used a fixed seed for random initialization to decrease the impact of the parameter initialization.

Baseline CNN

The baseline CNN was trained for 5 epochs (i.e., full iterations through the entire training set of 50,000 images) with a learning rate of 0.001, using an Adam optimizer with the PyTorch default values. I used the standard softmax cross entropy loss.

Asymptotic GNN

The encoder (ϕ_α) and decoder networks (ϕ_α^{-1}) were trained on randomly sampled 8x8 image patches from the training set of MNIST images. It was always trained for 20 epochs. The mean square error reconstruction loss was always used. The learning rate was set to 0.0001 and Adam was always used with the default PyTorch parameters.

The second network (ϕ_β) was trained on the full images. It was always trained after the encoder's weights were frozen. It was always trained for 20 epochs except for the ablation models. There it was trained for 100 epochs (learning branch ablation) and 120 epochs (asymptotic barrier function ablation). The cross-entropy loss was always used. The learning rate was always set to 0.001 and Adam was always used with the default PyTorch parameters.

Hyperparameters

Hyperparameter	Model	Value
learning rate	ϕ_α	0.0001
	ϕ_α^{-1}	0.0001
	ϕ_β	0.001
training epochs	CNN	5
	autoencoder, denoising	20
	GNN	20
	asymptotic barrier function ablation	120
	learning branch ablation	100
batch size	ϕ_α	512
	ϕ_α^{-1}	512
	ϕ_β	128
hidden layer	all	1024
latent layer	all	8

Table 7: List of Hyperparameters