ADVANCING AND LEVERAGING TRACTABLE LIKELIHOOD MODELS

Christopher M. Bender

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2022

Approved by:

Junier B. Oliva

Marc Niethammer

Martin Styner

Michael K. Reiter

Manzil Zaheer

ABSTRACT

Christopher M. Bender: Advancing and Leveraging Tractable Likelihood Models
(Under the direction of Junier B. Oliva)

The past decade has seen a remarkable improvement in a variety of machine learning applications thanks to numerous advances in deep neural networks (DNN). These models are now the de facto standard in fields ranging from image/speech recognition to driverless cars and have begun to permeate aspects of modern science and everyday life.

The deep learning revolution has also resulted in highly effective generative models such as score matching models [157], diffusion models [81], VAEs [92], GANs [63], and tractable likelihood models [47, 67, 132]. These models are best known for their ability to create novel samples of impressive quality but are usually limited to highly structured data modalities. *Expanding the capabilities and applications of likelihood models beyond conventional data formats and generative applications can increase functionality, interpretability, and intuition compared to conventional methods.*

This dissertation addresses shortcomings in likelihood models over less structured data and explores methods to exploit a learned density as part of a larger application. We begin by advancing the performance of likelihood models outside the standard, ordered data regime by developing methods that are applicable to *sets*, e.g., point clouds. Many data sources contain instances that are a collection of unordered points, such as points on the surface of scans from human organs, sets of images from a web page, or LiDAR observations commonly used in driverless cars or (hyper-spectral) aerial surveys. We then explore several applications of density models. First, we consider generative process over neural networks themselves and show that training over ensembles of these sampled models can lead to improved robustness to adversarial attacks. Next, we demonstrate how to use the transformative portion of a normalizing flow as a feature extractor

in conjunction with a downstream task to estimate expectations over model performance in local and global regions. Finally, we propose a learnable, continuous parameterization of mixture models directly on the input space to improve model interpretability while simultaneously allowing for arbitrary marginalization or conditioning without the need to train new models or develop complex masking mechanisms.

The past decade has seen a remarkable improvement in a variety of machine learning applications thanks to numerous advances in deep neural networks (DNN). These models are now the de facto standard in fields ranging from image/speech recognition to driverless cars and have begun to permeate aspects of modern science and everyday life.

The deep learning revolution has also resulted in highly effective generative models such as score matching models, diffusion models, VAEs, GANs, and tractable likelihood models. These models are best known for their ability to create novel samples of impressive quality but are usually limited to highly structured data modalities. Expanding the capabilities and applications of likelihood models beyond conventional data formats and generative applications can increase functionality, interpretability, and intuition compared to conventional methods.

This dissertation addresses shortcomings in likelihood models over less structured data and explores methods to exploit a learned density as part of a larger application. We begin by advancing the performance of likelihood models outside the standard, ordered data regime by developing methods that are applicable to sets, e.g., point clouds. Many data sources contain instances that are a collection of unordered points, such as points on the surface of scans from human organs, sets of images from a web page, or LiDAR observations commonly used in driverless cars or (hyper-spectral) aerial surveys. We then explore several applications of density models. First, we consider generative process over neural networks themselves and show that training over ensembles of these sampled models can lead to improved robustness to adversarial attacks. Next, we demonstrate how to use the transformative portion of a normalizing flow as a feature extractor in conjunction with a downstream task to estimate expectations over model performance in local and global regions. Finally, we propose a learnable, continuous parameterization of mixture models directly on the input space to improve model interpretability while simultaneously allowing for arbitrary marginalization or conditioning without the need to train new models or develop complex masking mechanisms.

"To my wife, whose support never fails, and to the rest of my family, without their insanity, I would surely have gone mad.".

ACKNOWLEDGEMENTS

When I decided to return to academia to pursue a PhD, I expected the process would be something of a battle, but one that I was experienced and practiced enough to take with some time and effort. Due to some combination of an over-estimation of my own abilities, an under-estimation of the extent of the task, and complications from a global pandemic, the undertaking was more a war than a battle. In the end, I have only persevered through the extensive advice, support, and help from family, friends, and colleagues. If not for them, I would still be mired in the mud. I don't really have the words to express my gratitude but please accept this as the best I can offer.

My wife, Sarah, easily deserves the most thanks. She has had complete faith (both times) when I uprooted us from our family and community to move hundreds (or thousands) of miles to pursue my goals. She has helped and carried me through the stress, frustrations, and sleepless nights. I want to say that she walked with me when times were good and carried me when things were hard, but the analogy is a bit used and it feels a bit sacrilegious... so instead I'll just borrow another cliche and say, I couldn't (and wouldn't have wanted to) have done it without her. It hasn't always been fun and easy, but her (and later our son's) presence has made life worth living.

My family (both by blood and law) have contributed heavily to my successes. This goes all the way back to when my parents and siblings encouraged and enabled me to participate in clubs and extracurricular activities and has continued well into adulthood and across time zones. I am extremely fortunate to have married into another family that is equally supportive, even when I took their daughter/sister and grandchild/nephew further away than they had ever hoped. I need to thank my brother-in-law, Wesley, and sister, Ashley, for putting their lives on hold and helping to take care of my infant son early in the pandemic when a vaccine was unavailable and daycares

were closed. Without their help and sacrifice, we would have been less safe and this process would have taken much longer.

My many friends and colleagues at The Johns Hopkins University Applied Physics Laboratory have been instrumental in my successes as an engineer and researcher. Without them I would not have the experience and diligence necessary to participate in today's fast-paced world. In particular, I would like to thank Dr. Shane Lani, Dr. CJ Della Porta, and their families for their unceasing friendship and support through the PhD. I am very lucky to have encountered such good friends and examples. Along these lines, I would like to call out Patrick Emmanuel for letting me bounce ideas off him and helping me troubleshoot issues whenever I found myself stuck. He is probably the most impressive hands-on-keyboard ML practitioner that I know and our many discussions have been very helpful in diversifying my knowledge and understanding.

Finally I would like to thank the many people here at UNC that have helped me along the way starting with my advisor, Dr. Junier B. Oliva, for all of his support and guidance. Most especially in the first two years as I transitioned from more of an engineer into a computer and machine learning scientist. His advice (and patience) were essential to this process and to my growth as a researcher and scientist. In this vein, the constructive feedback and collaboration from members of the the Lupa lab was very helpful and instructive. Similarly, the members of my cohort, notably Yifeng "Jack" Shi, provided a much-needed outlet and sympathetic ear. I would also like to thank the various members of my committee for their patience, comments, and advice. This work was supported in part by NSF grant IIS2133595 and NIH grant 1R01AA02687901A1.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND INITIALIZATIONS

AI	Artificial Intelligence
AR	Autoregressive model
BNN	Bayesian Neural Network
CNF	Continuous Normalizing Flow
CNN	Convolutional Neural Network
DL	Deep Learning
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
MFA	Mixture of Factor Analyzers
ML	Machine Learning
MLP	Multi-Layer Perceptron
NICE	Non-linear independent components estimation
NF	Normalizing Flow
NN	Neural Network
NODE	Neural Ordinary Differential Equation
ODE	Ordinary Differential Equation
RNVP	Real Non-Volume Preserving
VAE	Variational Autoencoder

LIST OF SYMBOLS

N	Number of examples
M	Number of dimensions
R	Rank
$f(x)$	Generic functions or neural networks
$h(x)$	Bijective functions or normalizing flows
x	Input data
y	Labels
z	Latent/Transformed data
μ	Mean
σ	Standard deviation
$\sigma(x)$	Sigmoid or Soft-max function
θ	Network parameters
$\mathbb{\Sigma}$	Covariance Matrix

CHAPTER 1: INTRODUCTION

Machine learning has exhibited incredible advances in recent years, achieving human (and even super-human) performance in applications ranging from image recognition [102], object detection [144], machine translation [127], drug discovery [91], augmented reality [60], *etc.*, through the use of large, deep neural networks. These improvements have revolutionized the way we interact with technology in our daily lives [**?** 69] and have begun to change the way we learn [133], explore, and process the world using computational mathematics [56] and advanced sciences [91, 148]. The deep learning revolution has also resulted in highly-effective generative models such as VAEs [92], GANs [63], energy machines [129], score matching models [157], diffusion models [81], and tractable likelihood models [47, 67, 132]. These models are most famously used to generate novel samples of impressive quality. Despite the increasing number and effectiveness of these models, they are still largely utilized primarily for generative purposes in and of themselves and are trained over fairly simple, highly-structured data modalities. Likelihood models in particular provide access to a principled estimate of a dataset's underlying probability density function. This function should enable a huge number of alternative processes, *e.g.*, likelihood ratio tests, but these possibilities have largely been ignored or have proven surprisingly brittle. Similarly, many real-world observations are better represented using unordered collections, namely sets, without resorting to presuming some predefined, and possibly arbitrary, order. Existing methods to construct likelihood models over exchangeable (orderless) data largely rely on *i.i.d.* assumptions that can over-simplify or ignore the complex inter-dependencies between points within a given set.

1.1 Learning Over Sets

Modeling unordered, non-*i.i.d.* data is an important problem in machine learning and data science. Collections of data objects with complicated intrinsic relationships are ubiquitous. These collections include sets of 3d points sampled from the surface of complicated shapes like human organs, sets of images shared within the same web page, or point cloud LiDAR data observed by driverless cars. In any of these cases, the collections of data objects do not possess any inherent ordering of their elements. Thus, any generative model which takes these data as input should not depend on the order in which the elements are presented *and* must be flexible enough to capture the dependencies between co-occurring elements. As a simple example, one may trivially generate a set of exchangeable points by drawing them *i.i.d.* from some distribution. More commonly, however, elements within an exchangeable set share information with one another, providing structure and inter-element dependencies. Despite the abundance of such data, the bulk of existing approaches either ignore the relation between points (*i.i.d.* methods) or model dependencies in a manner that depends on inherent orderings (sequential methods) [145, 183]. In order to accurately learn the structure of a set whilst preserving the exchangeability of its likelihood, one cannot rely solely on either approach. In this dissertation, we propose a tractable, non-*i.i.d.* density estimator for exchangeable sets that is suitable for a variety of cardinalities (number of points) and dimensionalities. Our method utilizes the composition of an invertible, permutation equivariant transform, a sorted scan, and a conventional, sequential density model such as an autoregressive methods. The utilization of the sorted scan is the first method that enables the application of fixed-order density models to exchangeable data in a principled way without relying on impractical averaging over permutations.

1.2 Applications of Likelihood models

Despite their impressive performance and successes, neural networks suffer from several shortcomings such as brittle out-of-distribution behavior [80] where models fail to recognize that a sample does not belong to the same distribution as the data from the training corpus; adversarial weakness [64] which allows a bad agent to influence the decision that a neural network makes for nefarious reasons; and a complete lack of explainability [120] which makes it difficult to understand why a network produces any given output or decision. These failures erode trust in trained agents and severely limit the deployment of ML solutions in environments related to safety and security. This is particularly relevant to the large-scale industrial push into driverless cars [10] and automated medicine [5]. How can we put our lives in the hands of an agent that we don't understand, cannot make any theoretical guarantees about their performance, and might behave erratically due to a unexpected (but otherwise minor) situation? We propose that a good estimator of the underlying data distribution can address many of these shortcomings.

The adversarial problem is the tendency of performant machine learning algorithms to fail under the addition of specially crafted perturbations. Generally, these perturbations are sufficiently small as to be unobservable to a human agent. Worse, a bad-actor is often capable of constructing attacks that do not just render the machine learning model incorrect or uncertain but the model will produce a specific (wrong) output at extremely high confidence levels. For example, you could imagine a hacker trivially bypassing the facial recognition locks on your phone or computer without any difficulty or raising any red flags. Researchers have even demonstrated that specially-designed stickers can be added to street signs that can trick a driverless car into thinking that a stop-sign is a 75 mile-per-hour sign with potentially devastating consequences [54]. A variety of hardening methods exist. We explore the utility of constructing a distribution over neural networks as a form of defense. Specifically, we train a Bayesian neural network (BNN) on classification tasks. This results in sampling a new neural network from the trained, posterior distribution over networks on each model execution. Unlike most BNNs, we train and evaluate

each example over an ensemble of randomly drawn networks, where we have encouraged output diversity across network draws to reduce the chance of ensemble degeneration.

While the BNN is reasonably successful at inducing adversarial resilience, it accomplishes its goals somewhat indirectly. What we would really like the network to do, is to maintain local consistency within some hypervolume around the training data so that no perturbation within that volume leads to bad results. Unfortunately, neural networks operate inherently on single examples, so even just evaluating a network's average performance or variation within some high-dimensional region is completely impractical. In fact, the problem of estimating high-dimensional integrals is difficult in general, let alone with all the complexities inherent in a neural network. However, there are special cases where these complex integrals become tractable. We develop one of these cases, where the neural network is composed of separable functions in conjunction with a bijective feature extractor. This combination allows to perform tractable integration in a latent space that is matched to a relevant integral in the input space. We utilize this idea to improve out-of-distribution performance in the global setting and somewhat improved adversarial performance in the local setting.

Finally, while modern machine learning algorithms achieve incredible performance on a variety of tasks, it is difficult to impossible to explain why a machine made a particular decision. A variety of methods exist to heuristically explain these decisions [**?**] based on gradient propogations and activations back to individual input features on any given example. However, these methods don't provide any description on how much expected variation might exist for any feature or how robust the model is to said variation. Simpler models, on the other hand, are often interpretable but generally have notably inferior performance. We propose to combine the complexities of modern ML models with the explainability and simplicity of mixture models by parameterizing a mixture model using a learnable, neural ordinary differential equation. This model provides an exact likelihood estimate directly over the input features. We combine several different continuously parameterized mixtures together in a heirarchical form to enable classification and clustering through Bayes rule. This combination allows us to perform interpretable

clustering (or classification) since we can isolate specific components for each decision and explain how a component was chosen based on the component mean and covariance. Unfortunately, training these models can be extremely difficult and the models tend to be very reliant on initialization. We offset this difficulty by creating a curriculum over data representations from a simple, known distribution to the more complex data distribution.

In conclusion, this dissertation demonstrates a variety of methods to exploit generative models to offset many of the shortcomings of modern machine learning by constructing flexible densities over sets, exploiting a generative process to jointly train ensembles of networks, relate integrals in the input space to a latent space, and by constructing a strong density estimator directly over the input space.

1.3 Thesis Organization

This dissertation is organized into the following chapters:

- **Chapter 2: Background** We provide a brief overview of relevant topics utilized in this dissertation from the open literature.

- **Chapter 3: FlowScans** We design a novel approach to density estimation of exchangeable, non-*i.i.d.* data which combines invertible transformations with a sorted scan to model data. FlowScans are the only method that creates a link between invertible, sequential transformations and exchangeable data without relying on naive averaging over permutations.

- **Chapter 4: Defense Through Diverse Directions** We augment the training process of Bayesian neural networks to require that the model maintain some expected uncertainty over all input features. We show that this augmentation naturally increases the adversarial resilience of the network.

- **Chapter 5: Practical Integration** In this chapter we construct a hybrid network as the composition of the transformative portion of a normalizing flow followed by a learned, separable function. This composition enables us to create tractable estimates of the expected performance of the total network by simplifying the high-dimensional integral over the input space in a collection of one-dimensional integrals.

- **Chapter 6: Continuously Parameterized Mixture Models** We parameterize mixture models using a learnable, ordinary differential equation. This parameterization allows us to construct a robust density estimator over the input space and allows for increased interpretability.

- **Chapter 7: Discussion and Future Work** We conclude with a general discussion of the contributions of the dissertation and possible future directions.

1.4 Contributions

This dissertation makes advances in the following ways:

1. We introduce a scanning-based method for modeling exchangeable data in a manner that relates the exchange likelihood to that of the sorted features. This connection enables us to utilize traditional density estimators without resorting to simple *i.i.d.* assumptions or naive averages over permutations.

2. We demonstrate that transforming points with an invertible, permutation-equivariant change of variables allows for modeling sets in an alternative space. This transformation provides the model with the flexibility to identify the appropriate scanning representation without relying on arbitrary, a priori decisions.

3. We propose several general penalties to augment the training of a Bayesian neural network and diversify the output variation relative to the input features.

4. We show that increasing output diversity leads to natural adversarial robustness without necessarily requiring online adversarial training and that this resilience generalizes to a variety of attack schemes.

5. We propose a general architecture that enables tractable integration over the high-dimensional input space by combining a bijective feature extractor with a separable, learnable function. This allows us to introduce supervision and regularization over hyper volumes.

6. We develop penalties over classification decisions in global and local regions to encourage consistency and offset shortcomings in out-of-distribution decisions. This application illustrates a method to overcome complications from the combination of a separable function and a constrained output vector by optimizing a bound to the desired objective in place of the objective itself.

7. We propose a parameterization of mixture models through the output a neural ordinary differential equation. This formulation induces a smoothly-varying trajectory through the data, learning a probabilistic sheath across the data manifold.

8. We demonstrate a curriculum-based training method to significantly improve model performance by offsetting difficulties from bad initializations. The curriculum allows us to define an annealing scheme from a simple, well-known distribution to the more-complex data manifold.

CHAPTER 2: BACKGROUND

In this chapter we briefly review several topics that are useful for using learnable, likelihood models and will be used throughout this dissertation.

2.1 Neural Networks

Neural networks (NNs) are at the heart of most modern machine learning algorithms. These algorithms are constructed as compositions of many different parameterized functions such that

$$f(x) = f_n \circ f_{n-1} \circ \ldots \circ f_1(x) \tag{2.1}$$

and each f_i is parameterized by some *learnable* variables, θ. The parameters of the network are updated through stochastic gradient descent and its various derivatives [93] by evaluating the network performance on mini-batches of a training set against some loss function. The training process introduces some light requirements on the choices of f_i, namely that the operations must be differentiable. The choice of the loss function depends on the goal of the network and the availability of target predictions. Common choices are cross-entropies for classification problems and mean squared errors for regression problems. However, both of these tasks are supervised and require matched outputs for each training input. In this dissertation, we will consider compositions of supervised tasks and unsupervised tasks where the unsupervised tasks are governed by a generative process and typically use maximum likelihood estimation.

2.1.1 Convolutional Neural Networks

A huge number of different architectures exist based on different choices of each f_i. One of the oldest constructions, known as the multi-layer perceptron (MLP), intersperses linear operations with nonlinear, element-wise activation functions. The linear operator allows the network to learn cross-feature dependencies while the activation function determine the importance of (hidden) representation. However, while the MLP is extremely flexible they can be difficult to train (and often exhibit difficulties in generalizing to held-out data)[149]. Some of these difficulties caused many researchers to despair that neural networks were capable of realizing their theoretical capacity. It was not until the advent of large-scale data, compute, and the introduction of *convolutional neural networks* (CNNs) that neural networks became mainstream and started to produce the super-human performance on spatial-temporal data they are now famous for.

The effectiveness of CNNs is typically attributed to their inductive bias for image and image-like data. This essentially means that convolutions are well-suited to processing gridded data. Convolutions are a subset of more general linear operators and are defined as a translating inner product between some kernel or filter, $k[n]$, and the data, $x[n]$, *e.g.*, in the 1D case,

$$z[n] = k[n] * x[n] = \sum_j k[j]x[n-j] \tag{2.2}$$

where we have explicitly included the grid index, n. The convolutional layer used in CNNs simultaneously performs convolutions over different input channels, c, before summing over channels. This convolve-and-sum operation is applied some pre-determined number of times using different filters to construct a variety of output channels, i,

$$z_i[n] = \sum_c k_{c,i}[n] * x_c[n] = \sum_c \sum_j k_c[j]x_c[n-j]. \tag{2.3}$$

CNNs are trained using gradient descent where the taps in the filters are the learnable variables and the number of taps (size of the kernel) is chosen during network construction. We will make

extensive use of CNNs in this work, both for constructing likelihood models and as part of other task networks.

2.1.2 Neural Ordinary Differential Equations

Most neural network architectures are defined explicitly, one layer at a time. Recently, new methods have emerged that allow us to define our architectures *implicitly*. In particular, we are interested in defining networks as the solution to an ordinary differential equation

$$\frac{df(s)}{ds} = g\left(f(s), s; \theta\right), \quad f(s_0) = f_{s_0}\,, \tag{2.4}$$

where we have explicitly defined the learnable function, g, but the network is still represented by the function, f, which must be solved for and cannot, in general, be represented in closed-form. Chen et al. proposed a computationally and memory efficient method for solving these ODEs using the adjoint method and any black-box solver to compute

$$f(s_{i+1}) = f(s_i) + \int_{s_i}^{s_{i+1}} g\left(f(s), s; \theta\right) ds\,. \tag{2.5}$$

across many time steps. Prior to their work, similar attempts maintained gradients through the iterative solver and were prohibitively expensive [115]. Alternatively, solving this equation using a single step with the Euler discretization leads to the famous Residual Networks [75].

In theory, it is possible to choose any network architecture and activation functions for g, however, in practice, some choices can cause the learned ODE to become increasingly stiff (numerically unstable) [20], resulting in an ever-decreasing step size and correspondingly increased computational cost or integration failures. In this dissertation, for tabular data, we set the architecture of g to be a multi-layer perceptron with an explicit dependence on t and use either $\tanh$ or $\sin$ as the activation function. We find that when using $\sin$, the ODE seems to be less prone to becoming stiff and is easier to numerically integrate. When using image data, we utilize a convolutional neural network with the same activation functions.

We utilize Neural ODEs for two purposes.

1. As the transformative operation within a normalizing flow, see Sec. 2.2.2.4 and 6.3.3.

2. To parameterize a Gaussian mixture model, see Chap. 6.

2.2 Likelihood Models

In this section we explore several different types of likelihood models.

2.2.1 Gaussian Mixture Models

Traditional (finite) mixture models model the presence of sub-populations within a dataset through a convex combination of distinct components of a chosen distribution. Gaussian mixture models (GMMs), the most common class of mixture models, restrain the set of chosen distributions that constitutes the mixture to the Gaussian family:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\mathbf{x}; \mu_k, \mathfrak{Sigma}_k) \,, \tag{2.6}$$

where $\mathbf{x} \in \mathbb{R}^M$ is the input data, π is the component weight, $\mu \in \mathbb{R}^M$ is the mean, $\mathfrak{Sigma} \in \mathbb{R}^{M \times M}$ is the covariance matrix, and K is the number of components in the mixture. Despite the seemingly stringent simplification GMMs make, they encompass a broad set of distributions. In effect, they are universal approximators to any smooth probability distribution given enough mixture components [65]. This offers us an insight when deploying GMMs for data modeling: the more complex (and high-dimensional) the data distribution is, the more components the GMM would likely require to yield a reasonable approximation. Motivated by this insight, analogous to defining an integral as an infinite sum, we take the limit $K \to \infty$ in Eq. 2.6 to arrive at the following continuous representation of GMMs

$$p(\mathbf{x}) = (2\pi)^{-M/2} \int_0^1 |\mathfrak{Sigma}(s)|^{-0.5} \exp\left(- \left(\mathbf{x} - \mu(s)\right)^T \mathfrak{Sigma}^{-1}(s) \left(\mathbf{x} - \mu(s)\right) / 2 \right) \pi(s) \, ds \tag{2.7}$$

where the set of parameters, $\{\pi_k, \mu_k, \mathfrak{Sigma}_k\}_k$, for the finite GMMs become the set of functions, $\{\pi(s), \mu(s), \mathfrak{Sigma}(s)\}$, that is parametrized by the variable s whose meaning depends on the specific context of the problem. Eq. 2.6 can then be reversely seen as a discretized formulation of Eq. 2.7.

2.2.2 Normalizing Flows and Bijective Networks

Bijective networks are the key component in flow-based likelihood models. A bijective network, $h : \mathcal{D} \to \mathcal{Z}$, has a known forward and inverse operation so that data can be exactly reconstructed (inverted) after the transformation. This allows for exact likelihood estimation via the change of variables formula:

$$\mathbf{z} = h(\mathbf{x};\ \theta), \quad \mathbf{x} = h^{-1}(\mathbf{z};\ \theta), \quad \log p_X(\mathbf{x}) = \log p_Z(h(\mathbf{x};\ \theta)) + \log \left| \frac{\partial h}{\partial \mathbf{x}} \right| \tag{2.8}$$

where p_Z is a predefined distribution over the latent space, often a standard Gaussian.

These models are trained via Eq. 2.8 to maximize the likelihood of the examples in the training set, $\mathcal{T}$. Once trained, flow-based likelihood models are commonly used as a generative process where samples are drawn from p_Z and are then inverted through h to arrive at an example in $\mathcal{D}$.

The requirement for bijectivity places a strong constraint on network design, eliminating many common choices due to the need to maintain dimension or invert element-wise activations. Even naive convolutional operations become unavailable since they are not generally invertible. Modern advances have demonstrated methods to work around these limitations through the use of clever partitioning and coupling tricks [47] or the use of constraints [30]. However, the field of learnable bijective functions is less advanced than its injective counterparts which results in reduced performance on auxiliary tasks. We briefly discuss a few common learnable, bijective functions that are used in this dissertation.

2.2.2.1 Linear and Element-wise Transformations

Possibly the most obvious choices for invertible transformations is to borrow the form of the common multilayer perceptron where we compose linear operators with nonlinear (element-wise) activation functions. Unlike standard MLPs we must place additional constraints on both the linear maps and the nonlinear functions.

Linear Transformations: If we define the linear map as

$$z = Ax + b \tag{2.9}$$

with $x \in \mathbb{R}^M$ then $A \in \mathbb{R}^{M \times M}$ and $\mathrm{rank}(A) = M$ such that the inverse of A exists. This simple transformation is extremely useful and appears in some form in most modern flow architectures due to its ability to reorganize the data prior to other operations.

Also unlike their use in an MLP, we require an estimate of the determinant of the Jacobian for this operation. This is trivial during inference, where the determinant of the A can be pre-calculated and cached, but can result in an $\mathcal{O}(M^3)$ cost for each batch when implemented naively. Similarly, when we utilize a normalizing flow for sampling, we would require an $\mathcal{O}(M^3)$ cost for inversion. There are two common tricks to offset this cost, both of which amount to reparameterizing A. The most popular method is to utilize the LU decomposition, $A = LU$, and learn both triangular matrices L and U directly instead of A [132]. This allows us to calculate the log-determinant of A as $\log |A| = \sum_m \log U_{m,m}$ with $\mathcal{O}(M)$ cost. Similarly, the inverse can be calculated in $\mathcal{O}(M^2)$ by solving two triangular matrix equations.

Element-wise Transformations: Like the linear transform, the space of available element-wise nonlinear functions is somewhat restricted when constructing a normalizing flow. For our purposes, it is sufficient that the function produces unique outputs for unique inputs and is (at least) $\mathcal{C}^1$ continuous. This immediately eliminates some common activation functions used in other ML applications, most notably, the rectified linear unit (ReLU) is not allowed since it maps all non-positive inputs to zero. Leaky-ReLU is allowable and is a simple substitute. Leaky-ReLU is

defined as

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{2.10}$$

and $0 < \alpha < 1$. Other common activation functions such as sigmoid, the hyperbolic tangent, or soft-plus are also permissible activations. However, in practice, functions whose codomain does not span $\mathbb{R}$ can be problematic when inverting the network, especially early in the training process. For example, if sigmoid is chosen as the activation function, the values passed in for inversion *must* be between zero and one or the inversion will be undefined. If the preceding process does not ensure this range, it will result in bad (NaN) samples and training collapse. A similar issue occurs as the values approach zero or one due to numerical errors.

2.2.2.2 Coupling Transformations

The familiar composition of linear operations and nonlinear activations requires either $\mathcal{O}(M^3)$ cost for determinant and inverse calculations or limited parameterizations via triangular decompositions and a reduced set of activations functions, see Sec. 2.2.2.1 [45]. This limitation has led to the development of alternative architectures that allow for more complex dependencies. Probably the most common architecture developed specifically for normalizing flows revolves around the idea of cross-dimension "coupling."

This method partitions the data into two groups. The first group undergoes some transformation *conditioned* on the second group while the second group remains untransformed. The second group is left untransformed to ease the inversion process but be modified in a downstream operation. For example, with tabular data the first half of the data, $x_1 \in \mathbb{R}^K$ undergoes an invertible transformation conditioned on the second half, $x_2 \in \mathbb{R}^{(M-K)}$, such that $z_1 = f(x_1 \mid x_2)$, $z_2 = x_2$, and $z = [z_1, z_2]$, see Fig. 2.1 for an illustration. This idea was originally proposed by Dinh et al. as NICE with $f(x_1 \mid x_2) = x_1 + m(x_2; \theta)$ where $m(x_2; \theta)$ is a learnable, nonlinear function, *e.g.*, a neural network, parameterized by θ. Since $m(x_2)$ is not used to get z_2, it does not have the same requirements that the other flow operations have, most especially, there is no need for m to be

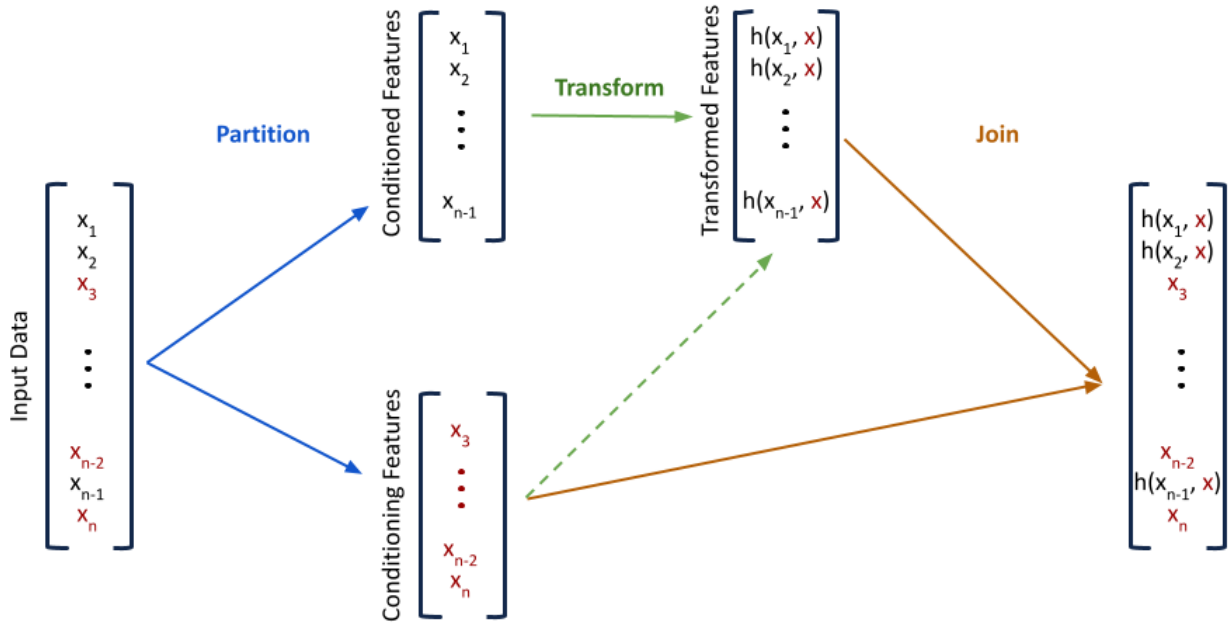


Figure 2.1: Example of general coupling for tabular data. Data is split into conditioned (black) and conditioning (red) partitions. The conditioned data is then invertibly transformed via a conditional operation given the conditioning information. Conditioning information is passed through unaltered.

invertible which means that we can use any architecture we want when constructing m. This relatively simple form performs surprisingly well and has the advantage that it is volume preserving so the determinant is identity and the inverse only requires executing the network, m, on z_2 and subtracting the result from z_1, *e.g.*, $x_1 = z_1 - m(z_2)$.

Real-NVP [47] extends NICE by modifying $f(x_1 \mid x_2)$ to include an element-wise multiplication: $f(x_1 \mid x_2) = \exp\left(a(x_2); \phi)\right) \odot x_1 + m(x_2; \theta)$ where $a(x_2; \phi)$ is a learnable function and the $\exp$ is applied to guarantee non-degenerate solutions. It is important to note that while m and s operate linearly with respect to the first partition, they behave nonlinearly relative to the second partition. Inversion is only slightly more complex than for NICE and proceeds in a similar manner, *e.g.*, $x_1 = \frac{z_1 - m(z_2)}{a(z_2)}$. The log-determinant of the Jacobian is still identity for the second group but must account for the rescaling of the first group and is given by $\log \left| \frac{\partial z_1}{\partial x_1} \right| = \sum_k s_k$. The general idea of coupling has been further extended in a variety of ways. The current state-of-the-art coupling methods construct f as a spline operator (of fixed order) whose knots are derived

from the second group [50, 123]. We refer the interested reader to the associated references for additional information. For the sake of simplicity and expediency, the work in this dissertation limits itself to Real-NVP coupling transformations unless otherwise stated.

An obvious difficulty when formulating coupling transformations is how to partition the dimensions. Arbitrarily partitioning the data into the first K features and then the rest of the features can introduce some strong biases into how transformations are conditioned and are likely to miss the appropriate combinations. A convenient way to offset this difficulty is to intersperse linear transformations between coupling transformations. At full flexibility, linear transformations can learn any ordering or the features to create optimal partitions for each stage. In this case, we can essentially consider that our architecture is akin to an MLP except that we have replaced the element-wise activations with nonlinear, conditioning functions. Given the added flexibility of the linear layers, it is more reasonable to arbitrarily partition the data, though the choice of K is still arbitrary. A flexible alternative to the bifurcation of the data as discussed so far is to create a separate partition for each dimension and construct the conditioning networks in such a way that *all previous* dimensions are used in an autoregressive form. This does notably improve performance by allowing more complex functions at later dimensions, it is considerably more expensive and time consuming than simpler splitting schemes [50].

For tabular data, we utilize arbitrary splits of the data into front and latter partitions and rely on previous, fully-connected layers to organize the features into conditioned and conditioning partitions. For higher-dimensional grids, like images, a simple scheme might take left/right or top/bottom splits or perhaps split by channels. Dinh et al. [47] proposed to extract pixels in an even/odd form using a chess board pattern and kept all channels for each pixel together as in Fig. 2.2. They argue that this split allows for better consistency of both global and local information when constructing the conditioning transforms and they demonstrate better performance empirically over other splits.

Practical Considerations It can be helpful to consider that the conditioning functions are a type of hypernetwork [72], where we derive the parameters of a transformation from the data using

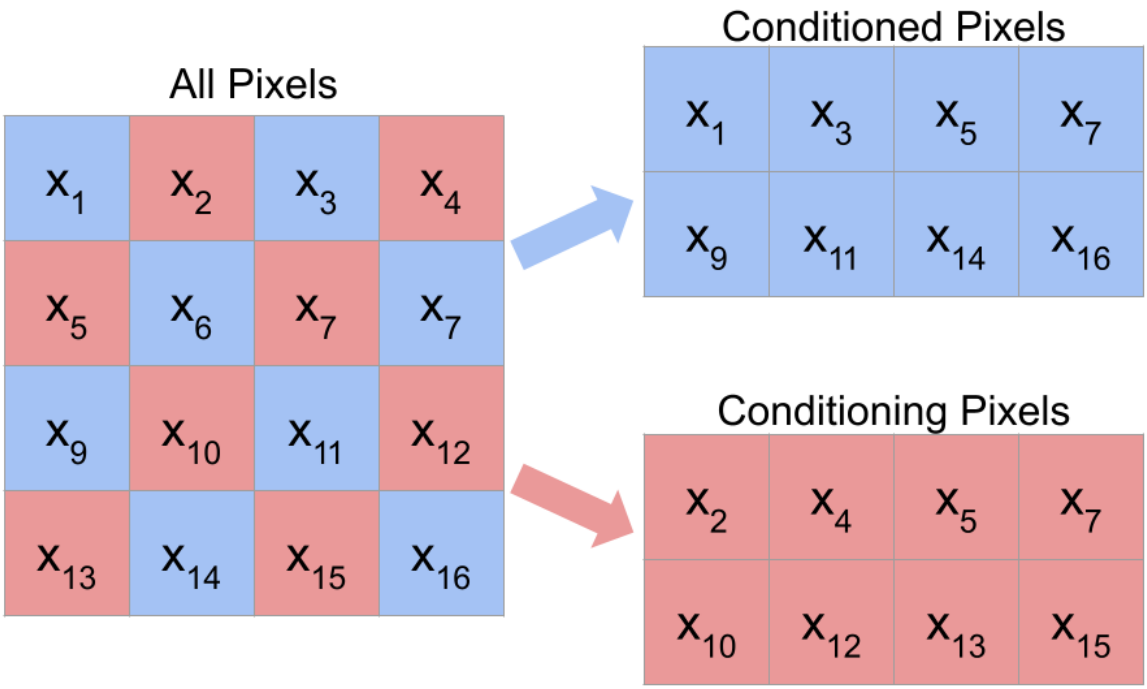


Figure 2.2: Checkerboard coupling strategy proposed by Dinh et al. [47] to maintain spatial correlations between conditioned and conditioning information across an image.

a neural network. These conditioning networks are often fairly large and can be very resource intensive, especially with respect to GPU RAM during training for backpropagation. In practice, we can reduce the memory footprint by wrapping each conditioning network with a gradient checkpointing operation [32]. This trick does not save the internal gradients during the initial forward pass of each conditioning network. Instead, we re-perform the forward operation of each hypernetwork during the backwards pass and save and apply the gradients then. When many coupling operations are utilized in one flow, we can significantly decrease our maximum RAM requirements. In our experiments, we often reduced RAM usage by nearly an order of magnitude and only incurred a 10-20% increase in training time, allowing us to train larger models or utilize larger (higher-dimensional) data.

2.2.2.3 Invertible 1×1 Convolutions

The primary instigator of the deep learning rush was the advent of convolutional neural networks (CNNs). Ideally, we would like to make use of this strong inductive biases (when appro-

priate) when constructing our normalizing flow models. Unfortunately, convolutions are not generally invertible. Specialized constructions of invertible convolutions do exist in 1D in the form of wavelets [39] but these ideas do not extend easily to higher dimensions and the additional trappings for invertibility reduce the number of free parameters available for learning.

Glow [96] exploit the use of stacked convolutions common in CNNs to maintain many of the advantages of the convolutional layers while still maintaining invertibility when processing image data. Specifically, Glow utilizes 1×1 convolutional kernels that maintain channel depth. This means that Glow enjoys the extreme parameter efficiency and inductive bias of convolutions at the expense of the smallest possible kernel. In essence, Glow is performing the same matrix multiplication operation over channels for each pixel. As a result Glow keeps some of the advantages of invertible linear operations while maintaining the advantages of convolutions. They additionally propose to gain some spatial dependencies by reshaping local neighborhoods into new channels, resulting in larger matrix operations and increased flexibility. For example, if an image is $H \times W \times C$ prior the reshaping operation, it might be $H/2 \times W/2 \times 4C$ after the operation.

The full Glow network architecture composes their 1×1 convolutional layers with coupling operations using the checker board partitioning scheme and CNN-based conditioning networks. Reshaping operations are performed intermittently to help introduce multi-resolution and cross-pixel dependencies.

2.2.2.4 Continuous Normalizing Flows

Neural ordinary differential equations (NODEs) create a novel perspective on constructing networks implicitely as the solution to a differential equation. Normalizing flows that utilize NODEs for the transformation are known as continuous normalizing flows (CNFs) [67] and have demonstrated remarkable performance, especially for tabular data, and allows us to bypass some of the limitations we required in discrete normalizing flows, somewhat akin to conditioning networks in coupling transforms. This good performance and somewhat less-restricted construction comes at the cost of significantly increased compute.

Utilizing a continuous transformation instead of a composition of discrete transformation requires that we modify Eq. 2.8 to

$$\log p(z(s_0)) = \log p(z(s_1)) + \int_{s_0}^{s_1} \operatorname{Tr}\left(\frac{\partial f}{\partial z}(s)\right) ds \tag{2.11}$$

where $z(s_0) = x$. Equation 2.11 must be solved concurrently with the transformation from x to z as a system on ODEs within the same black-box ODE solver. The trace operation can be computationally expensive to calculate and integrate over. FFJORD [67] proposes to reduce this complexity through the use of the unbiased Hutchinson trace estimator [85]. This process approximates the trace as the expectation of a bilinear product of the Jacobian with standard Gaussian (or Rademacher) noise such that Eq. 2.11 becomes

$$\log p(z(s_0)) = \log p(z(s_1)) + \mathbb{E}_{\mathcal{N}(v)}\left[\int_{s_0}^{s_1} v^T \frac{\partial f}{\partial z}(s)\, v\, ds\right] \tag{2.12}$$

Due to their expense, we primarily use CNFs to construct smoothly-varying curriculums and not as a layer within our normalizing flows.

2.2.3 Autoregressive Models

Autoregressive (AR) methods exploit the probabilistic chain rule to model the data one dimension at a time, conditioned on the observations of the previously seen dimensions

$$\log p(x_1, \ldots, x_M) = \sum_{m=1}^{M} \log p(x_m \mid x_{m-1}, \ldots, x_1) \tag{2.13}$$

in an inherently iterative process. For example, we can learn a flexible likelihood for each dimension using a Gaussian mixture model where the parameters of the GMM depend on the specific values of the previously seen dimensions such that

$$p(x_m|x_{<m}) = \sum_k \pi_k(x_{<m})\, \mathcal{N}\left(\mu_k(x_{<m}), \sigma_k^2(x_{<m})\right) \tag{2.14}$$

and

$$\left[\pi_k(x_{<m}), \mu_k(x_{<m}), \sigma_k^2(x_{<m})\right] = f_{m,k}(x_{<m}; \theta_{m,k}) \tag{2.15}$$

where $x_{<m}$ represents the set of all features with index less than m, e.g., $x_{<m} = (x_1, x_2, ..., x_{m-1})$.

A variety of methods exist for choosing $f_{m,k}$ to construct sufficiently powerful models based on different inductive biases and efficiencies, [7, 61, 131, 167]. Autoregressive models typically achieve better performance than normalizing flows but come at the expense of increased evaluation and sampling cost due to the sequential nature of the models. AR models can more easily handle missing data than normalizing flows since flows typically require all the data be available at every stage, though recent efforts have reduced this difficulty [110]. An obvious shortcoming of AR models is the order of the conditionals is fixed and may be arbitrary, *e.g.*, it may be easy to model x_1 given x_2 whereas the inverse distribution may be more difficult. Akin to TANS [131], we will use AR models in conjunction with normalizing flows to model the latent performance, relieving some of the shortcomings of both methods and allowing for a more flexible model.

CHAPTER 3: FLOWSCANS

We begin by developing expanding the capabilities of deep learning density estimation to exchangeable, non-*i.i.d.* data. This method, FlowScan, combines invertible flow transformations with a sorted scan to flexibly model the data while preserving exchangeability. Unlike most existing methods, FlowScan exploits the intradependencies within sets to learn both global and local structure and represents *the first approach* that is able to apply sequential methods to exchangeable density estimation without resorting to averaging over all possible permutations.

3.1 Overview

Modeling unordered, non-*i.i.d.* data is an important problem in machine learning and data science. Collections of data objects with complicated intrinsic relationships are ubiquitous. These collections include sets of 3d points [176] sampled from the surface of complicated shapes like human organs, sets of images shared within the same web page, or point cloud LiDAR data observed by driverless cars [172]. In any of these cases, the collections of data objects do not possess any inherent ordering of their elements. Thus, any generative model which takes these data as input should not depend on the order in which the elements are presented *and* must be flexible enough to capture the dependencies between co-occurring elements.

The unorderedness of these kinds of collections is captured probabilistically by the notion of *exchangeability*. Formally, a set of points $\{x_j\}_{j=1}^n \subset \mathbb{R}^d$ with cardinality n, dimension d, and probability density $p(\cdot)$ is called exchangeable if

$$p(x_1, ..., x_n) = p(x_{\pi_1}, ..., x_{\pi_n}) \tag{3.1}$$

Figure 3.1: A training dataset of sets. Each instance $\mathcal{X}_i$ is a set of points $\mathcal{X}_i = \{x_{i,j} \in \mathbb{R}^d\}_{j=1}^{n_i}$ ($d = 2$ shown). We estimate $p(\mathcal{X}_i)$, from which we can sample distinct sets.

for every permutation π. In practice $\{x_j\}_{j=1}^n$ often represent 2d or 3d spatial points (see Fig. 3.1) in which case we refer to the set as a point cloud. In other settings, the points of interest may be more complex like images represented as very high-dimensional vectors.

As a simple example, one may trivially generate a set of exchangeable points by drawing them *i.i.d.* from some distribution. More commonly, elements within an exchangeable set share information with one another, providing structure. Despite the abundance of such data, the bulk of existing approaches either ignore the relation between points (*i.i.d.* methods) or model dependencies in a manner that depends on inherent orderings (sequential methods) [145, 183]. In order to accurately learn the structure of a set whilst preserving the exchangeability of its likelihood, one cannot rely solely on either approach.

In this chapter, we focus on the task of tractable, non-*i.i.d.* density estimation for exchangeable sets. We explore both low cardinality sets of high dimension (10-20 points with many hundreds of dimensions each, *e.g.*,collections of images) and high cardinality sets of low dimension (hundreds of points with 2-7 dimensions each, *e.g.*,point clouds). We develop a generative model suitable for exchangeable sets in either regime, called FlowScan, which does not rely on *i.i.d.* assumptions and is provably exchangeable. Contrary to intuition, we show that one can preserve exchangeability while scanning over the data in a sorted manner. *FlowScan is the first method to achieve a tractable, non-*i.i.d.*, exchangeable likelihood by leveraging traditional (e.g.,sequential), non-exchangeable density estimators.* This chapter is adapted from our original work, "Exchangeable Generate Model with Flow Scans," [14].

Main Contributions. 1) We show that transforming points with an equivariant change of variables allows for modeling sets in a different space. 2) We introduce a scanning-based technique for modeling exchangeable data, relating the underlying exchangeable likelihood to that of the sorted covariates. 3) We demonstrate how traditional density estimators may be used for the task of principled and feasible exchangeable density estimation via a scanning-based approach. 4) We show empirically that FlowScan achieves the state-of-the-art for density estimation tasks in both synthetic and real-world point cloud and image set datasets.

3.2 Motivation and Challenges

We motivate our problem with a simple, yet common, set generative process that requires a *non-i.i.d.*, exchangeable density estimator. Consider the following generative process for a set: 1) generate *latent* "parameters" $\phi \sim p_\Phi(\cdot)$ and then 2) generate a set $\mathcal{X} \sim p(\cdot \mid \phi)$. Here $p(\cdot \mid \phi)$ may be as simple as a Gaussian model (where ϕ is the mean and covariance parameters) or as complex as a nonparametric model (where ϕ may be infinite-dimensional).

This simple set generative process requires a *non-i.i.d.* approach, even for the case when the ground truth conditional set likelihood, $p(\mathcal{X} \mid \phi)$, is *conditionally i.i.d.*. Somewhat akin to De Finetti's theorem [17], we show this by first noting that with *conditionally i.i.d.* $p(\mathcal{X} \mid \phi) = \prod_{j=1}^{n} p(x_j \mid \phi)$, the complete set likelihood is:

$$p(\mathcal{X}) = \int p_\Phi(\phi) \prod_{j=1}^{n} p(x_j \mid \phi) \, \mathrm{d}\phi. \tag{3.2}$$

One can show dependency (*non-i.i.d.*) with the conditional likelihood of a single point x_k given a disjoint subset $S \subset \mathcal{X} \setminus \{x_k\}$:

$$\begin{aligned} p(x_k \mid S) &= \int p_\Phi(\phi \mid S)\, p(x_k \mid S, \phi)\, \mathrm{d}\phi \\ &= \int p_\Phi(\phi \mid S)\, p(x_k \mid \phi)\, \mathrm{d}\phi \\ &\neq \int p_\Phi(\phi)\, p(x_k \mid \phi)\, \mathrm{d}\phi = p(x_k). \end{aligned}$$

That is, the conditional likelihood $p(x_k \mid S)$ depends on other points in $\mathcal{X}$ via the posterior $p_\Phi(\phi \mid S)$, which accounts for what ϕ was likely to have generated S and not through S directly, *e.g.*, $p(x_k \mid S, \phi) = p(x_k \mid \phi)$. As a consequence, the complete generative process Eq. 3.2 is *not marginally i.i.d.*, notwithstanding the *conditional i.i.d.* $p(\mathcal{X} \mid \phi)$. Thus, any model built on an *i.i.d.* assumption may be severely biased.

The generative process in Eq. 3.2 is especially applicable for surface point cloud data. For such sets, $\mathcal{X}_i$, points are drawn *i.i.d.* from (conditioned on) the surface of a shape with (*unknown*) parameters ϕ_i (*e.g.*,object class, length, orientation, noise, etc.), resulting in the dataset $\mathcal{D} = \{\mathcal{X}_i \sim p(\cdot \mid \phi_i)\}_{i=1}^N$ of N sets. As shown above, modeling such point cloud set data requires a *non-i.i.d.* approach even though points may be drawn independently given the surface parameters. FlowScan will not only yield an exchangeable, *non-i.i.d.* generative model, but will also directly model elements in sets without latent parameters. In effect, FlowScan will automatically marginalize out dependence on latent parameters of a given set, and is thus capable of handling complicated $p(\cdot \mid \phi)$.

Broadly, the primary challenge in direct exchangeable density estimation is designing a flexible, invariant architecture which yields a valid likelihood. As explained above, using an *i.i.d.* assumption to enforce this property will severely hamper the performance of a model. To avoid this simplification, techniques often shoehorn invariances to observed orderings by feeding randomly permuted data into sequential models [145, 183]. Such approaches attempt to average out

the likelihood of the model over all permutations:

$$p(\mathcal{X}) = \frac{1}{n!} \sum_{\pi} p_s(x_{\pi_1}, \ldots, x_{\pi_n}), \tag{3.3}$$

where p_s is some sequential model. Of course, the observation of all potential orderings for even a modest collection of points is infeasible. Furthermore, there are often no guarantees that the sequential model p_{seq} will learn to ignore orderings, especially for unseen test data [170].

Given that an *i.i.d.* assumption is not robust and averaging over all permutations is infeasible, what operation should be used to ensure permutation invariance of the architecture? Instead of attempting to wash out the effect of order in an architecture as in 3.3, we propose to enforce invariance by adopting a prespecified ordering and scanning over elements in this order. As will be discussed in the Methods section, the benefit of estimating a likelihood over sorted data is that it frees us from the restriction of exchangeability. Given the sorted data, we can apply any number of traditional density estimators. However, such an approach presents its own challenges:

- **Determining a suitable way to scan through an exchangeable sequence.** That is, one must map the set $\mathcal{X} = \{x_j\}_{j=1}^n$ to a sequence $\mathcal{X} \mapsto (x_{[1]}, \ldots, x_{[n]})$ where $x_{[j]}$ denotes the j'th point in the sorted order.

- **Relating the likelihood of the scanned sequence to likelihood of the exchangeable set.** Modeling the exchangeable likelihood through a scanned likelihood is not immediately obvious; *a simple equality of the two does not hold*, $p(\mathcal{X}) \neq p(x_{[1]}, \ldots, x_{[n]})$.

- **Scanning in a space that is beneficial for modeling.** The native input space may not be best suited for modeling or scanning, hence it would be constructive to transform the exchangeable input prior to the scan.

- **Developing an architecture that exploits the structure gained in the scan.** The scanning operation will introduce sequential correlations among elements which need to be modeled successfully.

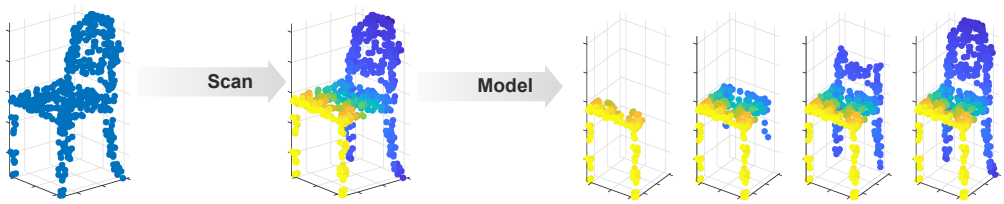


Figure 3.2: Illustration of our proposed method. First, input sets are scanned (in a possibly transformed space). After, the scanned covariates are modeled (possibly in a autoregressive fashion, as shown).

Next, we develop the FlowScan model while addressing each of these challenges.

3.3 Methods

FlowScan consists of three components: 1) a sequence of equivariant flow transformations, $\hat{q}_e$, *to map the data to a space that is easier to model*; 2) a sort with correction factor *to allow for the use of non-exchangeable density estimators*; 3) a density estimator ($\hat{p}_s$) (*e.g.*,an autoregressive model which may utilize sequential flow transformations, $\hat{q}_c$), *to estimate the likelihood while accounting for correlations induced by sorting* (see Fig. 3.2). In this section, we motivate each piece of the architecture and detail how they combine to yield a highly flexible, exchangeable density estimator.

3.3.1 Equivariant Flow Transformations

FlowScan first utilizes a sequence of equivariant flow transformations. So-called "flow models" rely on the change of variables formula to build highly effective models for traditional non-exchangeable generative tasks (like image modeling) [97]. Using the change of variables formula, flow models approximate the likelihood of a d-dimensional distribution over real-valued covariates $x = (x^{(1)}, ..., x^{(d)}) \in \mathbb{R}^d$, by applying an invertible (flow) transformation $\hat{q}(x)$ to an estimated base distribution $\hat{f}$:

$$\hat{p}(x^{(1)}, ..., x^{(d)}) = \left| \det \frac{\mathrm{d}\hat{q}}{\mathrm{d}x} \right| \hat{f}(\hat{q}(x)), \tag{3.4}$$

where $|\det\frac{\mathrm{d}\hat{q}}{\mathrm{d}x}|$ is the Jacobian of the transformation $\hat{q}$. Often, the base distribution is a standard Gaussian. However, [131] recently showed that performance may be improved with a more flexible base distribution on transformed covariates such as an autoreggressive density [**?** 68, 103, 166, 165].

There are a myriad of possible invertible transformations, $\hat{q}$, that one may apply to inputs $\mathbf{x} \in \mathbb{R}^{n\times d}$ in order to model elements in a more expressive space, where $\mathbf{x}$ is the set $\mathcal{X}$ represented as a matrix, see Sec. 2.2.2. However, in our case one must take care to preserve exchangeability of the inputs when transforming the data. For example, a simple affine change of variables will be sensitive to the order in which the elements of $\mathbf{x}$ were observed, resulting in a space which is no longer exchangeable. One can circumvent this problem by requiring that any transformation, $\hat{q}$, used is *equivariant*. That is, for all permutation operators, Γ, we have that $\hat{q}(\Gamma\mathbf{x}) = \Gamma\hat{q}(\mathbf{x})$. Proposition 1 states that equivariance of the transformations in conjunction with invariance of the base distribution is enough to ensure that exchangeability is preserved, *allowing one to model set data in a transformed space*. The proof is straightforward and relegated to the Appendix.

Proposition 1. *Let* $\hat{q} : \mathbb{R}^{n\times d} \mapsto \mathbb{R}^{n\times d}$ *be a* permutation equivariant*, invertible transformation and the base distribution,* $\hat{f}$*, be exchangeable. Then the likelihood,* $\hat{p}(\mathbf{x}) = \left|\det\frac{\mathrm{d}\hat{q}}{\mathrm{d}\mathbf{x}}\right|\hat{f}(\hat{q}(\mathbf{x}))$*, is exchangeable.*

Given an invertible transformation, $q : \mathbb{R}^d \to \mathbb{R}^d$, one may construct a simple permutation equivariant transformation by applying it to each point in a set independently: $(x_1, ..., x_n) \mapsto (q(x_1), ..., q(x_n))$. However, it is possible to engineer equivariant transformations which utilize information from other points in the set while still preserving equivariance. Proposition 1 shows that FlowScan is compatible with any combination of these transformations.

Set-Coupling Among others, we propose a novel set-level scaling and shifting coupling transformation akin to those used in Real-NVP [47], see Sec. 2.2.2.2. For d-dimensional points, the coupling transformation scales and shifts one subset, $S \subset \{1, \ldots, d\}$ of the d covariates given the

rest, S^c, as (letting superscripts index point dimensions):

$$\begin{aligned} x^{(S)} &\mapsto x^{(S)} \exp\left(f\left(x^{(S^c)}\right)\right) + g\left(x^{(S^c)}\right) \\ x^{(S^c)} &\mapsto x^{(S^c)}, \end{aligned} \tag{3.5}$$

for learned functions $f, g : \mathbb{R}^{|S^c|} \mapsto \mathbb{R}^{|S|}$. We propose a set-coupling transformation as follows:

$$\begin{aligned} x_i^{(S)} &\mapsto x_i^{(S)} \exp\left(f\left(\varphi(\mathbf{x}^{(S^c)}), x_i^{(S^c)}\right)\right) + g\left(\varphi(\mathbf{x}^{(S^c)}), x_i^{(S^c)}\right) \\ x_i^{(S^c)} &\mapsto x_i^{(S^c)}, \end{aligned} \tag{3.6}$$

where $\mathbf{x}^{(S^c)} \in \mathbb{R}^{n \times |S^c|}$ is the set of unchanged covariates, $\varphi(\mathbf{x}^{(S^c)}) \in \mathbb{R}^r$ are general, learnable permutation invariant set embeddings. We utilize embeddings from DeepSet architectures [184]; however, other embeddings are possible, *e.g.*,prescribed statistics [88], and $f, g : \mathbb{R}^{r+|S^c|} \to \mathbb{R}^{|S|}$ are learned functions. The embedding φ is responsible for capturing set-level information from other covariates. This is combined with each point $x_i^{(S^c)}$ to yield shifts and scales with both point- and set-level dependence (see Fig. 3.3). The log-determinant and inverse are detailed in the Appendix along with several other examples of flexible, equivariant transformations.

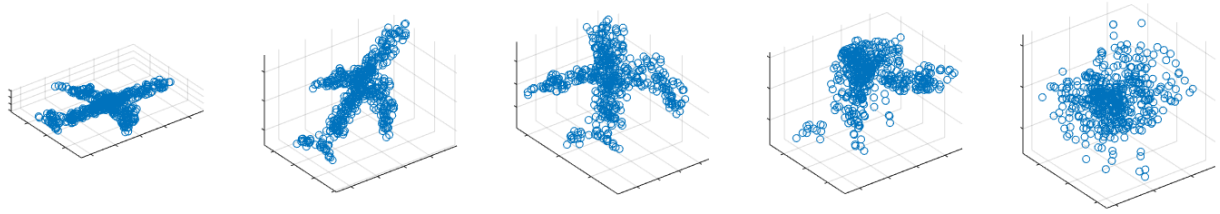


Figure 3.3: An illustration of how set-coupling transformations act on a set. The first plot shows the input data to be transformed. In the subsequent plots, the set is transformed in an invertible, equivariant fashion by stacking set-coupling transformations. Iteratively transforming dimensions of a set in this way yields a set with simpler structure that may be modeled more easily, as shown in the last plot.

3.1.2 Invariance Through Sorting

After applying a series of equivariant flow transformations, FlowScan performs a sort operation and corrects the likelihood with a factor of $1/n!$. Sorting in a prespecified fashion ensures that different permutations of the input map to the same output. In this section, we prove that this yields an analytically correct likelihood and comment on the advantages of such an approach. Specifically, we show that the exchangeable (unordered) likelihood of a set of n points $p_e(x_1, \dots x_n)$ (where $x_j \in \mathbb{R}^d$) can be written in terms of the non-exchangeable (ordered) likelihood of the points in a sorted order $p_s(x_{[1]}, \dots, x_{[n]})$ as stated in Prop. 2 below.

Proposition 2. *Let p_e be an exchangeable likelihood which is continuous and non-degenerate (e.g.,$\forall j \in \{1, \dots, d\}$ $\Pr[x_1^{(j)} \neq x_2^{(j)} \neq \dots \neq x_n^{(j)}] = 1$ — that is, all values will be unique with probability one). Then,*

$$p_e(x_1, \dots x_n) = \frac{1}{n!} p_s(x_{[1]}, \dots, x_{[n]}), \tag{3.7}$$

where $x_{[j]}$ is the jth point in the sorted order.

Proof. We derive 3.7 from a variant of the change of variables formula [27]. It states that if we have a partition of our input space, $\{\mathcal{A}_j\}_{j=1}^M$, such that a transformation of variables q is invertible in each partition $\mathcal{A}_j$ with inverse q_j^{-1}, then we may write the likelihood f of $z = q(u)$ in terms of the likelihood p of the input data u as:

$$f(z) = \sum_{j=1}^{M} \left| \det \frac{\mathrm{d}q_j^{-1}}{\mathrm{d}z} \right| p(q_j^{-1}(z)). \tag{3.8}$$

For the moment, suppose that the points $\{x_j\}_{j=1}^n$ are sorted according to the first dimension. That is, $x_{[1]}, \dots, x_{[n]}$ in 3.7 are such that $x_{[1]}^{(1)} < \dots < x_{[n]}^{(1)}$. The act of sorting these points amounts to a transformation of variables $s : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$, $s(x_1, \dots, x_n) = (x_{[1]}, \dots, x_{[n]})$. The transformation s is one-to-one on the partitions of the input space $\mathbb{R}^{n \times d}$ defined by the relative order of points. In other words, we may partition the input space according to the permutation

that would sort the data: $\mathcal{A}_\pi = \{\mathbf{x} \in \mathbb{R}^{n \times d} \mid x^{(1)}_{\pi_1} < x^{(1)}_{\pi_2} < \ldots < x^{(1)}_{\pi_n}\}$. We may invert s in $\mathcal{A}_\pi$ via the inverse permutation matrix of π, Γ_π^{-1}. Letting Π be the set of all permutations, 3.8 yields:

$$p_s(s(\mathbf{x})) \overset{*}{=} \sum_{\pi \in \Pi} \left|\Gamma_\pi^{-1}\right| p_e(\Gamma_\pi^{-1} s(\mathbf{x})) \overset{**}{=} n!\, p_e(\mathbf{x}), \tag{3.9}$$

where (*) follows from 3.8 and (**) follows from the exchangeability of p_e. Thus, we may compute the exchangeable likelihood $p_e(\mathbf{x})$ using the likelihood of the sorted points, as in 3.7. Trivially, similar arguments also hold when sorting according to a dimension other than the first. Furthermore, it is possible to sort according to any appropriately transformed space of x_j, rather than any native dimension itself (as this is equivalent to applying a transformation, sorting, and inverting said transformation). □

Consequently, the exchangeable likelihood may be estimated via an approximation of the scanned covariates: $p_e(\mathbf{x}) \approx \frac{1}{n!}\hat{p}_s(s(\mathbf{x}))$. Since the density of sorted scan is not exchangeable, we may estimate $\hat{p}_s$ using traditional density estimation techniques. *This gives a principled approach to reduce the problem of exchangeable likelihood estimation to a flat vector (or sequence) likelihood estimation task.*

3.1.3 Autoregressive Scan Likelihood

After performing equivariant flow transformations and sorting, FlowScan applies a non-exchangeable density estimator to model the transformed and sorted data. Let $z = s(\hat{q}(\mathbf{x})) \in \mathbb{R}^{n \times d}$ be the sorted covariates. Since z is not exchangeable, one can apply any traditional likelihood estimator on its covariates, *e.g.*, one may treat z as a vector and model $\hat{p}_s(\mathrm{vec}(z))$ using a flat density estimator. However, flattening in this way suffers from several disadvantages. First, it is inflexible to varying cardinalities. Furthermore, the total number of covariates, nd, may be large for sets with large cardinality or dimensionality. Finally, a general flat model loses the context that covariates are from multiple points in some shared set. To address these challenges, we

use an autoregressive likelihood:

$$\hat{p}(z_k) = \prod_{k=1}^{n} \hat{p}(z_k \mid h_{<k}), \tag{3.10}$$

where $\hat{p}(z_k \mid h_{<k})$ is itself a d-dimensional density estimator (such as 3.4) conditioned on a recurrent state $h_{<k} = h(z_1, \ldots, z_{k-1})$. This proposed approach is capable of sharing parameters across the n d-dimensional likelihoods and is more amenable to large, possibly varying, cardinalities.

3.1.4 Correspondence Flow Transformations

In much the same way that nearby pixels are correlated in image space, points with neighboring indices will be correlated in a scan space. Thus, we also propose a coupling [45] invertible transformation to transform adjacent points, exploiting existing correlations among points as follows. We note that it is straightforward to use a sequential coupling transformation to shift and scale points z_i as in 3.5, but based on inputting a recurrent output $h_{<i}$ to f and g functions. In addition, it is also possible to split individual points for coupling as follows. First, split the scanned points $z = s(\hat{q}(\mathbf{x})) = (z_1, \ldots, z_n)$ into two groups depending on the parity (even/odd) of their respective index. Second, transform each even point, with a scale and shift based on the corresponding odd point. That is for pairs of points (z_{2j}, z_{2j+1}) we perform the following transformation: $(z_{2j}, z_{2j+1}) \mapsto (s(z_{2j+1})z_{2j} + m(z_{2j+1}),\ z_{2j+1})$, where $s : \mathbb{R}^d \mapsto \mathbb{R}^d, m : \mathbb{R}^d \mapsto \mathbb{R}^d$ are scale and shifting functions, respectively, parameterized by a learnable fully connected network. This correspondence coupling transformation $z \mapsto z'$ is easily invertible and has analytical Jacobian determinant $\left|\det \frac{\mathrm{d}z'}{\mathrm{d}z}\right| = \prod_{j=0}^{n/2-1} |s(z_{2j+1})|$. Several of these transformations may be stacked before the autoregressive likelihood by alternating between shifting and scaling even points based on odd and vice-versa odd points based on even. We shall also make use of a similar splitting scheme to split sets of images into 3d tensors that are fed into 3d convolution networks for shifting and scaling.

3.1.5 Complete FlowScan Architecture

Since the scanned likelihood in 3.7 yields an exchangeable likelihood, one may use as the base likelihood following a permutation equivariant transformation as in Prop. 1. This enables us to apply the sorting step after performing any number of equivariant transformations and improve the flexibility of the model as a result. As no generality is lost, we choose to sort on the first dimension in our experiments detailed below. Combining the three components detailed above, we arrive at the complete FlowScan architecture: a sequence of equivariant flow transformations, a sort with correction factor, and an autoregressive scan likelihood. The estimated exchangeable likelihood that results is:

$$\hat{p}_{\text{fs}}(\mathbf{x}) = \frac{1}{n!} \left| \det \frac{\mathrm{d}\hat{q}_{\text{e}}}{\mathrm{d}x} \right| \hat{p}_s(s(\hat{q}_{\text{e}}(\mathbf{x}))), \tag{3.11}$$

where $\hat{q}_{\text{e}}$ and $\hat{p}_s$ are the estimated (via maximum likelihood) equivariant flow transformation and sorted flow scan covariate likelihood, respectively. When correspondence flow transformations are included after the sort operation, we obtain an estimated exchangeable likelihood:

$$\hat{p}_{\text{fs}}(\mathbf{x}) = \frac{1}{n!} \left| \det \frac{\mathrm{d}\hat{q}_{\text{e}}}{\mathrm{d}x} \right| \left| \det \frac{\mathrm{d}\hat{q}_{\text{c}}}{\mathrm{d}x} \right| \prod_{k=1}^{n} \hat{p}(\mathbf{z}_k \mid h(\mathbf{z}_{<k})), \tag{3.12}$$

where $\mathbf{z}$ is the resulting covariates from corresponding coupling transforming the flow scanned covariates. In both cases, FlowScan gives a valid, provably exchangeable density estimate relying neither on variational lower bounds of the likelihood nor averaging over all possible permutations of the inputs. Furthermore, FlowScan is easily adapted to input sets with varying cardinalities, as is commonly observed in practice. In the Experiments section, we demonstrate empirically that FlowScan is highly flexible and capable of modeling sets of both points clouds and images.

3.2 Related Work

Unlike the recent surge in flexible density estimation for flat vectors with deep architectures [45, 47**?** , 68, 96, 103, 132, 165, 166], few efforts cover exchangeable treatments of data in ML with some notable exceptions. Some recent work [108, 143, 184] has explored neural architectures for constructing a permutation invariant set embeddings. They featurize input sets exchangeably in a way that is useful for (typically supervised) downstream tasks; *but the embeddings themselves will not result in valid likelihoods*. In other work, Generative Adversarial Networks (GAN) have been explored as a means of sampling point clouds [185]. However, none of these methods provide a valid exchangeable likelihood estimate as is our focus.

A recently proposed model, BRUNO [99], preserves exchangeability by performing independent point-wise changes of variables, a simple equivariant linear transformation, and an *i.i.d.* base exchangeable process in the latent space. The Neural Statistician (NS) [51] estimates a permutation invariant code produced by an exchangeable VAE. That is, the Neural Statistician uses an encoder, called a statistics network, on the entire exchangeable set to get an approximate posterior on the latent code. Given the success of a point cloud autoencoder with a DeepSet network as the statistics network in [131], we consider this architecture for the variational Neural Statistician which is an especially strong baseline, representing the state-of-the-art likelihood method for point cloud data.

3.3 Experiments

In this section, we compare the performance of FlowScan to that of BRUNO and NS in a variety of exchangeable point cloud and image modeling tasks. In each experiment, our goal is to estimate an exchangeable likelihood $p(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^{n \times d}$ which models the inputs well. As is standard in density estimation tasks, we measure the success of the model via the estimated likelihood of a held out test set for each experiment. For readability, we report the estimated log likelihood divided by the number of points (per point log likelihoods, PPLL): $\frac{1}{n} \log \hat{p}(\mathbf{x})$. The

PPLL provides a set-level likelihood that eases comparison across dataset and cardinality. As NS does not yield a likelihood, we report its estimated variational lower bound on the PPLL. Results for each datasets can be found in Tab. 3.1.

As a qualitative assessment of each model's performance, we also include samples generated by each trained model. Those which are not reported in the main text can be found in the Appendix. Unless stated explicitly, the figures included are *not reconstructions*, but completely synthetic point clouds or images generated by each model. Further implementation details (including code and Appendices) can be found at `https://github.com/lupalab/flowscan`.

3.3.1 Shuffled Synthetic Sequential Data

We begin with a synthetic point cloud experiment to test FlowScan's ability to learn a known, ground truth likelihood. To allow for complex interactions between points, we study a common scenario that leads to exchangeable data: sequential data with time marginalized out. In other words, we suppose that all time-points $x_j \in \mathbb{R}^d$ of a sequence $(x_1, \ldots, x_n)$ are put into an unordered set $\{x_1, \ldots, x_n\}$. Effectively, this yields observations of sequences in matrices that are randomly shuffled from the sequential order. Hence, exchangeable instances are $\mathbf{x} = \Gamma_\pi \mathbf{x}_s$, for permutations $\Gamma_\pi \in \mathbb{R}^{n \times n}$ (drawn uniformly at random) and sequential data $\mathbf{x}_s = (x_1, \ldots, x_n) \in \mathbb{R}^{n \times d}$ (drawn via a sequential likelihood p_{seq}). Here we consider a synthetic ground truth sequential model p_{seq} where the likelihood of an instance is computed by marginalizing out the permutation: $p(\mathbf{x}) = \sum_{\pi'} \Pr(\pi = \pi')\, p_{\text{seq}}(\Gamma_{\pi'}^{-1}\mathbf{x}) = \frac{1}{n!} \sum_\pi p_{\text{seq}}(\Gamma_\pi \mathbf{x})$. To obtain interesting non-linear

Dataset	BRUNO	NS	FlowScan
`Synthetic`	-2.28	-1.07	**0.14**
`Airplanes`	2.71	4.09	**4.81**
`Chairs`	0.75	2.02	**2.58**
`ModelNet10`	0.49	2.12	**3.01**
`ModelNet10a`	1.20	2.82	**3.58**
`Caudate`	1.29	4.49	**4.87**
`Thalamus`	-0.815	2.69	**3.12**
`SpatialMNIST`	-5.68	-5.37	**-5.26**

Table 3.1: Per-point log-likelihood (PPLL) of the test set for all point cloud experiments. Higher PPLL indicates better modeling of the test set.

dependencies we consider a sinusoidal sequence (see Fig. 3.4 and Appendix for details). To allow for computing the ground truth likelihood in a timely manner, we consider $n = 8$, leading to a large number, $8! = 40320$, of summands in the likelihood of the data.

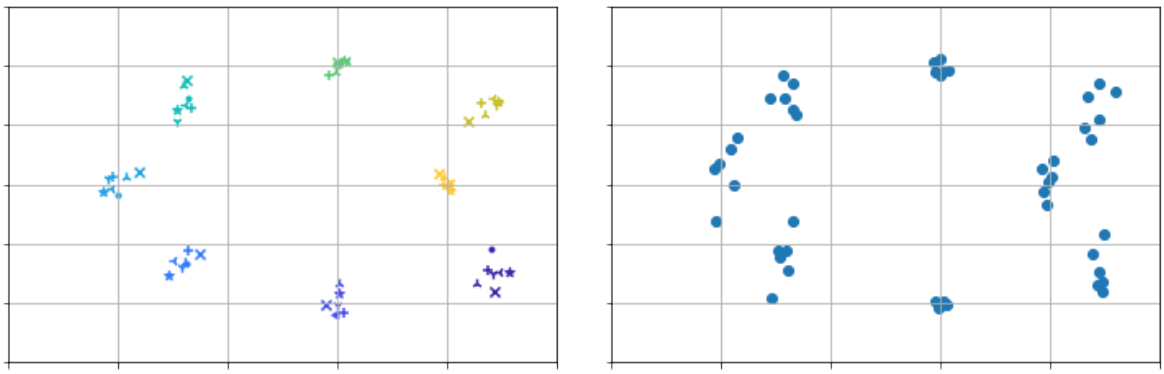
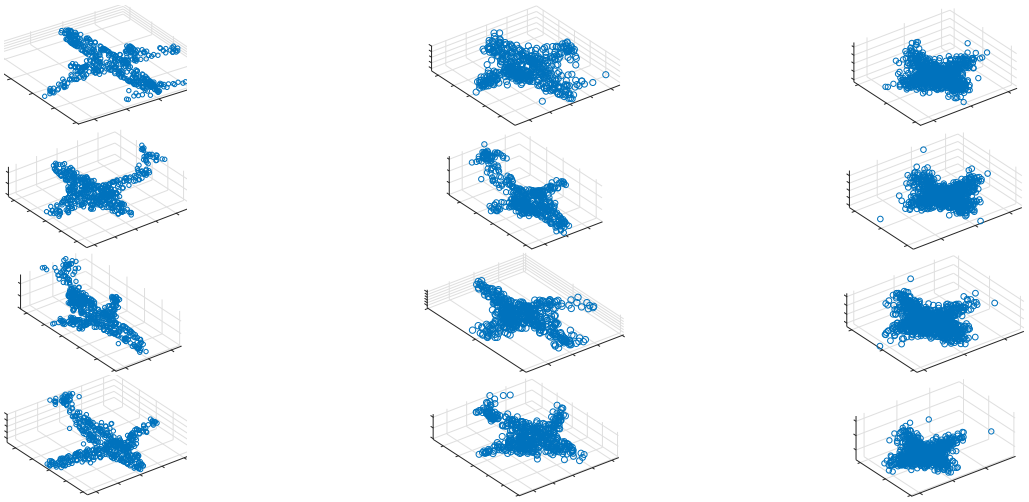


Figure 3.4: Left: true samples; markers and colors indicate instances and sequential order, respectively. Right: FlowScan samples.

Table 3.1 illustrates the per point log likelihood (PPLL) estimates across the synthetic sets using BRUNO, the NS, and FlowScan. The FlowScan model outperforms the other methods, achieving nearly the same PPLL as the ground truth (0.23) despite not averaging over all $n!$ permutations. For further comparison, we also trained a sequential model on the randomly permuted instances (and marginalizing out the permutation as in 3.3). However, randomly permuting the input sequence proved to be ineffective and resulted in low test PPLLs (with severe overfitting).

3.3.2 ModelNet

Next, we illustrate the efficacy of our model on real world point cloud data. We consider object classes from the ModelNet dataset [177], which contains CAD models of common real world objects. Point clouds were created by randomly sampling 512 points from the surface of each object. All point cloud sets are modeled in an unsupervised fashion. That is, we estimate $p(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^{512\times 3}$. Models are compared on the following datasets comprised of different subsets of point cloud classes: `airplanes`, `chairs`, `ModelNet10`, and `ModelNet10a`. `ModelNet10` is the standard subset [177] consisting of *bathtub*, *bed*, *chair*, *desk*, *dresser*, *mon-*



(a) FlowScan (b) NS (c) BRUNO

Figure 3.5: Synthetic plane samples from trained models

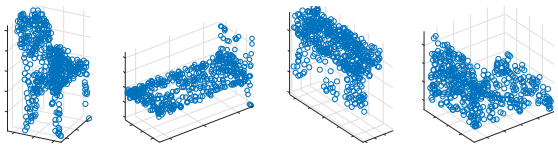


Figure 3.6: FlowScan `ModelNet10` samples

itor, *night stand*, *sofa*, *table*, and *toilet* classes. Since `ModelNet10` is composed largely of furniture-like objects, we also select a more diverse, ten-class subset that we will refer to as `ModelNet10a`, containing *airplane*, *bed*, *car*, *chair*, *guitar*, *lamp*, *laptop*, *plant*, *stairs*, and *table* classes.

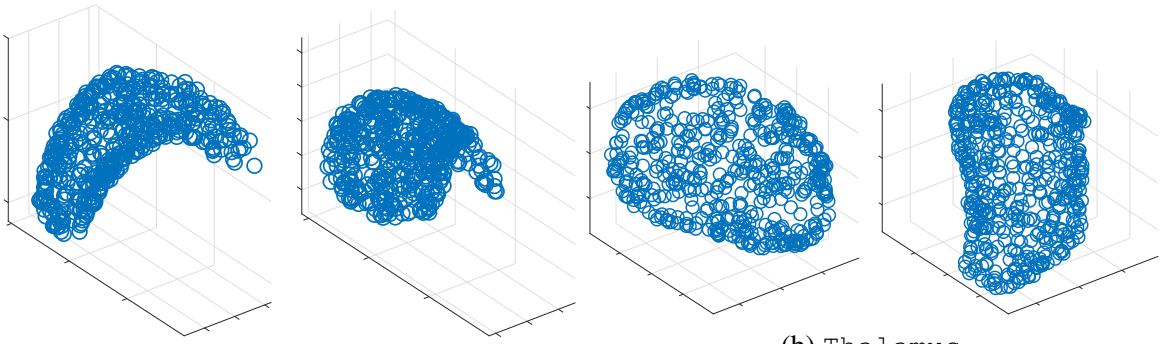
Results can be found in Tab. 3.1 and four samples from FlowScan are included in Fig. 3.6. For each of the four datasets tested, we find that FlowScan achieves the highest average test log-likelihood. Qualitatively, we also observe superior samples from the FlowScan model as can be seen in Fig. 3.5 and in the Appendix. In addition to training on these ModelNet datasets, we also performed an ablation study (see the Appendix) where we see that our full architecture yields the best performance over alternatives.

3.3.3 Brain Data

We test FlowScan's performance on a medical imaging task in a higher dimensional setting using samples of the Caudate and Thalamus [35]. Each set contains 512 randomly sampled 7d points. The first three dimensions contain the Cartesian coordinates of the surface boundary (as in ModelNet). The next two dimensions represent the normal direction at the boundary in terms of angles. The final two dimensions represent the local curvature (expressed as shape index and curvedness [98]). Table 3.1 enumerates the PPLL for both datasets across all three methods. Comparing samples from FlowScan (see Fig. 3.7) to that of NS and BRUNO (included in the Appendix) we see that FlowScan better captures the geometric features of the data than NS. Overall, superior PPLLs and samples suggest that FlowScan seamlessly incorporates the additional geometric information to model point clouds more accurately than baseline methods.

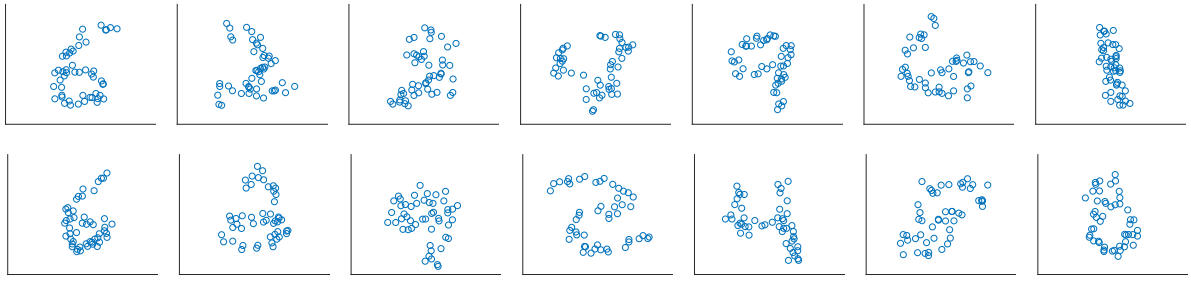
3.3.4 Spatial MNIST

For a direct comparison to NS, we also trained our model on the `SpatialMNIST` dataset, used by [51]. Each set consists of 50 points sampled uniformly at random from active pixels of a single `MNIST` [106] image with uniform noise added to ensure non-degeneracy. The dataset that results consists of 2-dimensional point clouds each representing a digit from 0 to 9. PPLLs for each model can be found in Tab. 3.1 and a random selection of samples from each can be found

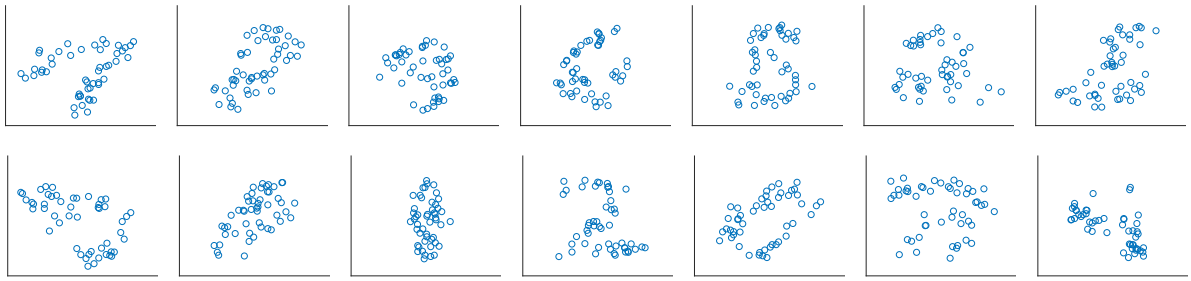


(a) `Caudate` (b) `Thalamus`

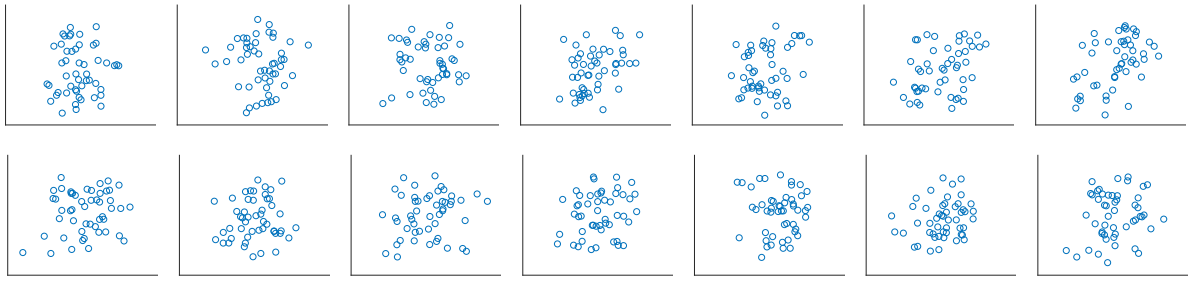
Figure 3.7: FlowScan `Caudate` and `Thalamus` samples



(a) FlowScan



(b) NS



(c) BRUNO

Figure 3.8: `SpatialMNIST` samples from each model.

in Fig. 3.8. Both the (unconditioned) likelihoods and the samples indicate that FlowScan gives superior performance in this task.

3.3.5 MNIST

Finally, we show that FlowScan exhibits superior likelihoods and samples in a high-dimensional, low-cardinality setting. Following [99], sets are composed of 20 random images corresponding to the *same digit class* from the `MNIST` dataset. After training, PPLLs are evaluated on held out

test sets constructed from unseen images. Our baseline is BRUNO, which achieves a PPLL of -643.6. BRUNO's unconditional samples (Fig. 3.9c) often contain elements from different digits, indicating a lack of intra-set dependency in the resulting model. We improve upon BRUNO by first adding convolution-based Set-Coupling transformations (but keeping the *i.i.d.* base likelihood), which achieves a PPLL of -634.8. Still, sample sets (Fig. 3.9b) show mixed digit classes. Finally, we consider a full FlowScan model that adds a sort, scan, and 3d convolution-based correspondence coupling transformations, which achieves the best PPLL of -621.7. Furthermore, FlowScan samples consistently contain the same digit class (Fig. 3.9a), showing that we are able to fully model the intra-set dependencies of elements.

3.4 Limitations

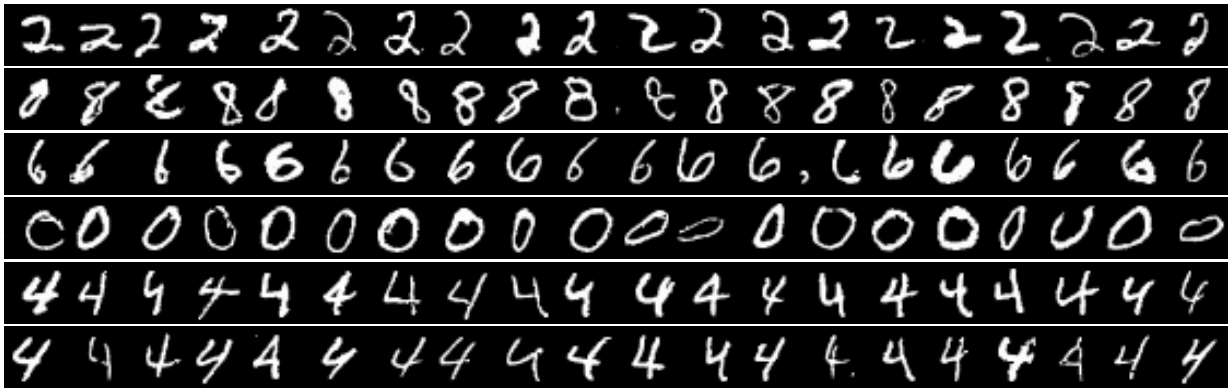
While FlowScan are demonstrably powerful likelihood models for exchangeable data, they suffer from a few shortcomings. First, these models require that the input data must be continuous so that we do not violate the assumptions in Proposition 2, *e.g.*, that the we will not have ties with probability one and that we cannot deterministically recover any subset of features given the remaining features. In a precise sense, this limits the types of data that our model is capable of consuming. This limitation is shared with common, fixed-order likelihood models such as normalizing flows. However, it is standard practice when training these classes of models to add a small amount of (continuous) random noise. This introduces a continuous component to the data that meets the conditions required to utilize our, and other, methods.

The second shortcoming pertains to the use of the sorting operation itself. This operation enables the critical bridge from the exchangeable to sequential spaces and, while we can perform backpropogation through the operation via the appropriate permutation matrix, the operation itself cannot be updated via gradient descent. The model must implicitly learn how to organize information such that the scan will construct a pertinent ordering that the downstream sequential model can most easily model. Our results demonstrate the effectiveness of this implicit learning

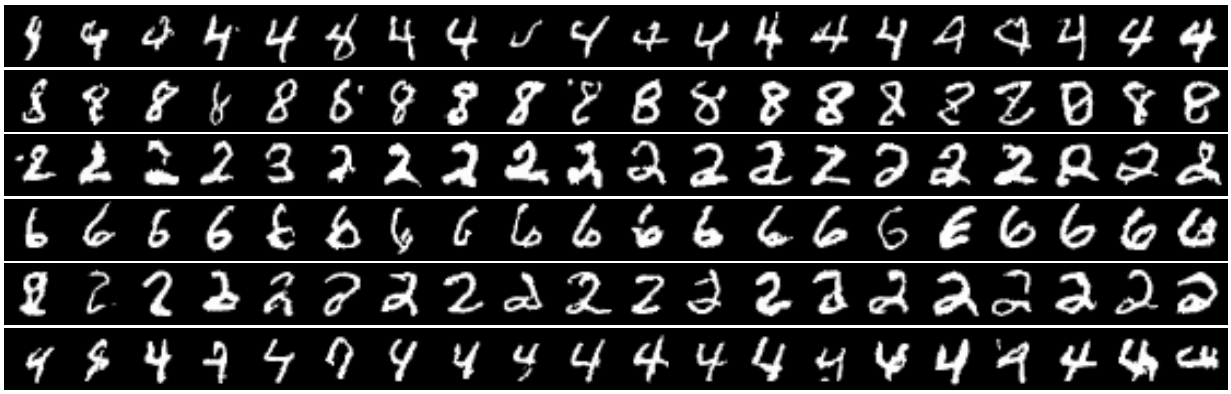
process, however, it would likely improve training efficiency and overall performance if the importance of the sorting dimension was explicit in the learning process.

3.5 Conclusion

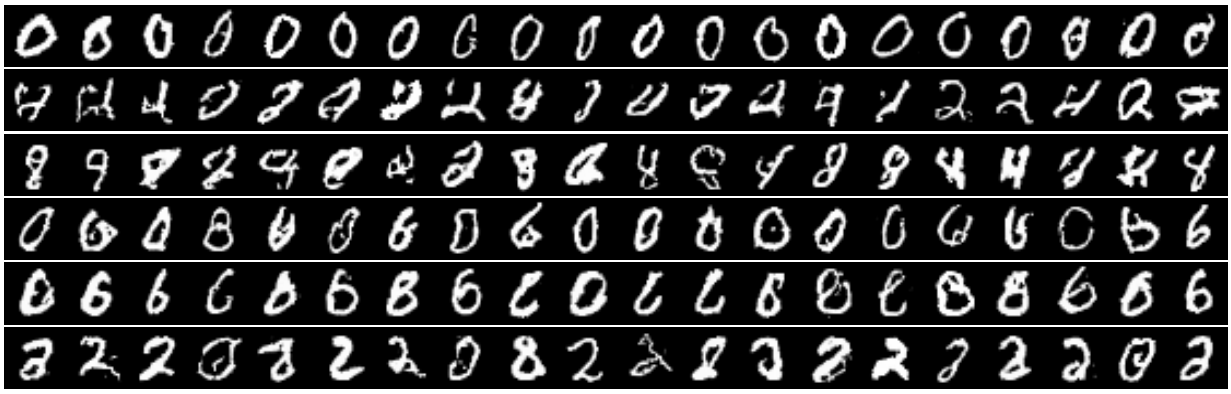
In this chapter, we expanded the capabilities of modern deep learning density estimation to exchangeable, non-*i.i.d.* data by introducing FlowScan and demostrated considerable improvements over alternate methods that rely on simpler transformations, latent distributions, or approximations. This is a difficult task, where models were previously limited to either exchangeable base likelihoods such as Bruno [99], or conditionally *i.i.d.* restrictions with variational approximations of the likelihood like the Neural Statistician [51]. We explored how to map inputs to a space that is easier to model whilst preserving exchangeability via equivariant flow transformations. Among others, we proposed the Set-Coupling transformation which extends existing pointwise coupling transformations [45] to sets. Additionally, we demonstrated how to apply non-exchangeable density estimators to this task via sorting and scanning. This is the first tractable approach to achieve this, avoiding averaging over any permutations of the data while unlocking a much larger class of base likelihoods for exchangeable density estimation. Finally, we argued for the use of an autoregressive base likelihood with sequential transformations to exploit the sequential structure gained in the sort and scan. Combining equivariant flow transformations, sorting and scanning, and an autoregressive likelihood, we arrived at FlowScan. We showed empirically that FlowScan's ability to model intradependencies within sets surpassed that of other state-of-the-art methods in both high-cardinality, low-dimensionality and low-cardinality, high-dimensionality settings. Quantitatively FlowScan's likelihoods were a substantial improvement (see Tab. 3.1). Furthermore, there was a clear qualitative improvement in samples from FlowScan.



(a) FlowScan



(b) Set-Coupling



(c) BRUNO

Figure 3.9: Single digit set samples from FlowScan, Set-Coupling, and BRUNO trained on `MNIST`. Each row corresponds to a single set of 20 images generated by one model. Sets generated from BRUNO often contain unidentifiable or multiple digits whereas FlowScan samples are relatively homogeneous and representative of the training set.

CHAPTER 4: DEFENSE THROUGH DIVERSE DIRECTIONS

In this chapter, we utilize learned distributions over network parameters to construct a Bayesian neural network and demonstrate strong adversarial robustness without the need for expensive online adversarial training. Unlike previous efforts in this direction, we do not rely solely on the stochasticity of network weights by minimizing the divergence between the learned parameter distribution and a prior. Instead, we additionally require that the model maintain some expected uncertainty with respect to all input covariates. We demonstrate that by encouraging the network to distribute evenly across inputs, the network becomes less susceptible to localized, brittle features which imparts a natural robustness to targeted perturbations. We show empirical robustness on several benchmark datasets.

4.1 Overview

Neural networks currently achieve greater-than-human performance in a variety of tasks such as object recognition [76], language understanding [41, 168], and game playing [154, 156]. Despite their incredible successes, these same networks are easily fooled by seemingly-trivial perturbations that humans overcome with minimal difficulty [**?**]. This weakness poses considerable concern in a world that is increasingly reliant on machines from the perspectives of both security (*e.g.*, face recognition) and safety (driverless cars). Despite considerable effort to overcome these difficulties, the problem persists [8, 53].

The most successful methods for improving adversarial robustness utilize online adversarial training [116]. Online adversarial training requires an iterative training procedure where adversarial examples are produced based on a particular attack scheme with respect to the current network state and the model is updated to resist the particular attack. Unfortunately, this method is compu-

tationally expensive, as attacks must be generated and the model updated multiple times. Zhang et al. and Sharma and Chen have demonstrated that while this process makes the model robust to the particular type of attack used in the training process, the model can be susceptible to attacks from alternate schemes.

To scale adversarial training, researchers have tried to transfer adversarial examples from another model. Tramèr et al. [160] has found that this offline adversarial training scheme can perform equally well in practice but can be much more efficient since it decouples the adversarial examples generation from the training process.

Alternative lines of research introduce randomness into the model. Early attempts include adding Gaussian noise to the inputs [186] and randomly pruning the network [173, 42]. Liu et al. propose to add Gaussian noise to all the intermediate activations. Wang et al. train multiple copies for each block of the network and randomly select one during inference. Along these lines, we utilize Bayesian neural networks (BNNs) as a principled way to inject noise into the model.

Recent work [112, 113] has incorporated stochasticity by utilizing BNNs. Similar to our method, they demonstrate that randomness of BNNs alone is not sufficient for robust classification. They turn to an online adversarial training scheme to implicitly boost the randomness.

We instead choose to explicitly penalize the model so that it evenly distributes the sensitivity of the output w.r.t. the input elements. We estimate the output sensitivity for each input through a first order Taylor approximation and exploit the inherent ensembling of BNNs to evaluate statistics for each input example. These statistics become the basis for a defense-promoting regularization scheme by diversifying the directions of the output. This chapter is adapted from our original work, "Defense through Diverse Directions," [13].

Our contributions are as follows:

- We propose several general penalties that can be added to the loss function of any Bayesian neural network to diversify the output variation with respect to the input covariates.
- We demonstrate that increased output diversity leads to natural adversarial robustness, without requiring online adversarial training.

- We show that models trained with our diversity inducing penalties generalize to a variety of attack schemes.

This work begins with a review of Bayesian neural networks and the adversarial problem. We then discuss our motivations and methods for improving model robustness. Finally, we illustrate our methods on several datasets and discuss the implications. Particularly, we show that our method improves robustness over state-of-the-art BNN methods [113], all without online adversarial training.

4.2 Background & Related Work

In this section we provide an overview of background material and related work.

4.2.1 Bayesian Neural Networks

In the context of supervised deep learning, a conventional neural network seeks to perform some variant of a classical functional estimation task, to learn a *point estimate* of the optimal function in the chosen functional space that maps each input from the input space to its corresponding output in the output space. However, such an estimate does not consider, and thus cannot effectively adapt to, the inherent uncertainty throughout the training procedure (*e.g.*, data collection, random initialization of network weights). Such deficiency leads to problems including over-fitting and overly confident predictions.

To remedy this deficiency, a *Bayesian neural network* (BNN), introduced in the same vein as continuous stochastic processes such as the Gaussian process, seeks to directly model the distribution, whose density we denote as p, over random functions

$$f \sim p(f), \quad f : \mathbb{X} \mapsto \mathbb{O}$$

where $\mathbb{X}$ and $\mathbb{O}$ denote the input and the output spaces, respectively. However, directly learning such a distribution can be arduous as functional spaces are usually infinite-dimensional. Utilizing

the fact that neural networks can be regarded as universal approximators for functions [84], a distribution over random functions can be thought of as a choice over of neural networks. We materialize such connection through learning a distribution over the network weights. More specifically, in assuming that the network weights are random and distributed according to a prior distribution $P(\mathbf{w})$, a BNN seeks to learn the posterior distribution of the network weights, $\mathbf{w}$, given the available data, *i.e.* $P(\mathbf{w}|D)$.

While a clever idea, learning $P(\mathbf{w}|D)$ is prohibitively expensive even for moderately sized networks since utilizing Bayes rule would require integrating over all the probabilistic parameters in the network:

$$p(\mathbf{w}|D) = \frac{p(D|\mathbf{w})p(\mathbf{w})}{p(D)}$$
$$p(D) = \int_{\mathbf{w}} p(D|\mathbf{w})p(\mathbf{w})d\mathbf{w}$$

Borrowing ideas from variational inference and the recent success of unsupervised methods like the *variational auto-encoder* [94], Blundell et al. propose to learn a *variational posterior distribution*, $q(\mathbf{w}|\theta)$, to approximate the true posterior by optimizing the following objective

$$\max_{\theta} \ \mathbb{E}_{q(\mathbf{w}|\theta)} \left(\log P((D|\mathbf{w})) - \mathrm{KL}\left(q(\mathbf{w}|\theta)||p(\mathbf{w})\right)\right) \tag{4.1}$$

which is the *evidence lower bound* (ELBO) for the data likelihood. The expectation term ensures the learnt variational posterior distribution is informed by the data, and the KL divergence acts as a regularizer over the weights. Although technically any choice of the variational posterior and prior distributions pair is possible, the convention, which we adopt in this work, is to choose both to be independent Gaussians where we learn the mean and variances of the variational posterior distribution. One benefit of deploying a BNN, which we exploit in our proposed framework, is that for one input one can *draw* multiple functions f, which in practice is actualized by drawing a set of different weights from the posterior distribution $P(\mathbf{w}|D)$, to form an ensemble of

inferences for various purposes, such as assessing the uncertainties in predictions, gradient evaluations, etc. In this work, we exploit the network's variation to control how much sensitivity we expect from each input element.

4.2.2 Adversarial Attack

Adversarial examples are constructed by making small perturbations to the input that induce a dramatic change in the output. Attacks are typically broken down into two categories: white and black box attacks. The exact definition of both methods vary, but we will use the following definition in this work. In the white box setting, the attack has access to the training data set, the fully specified underlying model, and the loss functions. Attacks are found by performing gradient ascent with respect to the input. In the black box setting, the attacker has access to the training data and the loss functions but does not have access to the underlying model parameters. A black box attack can then be constructed by using a stand-in network trained on the same data with the same loss. Previous works have shown that these examples are still effective against a variety of other models [114, 161].

The attacker's goal is:

$$\max_{\|\varepsilon\|_p < \varepsilon_{\max}} \mathbb{E}\left[\mathcal{L}\left(f(\mathbf{x}+\varepsilon;\theta),\mathbf{y}\right)\right] \tag{4.2}$$

where ε is the attack perturbation, p is the norm (typically taken to be ∞), $\varepsilon_{\max}$ is the attack budget (the maximum perturbation), $\mathcal{L}$ is the loss, f is the network, $\mathbf{x}$ is the input, θ is the network parameters, and $\mathbf{y}$ is the truth.

Typically the attack methods generate the adversarial examples by leveraging the gradient of the loss function with respect to the inputs. The Fast Gradient Sign Method (FGSM) [64], for instance, takes one step along the gradient direction to perturb an input by the amount ε:

$$x_{adv} = x + \varepsilon \cdot \text{sign}(\nabla_x \mathcal{L}(f(\mathbf{x};\theta),\mathbf{y})). \tag{4.3}$$

Projected Gradient Descent (PGD) [116] generalizes FGSM by taking multiple gradient updates:

$$x_k = x_{k-1} + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(f(\mathbf{x};\theta), \mathbf{y})), \tag{4.4}$$

where α is the step size. After each update, PGD projects the perturbed inputs back into the ε-ball of the normal inputs.

There are also other types of attacks, such as C&W attack [25], Jacobian Saliency Map Attack (JSMA) [136] and DeepFool [121]. Among all the attack methods, PGD is regarded as the strongest attack in terms of the L_∞ norm.

4.2.3 Adversarial Defense

The goal of adversarial defense is to render all (bounded) perturbations ineffective against a model. It is necessary to bound the perturbation or the attack could simply replace an input with an example from a different distribution or class. A simple intuition to achieve this would be to require that the model be Lipschitz smooth such that

$$\|f(\mathbf{x} + \varepsilon; \theta) - f(\mathbf{x}; \theta)\| < C \|\varepsilon\| . \tag{4.5}$$

Unfortunately, estimating the Lipschitz coefficient C for an arbitrary network can be extremely difficult, making optimizing over it nontrivial. Cissé et al. has attempted to control the Lipschitz coefficient of each layer in the network and, therefore, the network as a whole.

Recent works have introduced a number of defense methods, such as distillation [136], label smoothing [74], input denoising [158], feature denoising [179], gradient regularization [147], and preprocessing based approaches [38, 70, 24]. Most of these defenses have unfortunately been defeated by subsequent attacks.

The most popular adversarial defense technique incorporates adversarial examples in the training process [116]. Online adversarial training requires generating new examples throughout the training process to map out the local region around known examples and require that the

region map to the expected output. Unfortunately, while this approach does produce a robust defense, it is computationally expensive.

Pang et al. utilize a diversity promotion scheme across several, independently-trained, non-Bayesian networks. They promote diversity by encouraging the distribution of the probabilities across all classes, excluding the true class, should be different for each independent model. Our method could be considered a generalization that uses a diversity promotion penalty that is not unique to the classification problem and that uses a stochastic ensembling scheme instead of a deterministic scheme. The stochastic scheme allows access to a potentially unlimited number of models instead of the fixed number chosen at the outset of the training process. Several other methods utilize different forms of ensembling to improve robustness, *e.g.*, [152].

Most similar to our work, ADV-BNN [113] attempts to use BNN to combat adversarial attack. Their proposed method is dependent on online adversarial training and aims to incorporate it into the standard ELBO objective, Eq. 4.1, as a min-max problem. In contrast, our proposed methodology does not require online adversarial training, and achieves better performance on CIFAR-10 compared to ADV-BNN.

4.2.3.1 Obfuscated Gradients

Athalye et al. [8] warn of a failure mode in defense methods that they term "obfuscated gradients," where seemingly high adversarial accuracy is only superficial. Networks that achieve apparent improvements in white box attacks through obfuscated gradients do so by making it difficult for an attacker to find ε. However, successful ε still exist, which indicates that the network has not increased adversarial accuracy despite its improved adversarial test accuracy. Distinguishing between whether a defense mechanism has increased adversarial accuracy or merely increased attack difficulty is nontrivial. Typical methods rely on comparing white-box and black-box attacks. A strong indication that a defense is obfuscating gradients is when black-box attacks are successful while white-box attacks fail (low black-box accuracy and high white-box accu-

racy). A defense with high-white box accuracy and fair black-box accuracy is indicative of a mixture of obfuscated gradients and substantive adversarial accuracy improvement.

4.3 Motivation

Our primary motivation comes from the observation that the fewer inputs an attack needs to perturb, the less robust a model is. Therefore, we wish to diversify the importance of each input to the output. Alternatively, we wish to reduce the sensitivity of the output to variations in each input, possibly weighted by some foreknowledge of input uncertainty.

Since adversarial examples are best known for image recognition due to the dramatic difference in human robustness versus machine robustness, we attempt to provide some intuition in this setting. For the general image setting, the object of interest may exist anywhere in image, and there is no way to know ahead of time which pixels are more reliable. Therefore, we assume that, on average, the sensitivity due to any single pixel should be roughly equal. So, we estimate the sensitivity per pixel, normalize across pixels, and penalize any divergence from our expectation.

We can estimate the sensitivity of the mth output, δy_m, with respect to the expected sensitivity of the dth input, δx_d, through a truncated Taylor series

$$\delta y_{m,d}(\mathbf{x}) \equiv y_m(\mathbf{x} - \delta\mathbf{x}_d) - y_m(\mathbf{x}) \tag{4.6}$$

$$\approx \delta\mathbf{x}_d^T \frac{\partial y_m(\mathbf{x})}{\partial \mathbf{x}} \;=\; \delta x_d \frac{\partial y_m(\mathbf{x})}{\partial x_d} \tag{4.7}$$

and collecting across inputs

$$\delta\mathbf{y}_m(\mathbf{x}) \approx \delta\mathbf{x} \odot \nabla y_m(\mathbf{x}) \tag{4.8}$$

where $\delta\mathbf{y}_m$ is a length D vector corresponding to the sensitivity of the mth output with respect to each input. For simplicity, we assume there is only one output and drop the dependence on m.

There are several ways to normalize the result across inputs. We choose to normalize by the L_2 norm. For datasets where each input is equitably important/reliable, $\delta x_d = \delta x$ and the

normalized result becomes

$$\mathbf{u}_y(\mathbf{x}) = \delta\mathbf{y}/||\delta\mathbf{y}||_2 = \nabla y/\left\|\nabla y\right\|_2 \tag{4.9}$$

where $\mathbf{u}_y(\mathbf{x})$ is the direction of the gradient of the the output with respect to the input, $\mathbf{x}$.

This means that we expect the direction of the output gradient to be uniformly distributed over the unit hypersphere. In terms of the loss surface, all directions become equally likely,

$$\mathbf{u}_y \sim U_{\text{sph}}(\mathbf{u}). \tag{4.10}$$

In the event where the input uncertainty varies, the distribution would be uniformly distributed over a hyper-ellipsoid.

4.4 Method

Typically, neural networks are supervised to map an input to an output. We use the approximation from Sec. 5.2 to further supervise the uncertainty of the output. However, all networks have some inherent uncertainty (*e.g.*, from random initializations) that changes the expected sensitivity. We execute K draws of the BNN and include additional penalties that attempt to maintain the expected distribution of the output sensitivity. We experiment with a variety of penalties based on this premise. For the sake of brevity, we drop the dependence of $\mathbf{u}_y$ on $\mathbf{x}$.

4.4.1 Entropy and Variances

As motivated previously, in order to increase the network's robustness against adversarially perturbed input, we encourage the network to maintain, and hence to evenly distribute, some expected sensitivity with respect to all input covariates. We materialize this idea by encouraging the normalized gradient in Eq. 4.9 to be uniformly distributed over the unit hypersphere, which in turn is equivalent to maximizing the entropy of $\mathbf{u}_y$, denoted as $H(\mathbf{u}_y)$, because $\mathbf{u}_y$ is bounded

within the unit hypersphere. However, as a function of the network weights $\mathbf{w}$, the density of $\mathbf{u}_y$ is intractable even for moderate-sized network, making maximizing $H(\mathbf{u}_y)$ directly prohibitively expensive and impractical in practice.

In a simplified setting where the elements are independent, maximizing the sum of the variances of each element is equivalent to maximizing the entropy of the random vector.

Proposition 3. *Given a random vector $\boldsymbol{X} = (X_1, X_2, \cdots, X_D)^T$ where its elements are independent and $X_i \in [a_i, b_i]$ for all i, there exists a monotonically increasing relationship between the entropy of $\boldsymbol{X}$, $H(\boldsymbol{X})$, and the sum of the variances of each element of $\boldsymbol{X}$, $\sum_i \mathrm{Var}(X_i)$.*

Proposition 1 indicates that maximizing the entropy of $\mathbf{X}$ is equivalent to maximizing $\sum_i \mathrm{Var}(X_i)$. See Appendix A for the proof. While $\mathbf{u}_y$ is not independent in our case, we use Prop. 1 as an analogy and adopt the sum of the variances of the elements of $\mathbf{u}_y$ as a surrogate for $H(\mathbf{u}_y)$.

4.4.2 Direct Loss

A simple method might be to maximize the sum over inputs of the variance of $\mathbf{u}_y$ across the K draws

$$\sum_{d=1}^{D} \mathrm{Var}\left[u_{y,d}\right]. \tag{4.11}$$

However, since $\mathbf{u}_y$ is a unit vector, this loss degenerates and only serves to minimize the average value of the output sensitivity

$$\sum_{d=1}^{D} \mathrm{Var}\left[u_{y,d}\right] = \mathbb{E}\left[\sum_{d=1}^{D} u_{y,d}^2\right] - \sum_{d=1}^{D} \mathbb{E}\left[u_{y,d}\right]^2 \tag{4.12}$$

$$= 1 - \sum_{d=1}^{D} \mathbb{E}\left[u_{y,d}\right]^2. \tag{4.13}$$

Since we wish to maximize the variances w.r.t. $\mathbf{u}_y(\mathbf{x})$ we consider the mean penalty $\Omega_M(x)$:

$$\Omega_M(\mathbf{x}) = \sum_{d=1}^{D} \mathbb{E}\left[u_{y,d}\right]^2. \tag{4.14}$$

4.4.3 MinVar

While Eq. 4.14 increases the total variance across inputs, it does not necessarily encourage diversity. We consider several additional penalties to include with Eq. 4.14 that do increase diversity.

A simple penalty to increase the minimum variance over dimensions is

$$\Omega_{V,1}(\mathbf{x}) = -\min_{d} \operatorname{Var}\left[u_{y,d}\right]. \tag{4.15}$$

Equation 4.15 exploits the fact that the sum (over dimensions) of the variance is fixed. Therefore, increasing the minimum necessarily decreases the other values. In theory, as the network trains, the minimum element changes and eventually all the elements converge to the same variance.

Unfortunately, since the loss only supervises one pixel at a time, the minimum shifts across a few elements and never influences the pixels with the largest variance. We consider two simple methods to correct this difficulty. One is to supervise all the pixels simultaneously by replacing the $\min$ operation with a soft-min weighted sum so that Eq. 4.15 becomes

$$\Omega_{V,2}(\mathbf{x}) = -\sum_{d=1}^{D} \text{softmin}(\alpha u_{y,d}) \cdot \operatorname{Var}\left[u_{y,d}\right] \tag{4.16}$$

where α corresponds to the temperature. However, since the sum of the variances is fixed and this loss attempts to increase the variance of all pixels simultaneously, we find that it can result in a counterproductive competition across pixels.

A more direct method to supervise the variance is to minimize the distance between the observed variance and the expected variance of $1/N$. Using the Euclidean distance, Eq. 4.15 becomes

$$\Omega_{V,3}(\mathbf{x}) = \sum_{d=1}^{D} \left(\operatorname{Var}\left[u_{y,d}\right] - 1/D\right)^2. \tag{4.17}$$

We find that this final representation yields the best results amongst the variance encouraging losses and choose it as the variance penalty $\Omega_V(\mathbf{x})$.

4.4.4 Non-Sparse Promoting Losses

Directly encouraging the model to match a specific distribution's second order moment may be too strict a requirement. As an alternative to matching the second moment of the uniform hypersphere, we consider penalizing *small* L_1 norms of $\mathbf{u}_y$. By requiring that the L_1 norm be large, we bias the network away from over reliance on a few features and encourage greater diversity across the input covariates. Unlike in the variance-based losses, the network does not have to match each input direction to the same value. This allows for increased flexibility while still maintaining the same intuitive effect on the diversity of dependence. The added penalty becomes

$$\Omega_S(x) = -\mathbb{E}\left[\|\mathbf{u}_y\|_1\right]. \tag{4.18}$$

4.4.5 Batch Loss

We summarize the full possible loss of our model with respect to posterior parameters, $\mathcal{L}(\theta)$, with all the above penalties, a supervised loss l and a batch of data $\{(x_n, y_n)\}_{n=1}^N$:

$$\begin{aligned}\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\Big[&l(x_n, y_n) + \lambda_M\Omega_M(x_n) \\ &+\lambda_V\Omega_V(x_n) + \lambda_S\Omega_S(x_n)\Big].\end{aligned} \tag{4.19}$$

We vary Eq. 4.19 below, by setting various penalty weights λ to zero.

4.4.6 Further Benefits of Adversarial Training

Finally, we test the benefits of offline adversarial training [160] in addition to the diversity inducing penalties. Adversarial examples are pre-computed from a trained, non-Bayesian neural network of a matching architecture and then added statically to the training set. Thus, this form of defense is more efficient than online adversarial training, which requires on-the-fly computation of adversarial examples.

4.5 Experiments

In this section we explore the effect of inducing diversity on a variety of open-source datasets.

4.5.1 Penalty Shorthand

We present our results by adding different combinations of the diversity-encouraging penalties in Sec. 6.3. To declutter the results, we use the following shorthand to refer to the different penalties we apply to the model. The unit gradient mean penalty, Eq. 4.14, is given as "M;" the variance penalty, Eq. 4.17, by "V;" the non-sparse penalty, Eq. 4.18, by "S;" and any offline training by "Off." We additionally denote when a network is Bayesian by prepending the network name with a "B." For example, the case where we use a Bayesian VGG16 with the mean and variance penalty is shorthanded as "BVGG16-M-V."

As mentioned previously, all adversarial examples appended to the training set for offline training were constructed using conventional networks with matching architectures. Aside from the models trained with offline adversaries, we do not include any form of data augmentation.

4.5.2 Practical Considerations

In this section we discuss several practical steps taken to implement various networks and diversity-promoting penalties. We utilize TensorFlow [1] and Tensorflow Probability [43] to implement the general and probabilistic components of our models, respectively. During training, the classification loss is assessed per network draw and then averaged. During inference and attack, ensembling is performed by averaging the logits over draws.

4.5.2.1 Drawing from the Bayesian Network

Since we optimize over statistics of the probabilistic network, we require multiple network samples for the same input data. A simple method to exploit extant parallelisms in most neural network frameworks would be to duplicate each element in the batch K (the number of draws)

times. Unfortunately, Bayesian networks implement the Bayesian layers using either FlipOut [175] or the reparameterization trick [94] to efficiently draw parameters that are *shared* across each element in the batch. While this shortcut is sufficient to decorrelate training gradients, it is not sufficient to obtain the level of independence required to inform our methods. As a result, we resort to executing the network K times for *each batch*.

This becomes prohibitively expensive for networks of sufficient depth. To offset some of this cost, we break our networks into two parts. In the first part, the layers are constructed in the conventional fashion without Bayesian components. We will refer to this component of our models as the deterministic network. After the deterministic network, we construct a Bayesian network. The deterministic component is executed once per batch and the Bayesian component, K times. The gradient that informs diversity promotion is still taken with respect to the input of the full network. This means that while the deterministic component does not directly add variation it is still trained to encourage overall network diversity. The specifics of where the transition between determinism and Bayesian can be found in each experiment's section. In general, we found that it was sufficient to set the last quarter of the network as Bayesian and use $K = 10$ draws.

4.5.2.2 Attack Schemes

We test our defense against two types of common attacks: the Fast-Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD). We deploy both L_∞ and L_2 attacks. Black box attacks are performed using examples from external sources when available. All attacks are performed using the Adversarial Robustness Toolbox [130] and use the same number of network draws as are used in training.

4.6 Synthetic Dataset

We consider the synthetic dataset constructed by (Tsipras et al., 2018) In this dataset, adversarial examples are constructed analytically (without gradients) such that one feature is adversarially robust but all others are not.

Tsipras et al. prove that any classifier that achieves arbitrarily high standard accuracy on this dataset necessarily has poor adversarial accuracy. See the original work for a detailed discussion.

The dataset is constructed as a binary classification problem over y with input features $\mathbf{x}$ such that

$$\begin{aligned} y &\sim \{-1, +1\} \\ x_1 &= \begin{cases} +y, & \text{w.p. } p \\ -y, & \text{w.p. } 1-p \end{cases} \\ x_2, ..., x_D &\sim \mathcal{N}\left(\eta y, 1\right), \end{aligned}$$

for the standard dataset. Adversarial examples are constructed by sampling $x_2, ..., x_D$ from a distribution that is inversely correlated with the label y, *i.e.*

$$x_2, ..., x_D \sim \mathcal{N}\left(-\eta y, 1\right). \tag{4.20}$$

To better match this toy dataset to our assumptions, we construct the data (including adversarial examples) as described and then orthonormally project the features into a new space. We construct the orthonormal matrix, $\mathbf{W}$, by choosing the ones vector for the first column and the remaining columns as any orthogonal space and then normalizing, *i.e.*, for the d-th column vector, w_d, in W, $\|w_d\| = 1$, $w_d \cdot w_k = 0$ for $k \neq d$, and $w_0 = [1, 1, ..., 1]/\sqrt{D}$.

It is easy to show that the bounds given in [162] hold through this process and that the importance of features in this new space are more evenly distributed by considering that, in the original

formulation, the adversarially-robust weight matrix, v, corresponds to a the standard basis vector in the first dimension and calculating how this vector is modified when the data is transformed by W. If we let $\mathbf{v_u}$ be the new robust classification weights, W be the orthonormal transformation matrix, and u be the transformed space, then

$$\begin{aligned}
\mathbf{u} &= \mathbf{W}\mathbf{x} \\
y &= \mathbf{v}^T\mathbf{x} \\
&= \mathbf{v}^T\mathbf{W}^T\mathbf{W}\mathbf{x} \\
&= \mathbf{v}^T\mathbf{W}^T\mathbf{u} \\
\mathbf{v_u} &= \mathbf{W}\mathbf{v}.
\end{aligned}$$

And since $\mathbf{v}$ is the standard basis for the first dimension, $\mathbf{v_u} = \mathbf{w}_1$, which was selected as the ones vector.

The choice of W can be generalized to any invertible matrix to achieve the same bounds. We choose to use a constant vector for the first column as it best matches our expectations for

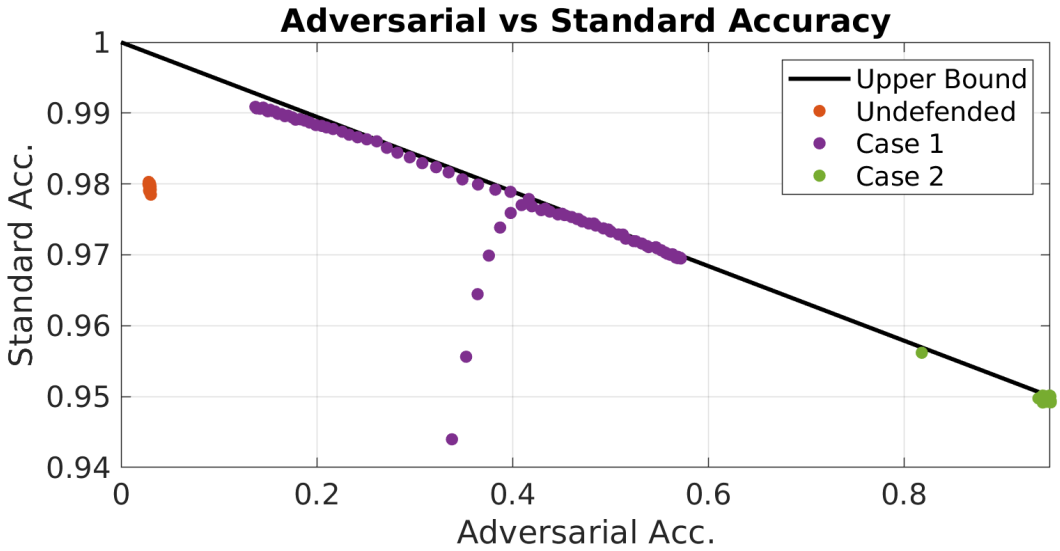


Figure 4.1: Standard and adversarial accuracy for various models compared to the upper bound. Each dot represents the model performance at the end of different epochs across the training process. Note, some negative jitter was introduced to Case 2 to enhance visualization.

practical data. In our experiment, we set $p = 0.95$ and $\eta = \frac{2}{\sqrt{D-1}}$. A robust classifier, which only depends on x_1, would achieve 95% standard and adversarial accuracy whereas a classifier which depends on all covaraites can achieve arbitrarility good accuracy but with low adversarial accuracy. (see Eq. 4 in (Tsipras et al., 2018)).

We train a simple model on this dataset and test the standard and adversarial accuracies throughout the training procedure. Figure 4.1 illustrates the standard and adversarial accuracy with respect to the adversarial upper bound for three different Bayesian networks: with no defense, with $M = 100, V = 120, S = 10$ (Case 1), and with $M = 0, V = 0, S = 40$ (Case 2). Points in the figure are collected at different epochs to assess how the adversarial accuracy compares to the upper bound throughout the training process.

As expected, the Bayesian network achieves strong standard accuracy with poor adversarial accuracy and does not even achieve the adversarial upper bound. Both models penalized by our methods consistently achieve the upper bound and influence the model so that it is less prone to achieving arbitrary accuracy. Case 1 showcases good standard accuracy and strong adversarial accuracy, corresponding to the theoretical upper bound, after a warm up period. Case 2 achieves the maximum adversarial accuracy possible. Additional details can be found in Fig. 4.3 and Fig. 4.2.

These two cases are representative of the behavior of a variety of different hyperparameter choices. We observe that our penalties cause the model to either achieve and sustain maximum adversarial accuracy or increase adversarial accuracy to a point before decaying to arbitrary standard accuracy.

We take this as encouragement that, since adversarial examples in this setting are constructed analytically (without model gradients), this indicates that our method is capable of creating general adversarial robustness and does not simply obfuscate gradients. The tendency towards the upper bound is striking because we have not directly informed the model about the adversarial problem — there is no adversarial training we only require a diverse ensembling based on inherent properties of Bayesian networks and some diversity encouraging penalties.

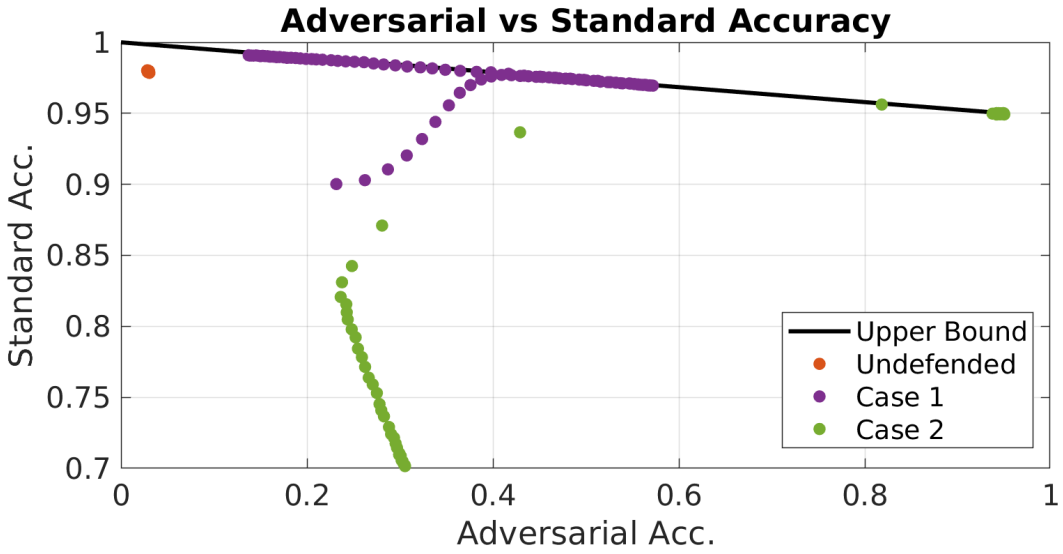


Figure 4.2: Standard and adversarial accuracy for various models compared to the upper bound with increased plotting range. Each dot represents the model performance at the end of different epochs across the training process. Note, some negative jitter was introduced to Case 2 to enhance visualization.

Figure 4.3 provides additional context of the accuracies over the training process. As expected, the undefended model converges rapidly to a high standard accuracy but with adversarial robustness well below the upper bound. In the first case, once the model has achieved the robust, standard accuracy, the adversarial accuracy gradually increases before it ultimately peaks and degrades. While this degradation is unfortunate, it is noteworthy that the adversarial accuracy stays near the upper bound throughout the decay process. In the other case, the model appears to undergo a phase change and quickly converges to and maintains the maximum possible adversarial accuracy.

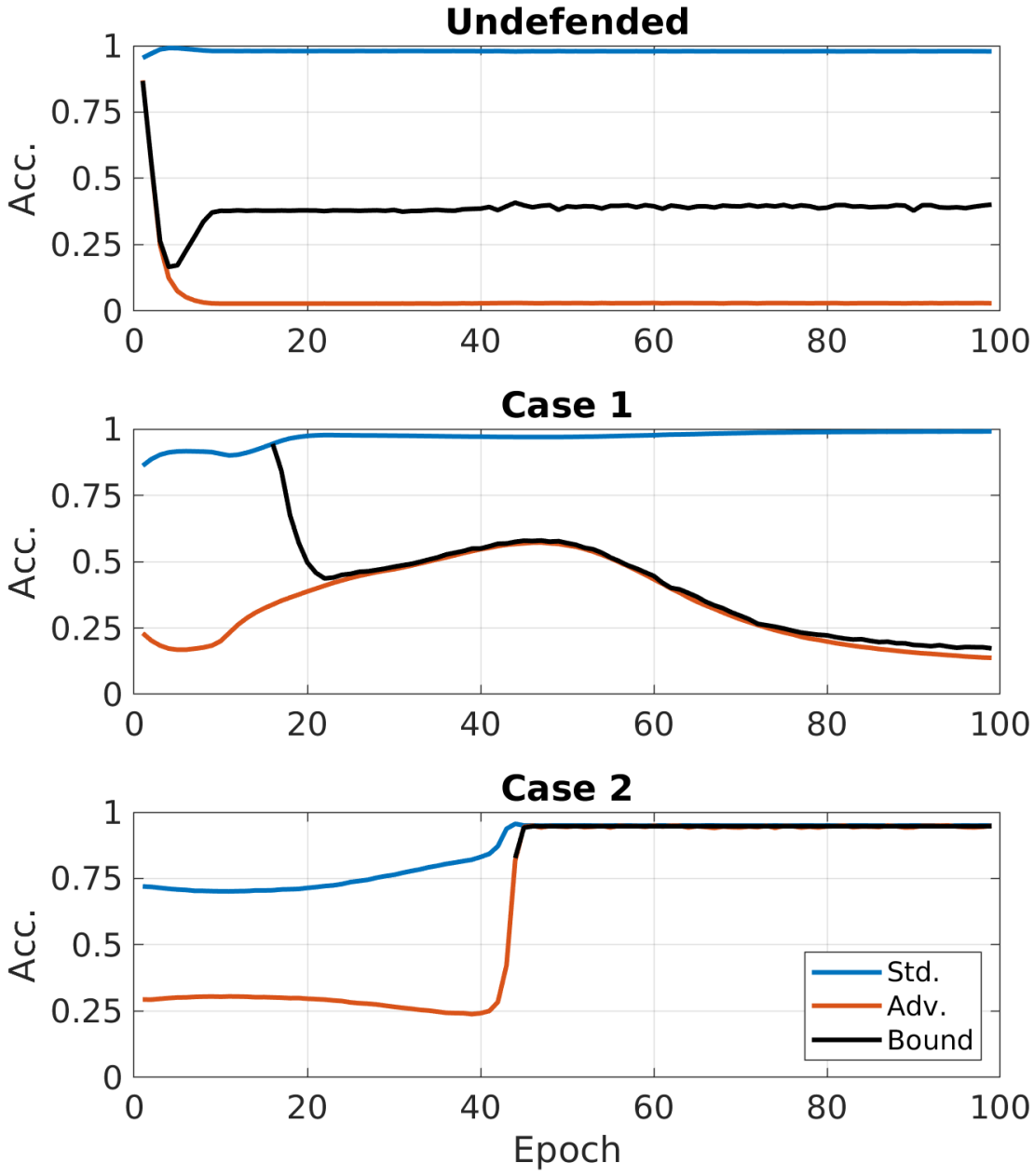


Figure 4.3: Standard and adversarial accuracy evolution for the test set with various models.

Table 4.1: Adversarial accuracy (%) on MNIST with various combinations of diversity promotion against L_∞ attacks.

Method	FGSM	PGD	Black-Box
CNN	36.8	2.6	57.8
BNN	55.2	0.9	56.4
BNN-Off	40.0	2.1	**93.6**
M	93.0	55.2	60.1
M-V	94.0	70.0	61.8
M-S	96.3	94.0	54.3
M-V-S	97.2	95.8	59.8
M-S-Off	**97.6**	**95.8**	89.9
M-V-S-Off	97.4	94.5	90.6

4.6.1 MNIST

We test our diversity induced models on MNIST [105]. For these tests we use a simple CNN with three convolutional layers followed by two fully connected layers. Since this network is fairly shallow, we compose the deterministic part of the network using the first two convolutional layers and use Bayesian layers for the final convolutional and both fully connected layers. When the corresponding penalty is used, the loss hyperparameters are: $\lambda_M = 20$, $\lambda_V = 40$, and $\lambda_S = 40$.

Table 4.1 shows the accuracy of the model when trained using different combinations of diversity inducing penalties and adversarial training against L_∞ attacks with a maximum attack budget of 0.3. All models obtain better than 99% standard accuracy. Black box attacks were obtained using examples from an open repository (https://github.com/MadryLab/mnist_challenge).

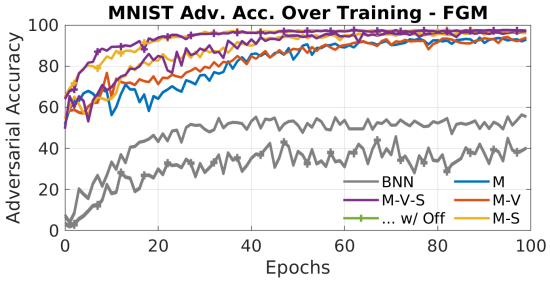
When only the mean penalty (Eq. 4.14) is imposed, the model shows modest improvements in both forms of attack; indicating that the defense has succeeded in improving the robustness of the model against attack. While this defense has not completely succeeded in overcoming attacks, it is a positive step. As we include additional penalties, we observe that the white-box attack improves, most notably with the inclusion of the non-sparse penalty (Sec. 4.4.4). However, these models generally show a slight *decrease* in black-box performance. We infer that these penalties tend to cause the model to over fit the defense by obfuscating gradients. Fortunately,

offline adversarial training appears to sufficiently augment the training set so that the defenses can generalize.

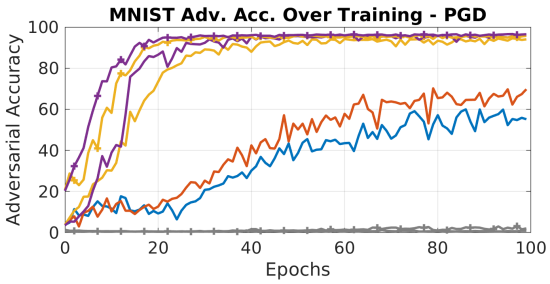
4.6.1.1 MNIST Accuracy Evolution

Figures 4.4a 4.4b illustrates how the adversarial accuracy evolves over the training process for FGSM and PGD, respectively. Other attack evolutions can be found in Fig. B.1. We chose to train for 100 epochs to give the defenses adequate opportunity to influence the model. Models trained with diversity promoting penalties are given in color and models without are given in gray-scale. Models supplemented with offline adversarially training have "+" symbols in addition to the matched colors to identify their training penalties. In both cases, attacks are generated using the *test* set against the current model state. These figures provide some insight into how the defense is instilled in the model as it is trained. The mean and variances penalties appear to slowly (but consistently) influence the model throughout the training process. These penalties seem to still be improving the adversarial accuracy against PGD attacks even after 100 epochs have elapsed, well after the standard accuracy has converged.

As speculated in Sec. 4.4.4, the non-sparse penalty appears to give the model greater flexibility and is easier to learn. This quick increase and the disparity between white and black box performance in Table 4.1 may indicate that the penalty is prone to causing the defense to over fit by obfuscating gradients. However, this is assuaged with the use of offline adversarial training. All models converged to a standard accuracy of 99% within the first two epochs.



(a) FGSM L_∞ $\varepsilon = 0.3$



(b) PGD L_∞ $\varepsilon = 0.3$

Figure 4.4: Evolving white box attack accuracy after each epoch.

Table 4.2: Adversarial accuracy (%) under L_∞ white and black box attacks on CIFAR-10 with various methods of diversity promotion.

Loss	Std.	FGSM	PGD	Black-Box
VGG16	**90.4**	14.4	5.9	58.0
BVGG16	89.1	12.9	5.6	24.5
BVGG16-Off	85.8	58.8	57.0	84.3
M	90.1	14.2	6.1	64.1
M-V	87.1	19.6	10.2	55.5
M-S	88.4	50.2	47.7	55.2
M-V-S	88.7	56.6	60.2	55.1
M-S-Off	86.3	73.1	**72.3**	**84.8**
M-V-S-Off	85.7	**73.8**	63.8	83.4

4.6.2 CIFAR-10

In this section, we evaluate our proposed diversity inducing penalties on the CIFAR-10 dataset [100]. The backbone network is VGG16. A Bayesian version of VGG16 is built by replacing the last convolutional block and the fully connected layers with variational alternatives. When training with our proposed penalties, we use the hyparameters: $\lambda_M = 10$ and $\lambda_S = 20$. We report the L_∞ attack with a budget of $\varepsilon = 0.03$ here. PGD attack perform 40 gradient updates with a step size 0.001. Refer to Sec. 4.7 for ablation experiments on the attack budget. We report the black box attack accuracy using examples from an open repository.[1]

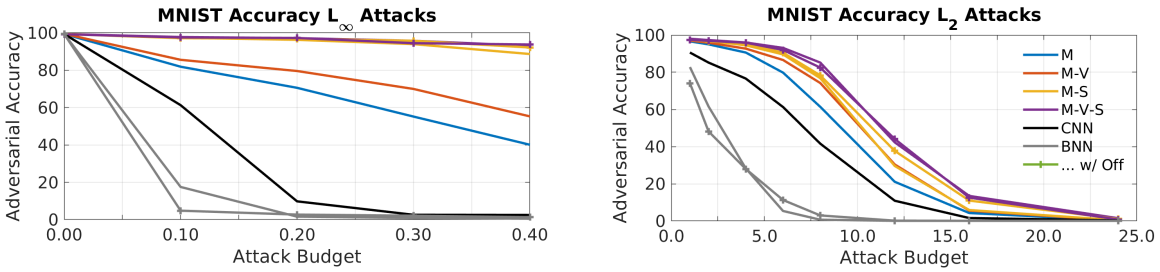
Table 4.2 demonstrates the accuracy of the defended models on CIFAR-10. The standard loss is included in the table since it varies across models. Given the marginal improvements in adversarial accuracy from the variance-based penalties, we forego their use in this case.

Unlike in the MNIST experiments, the mean penalty is insufficient to overcome white box attacks; however, it does offer improvements in the black box setting. Also unlike MNIST, these results do not indicate that our methods suffer from the obfuscated gradients problem as white box attacks are consistently more effective than black box attacks. The additional data augmentation from offline adversarial training further improves defense.

[1] https://github.com/MadryLab/cifar10_challenge

Table 4.3: Comparison of accuracy (%) under PGD attack with different budget. Results for "Adv-CNN" and "Adv-BNN," representing adversarially trained CNN and BNN, are from [113].

Method/Budget	0.0	0.015	0.035	0.055	0.07
Adv-CNN	80.3	58.3	31.1	15.5	10.3
Adv-BNN	79.7	68.7	45.4	26.9	18.6
M-S	**88.4**	61.3	47.8	39.8	34.3
BVGG16-Off	85.8	58.2	57.8	57.3	57.3
M-S-Off	86.3	**73.2**	**73.1**	**73.0**	**73.0**



(a) L_∞ attack budgets.

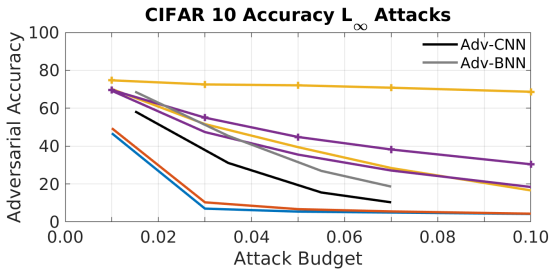
(b) L_2 attack budgets.

Figure 4.5: Varying attack budgets with 40 step PGD attacks against MNIST defenses.

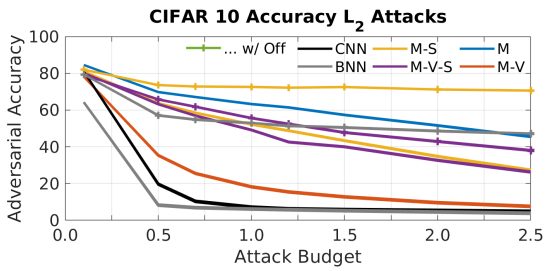
Table 4.3 juxtaposes several of our proposed defenses against results reported in Liu et al. for several attack budgets. We observe that our method without any adversarial training maintains higher standard accuracy and shows improved robustness to higher attacks budgets. Surprisingly, we observe that a Bayesian VGG16 with offline adversarial training obtains consistent accuracy above the other methods. We suspect this is because the adversarial examples used in offline training were constructed using PGD with 40 steps whereas the examples in ADV-BNN were defended using online training with 10 steps. Including our penalties consistently increases the performance of the offline model by approximately 15%.

4.7 Ablation

To test our defenses' efficacy under more diverse attack cases, we vary the PGD attack budget against models trained on MNIST and CIFAR-10. Figure 4.5 illustrates results from tests on MNIST and Fig. 4.6 from tests on CIFAR-10. Colors and symbols follow the same conventions as in Fig. B.1.



(a) L_∞ attack budgets.



(b) L_2 attack budgets.

Figure 4.6: Varying attack budgets with 40 step PGD attacks against CIFAR-10 defenses.

Figures 4.5a and 4.6a demonstrates how the various models respond to increases in the attack budget of PGD L_∞ attacks. For L_∞ attacks against CIFAR-10, we compare against results reported in [113] instead of a generic CNN or BNN as in other cases. Unsurprisingly, all the models show decreases in performance as the attack budget is increased. The best model consistently utilizes the mean and non-sparse penalized models with offline adversarial training. The inclusion of the variance penalty shows a slight improvement on MNIST but significantly worse performance on CIFAR-10. Against MNIST, the diversity penalties dramatically increase adversarial accuracy over offline training and likewise improve CIFAR-10 accuracy by approximately 15%. The online adversarially trained models outperform diversity-based models that do not include the non-sparse penalty, but are consistently out-performed by models that do include the penalty.

Figures 4.5b and 4.6b uses L_2 attacks but otherwise demonstrates the same variation as in Figs. 4.5b and 4.6a (changes in the attack budget of PGD). The most interesting feature is how well the mean penalty performs without any additional augmentations. It begins with the highest accuracy and is consistently higher than the model trained with the mean and non-sparse penalties and is primarily outperformed only by the full mean and non-sparse penalized models with offline training. Otherwise, the results are consistent with those found in the L_∞ study.

4.8 Discussion and Limitations

We have demonstrated the efficacy of explicitly encouraging diversity of the output with respect to the input. On MNIST, we show that we can obtain strong adversarial robustness without the need for any form of adversarial training. In this case, our black box accuracy falls short of the white box accuracy, indicating we may be obfuscating gradients. Fortunately, the black box performance is still fair, which may mean that our method improves model robustness and obfuscates gradients. Including offline adversarial training in these models improves the black box accuracy so that it is on par with the white box accuracy. We suspect that the original MNIST training set is not diverse enough itself and that additional data or augmentations may be sufficient to prevent the defense from over fitting and obfuscating gradients.

Our results on CIFAR-10 are quite encouraging. We demonstrate that our method is capable of improving adversarial accuracy with only a small reduction in standard accuracy. These models do not appear to suffer from the obfuscated gradients problem: black box accuracy is consistently higher than white box performance. Further, the ablation studies show a consistent reduction in accuracy as attack strength is increased. Finally, our model compares favorably with other Bayesian defense mechanisms, achieving superior performance in most cases without any adversarial training. The added use of offline adversarial training improves our models' performance so that they are superior in all cases.

We speculate that it may be possible to further improve our defense results by including additional forms of standard augmentation, *e.g.*,the addition of Gaussian noise, shifting, etc. Similarly, we performed only a small search for the hyperparameters of the diversity penalties, λ, and used them for all penalty combinations and regardless of the use of offline training. Additional gains may be possible by performing a finer-grained search over these parameters.

4.9 Conclusions

In this chapter, we explored how to utilize learned distributions over neural networks by learning distributions over network parameters to improve adversarial robustness. This is a daunting task, where models that utilize the concept of randomness to combat adversarial attack were previously limited to adding random noise layers to the network [112] or simply applying a Bayesian neural network to take advantage of the stochasticity of the weights [113]. To the best of our knowledge, this is the first attempt to attain adversarial robustness through explicitly requiring the network to maintain and evenly distribute expected and sensible uncertainty with respect to input covariates, achieved by the various penalty terms we introduce. Without the need for online adversarial training, we demonstrate the effectiveness and robustness of our approach by achieving the strong adversarial (after-attack) accuracies on various datasets against different adversarial attacks. This effort provides concrete evidence that we can improve the performance of machine learning algorithms by taking advantage of learned distributions.

CHAPTER 5: PRACTICAL INTEGRATION

We expand our ability to compute integrals (expectations) over deep learning models by exploiting the approximate density learned by a normalizing flow model (Sec.2.2.2) when composed with a separable function. This allows us to generalize the training procedure from only evaluating on a finite collection of augmented points to training on continuous hypervolumes. We explore how to construct architectures and continuous penalties to control/regularize model behavior in global and local regions.

5.1 Overview

Most supervised learning problems operate by training a model on a finite collection, $\mathcal{T}$, of N (typically paired) examples, (x, y). The model is updated by comparing its predicted output to the expected output and performing some flavor of stochastic gradient descent based on the comparison and various regularizers. The process is repeated by reexamining the elements of $\mathcal{T}$ in a random order, possibly with augmentation, until the model parameters converge or an iteration budget is exceeded. This relatively simple procedure has proven to be remarkably effective in a variety of domains and these models have begun to permeate every aspect of modern science and everyday life [23, 77, 155].

The deep learning revolution has also resulted in highly effective generative models such as VAEs [92], GANs [63], and tractable likelihood models [47, 67, 131]. These models are largely used to create novel samples of impressive quality. In addition to sampling, likelihood models provide an estimate of the probability density function of the data which can be used for additional, downstream processes.

We *augment the training process* by constructing neural networks that allow for tractable integration over the *input domain*. This differs from implicit layers which utilize integration over a *parameterized* variable [29, 67]. Access to fast and differentiable integrals allows us to regularize a model's *average* behavior using metrics that may not be available otherwise. Integration over the input space also allows us to supervise how the model behaves in *continuous regions that are not directly observed* in $\mathcal{T}$ and may even be out-of-distribution (OOD).

Alternative methods attempt to supervise examples outside of $\mathcal{T}$ by performing random perturbations [71], along the line between known examples [189], or via a generative process [6, 187]. However, these methods are only capable of observing a small quantity of the total space. By integrating over entire regions, it is possible to observe a large portion of the space based on statistical relevance. This chapter is adapted from our original work, "Practical Integration via Bijective Networks," [15].

Main Contributions We propose a general architecture that enables tractable integration over the input space, enabling supervision and custom regularization over continuous regions. We demonstrate how to construct this network and how it allows for a reduction in the computation cost required for dense numeric integration from exponential in the number of dimensions to linear. We derive several useful integrated formulations over continuous regions. Finally, we explore the impact of this architecture and regularizers on the standard accuracy and robustness to OOD examples on several standard classification datasets. The code utilized in this paper can be found at `https://github.com/lupalab/sep_bij_nets`.

Notation Throughout this work, we consider the M dimensional input features, $\mathbf{x} = [x_1, ..., x_M] \in \mathbb{R}^M$; the latent features, $\mathbf{z} \in \mathcal{Z} \subseteq \mathbb{R}^M$; and the K-wise classification probability, y. The input features $\mathbf{x}$ in the training set, $\mathcal{T}_x$, are a random subset of the in-distribution data, $\mathcal{D} \subseteq \mathbb{R}^M$. Subscripts represent a particular dimension, *e.g.*, $\mathcal{D}_m$ corresponds to the m^{th} dimension of the space. Paranthetical superscripts represent the subspace corresponding to a particular class, *e.g.*, $\mathcal{D}^{(c)}$ is the subset of $\mathcal{D}$ where the data belongs to class c. The bijective network is given as h such that $h : \mathcal{D} \to \mathcal{Z}$. Probability distributions over $\mathcal{D}$ and $\mathcal{Z}$ are given by p with the corresponding sub-

script. Classification networks are given as f and g. Variables with a "hat," $\hat{y}$, are predictions of the true quantity, y.

5.2 Motivation

Neural networks are highly effective function approximators between two (typically) continuous spaces: $f : \mathcal{X} \rightarrow \mathcal{Y}$. However, networks are typically trained and evaluated using a finite collection of *points* without any explicit assessment of the complete hypervolume spanned by the data. This omission is understandable from an implementation perspective as the number of samples required to obtain a reasonable estimate over a volume scales exponentially with data dimensionality. However, human beings often have an understanding of how a process should behave *on average*. Ideally, we would like to embed this intuition into the model but currently cannot assess the average performance of a trained model outside of the held-out test set. Specifically, we would like to regularize the model by estimating the expected behavior of some metric, Ω, produced by the model over the training data

$$\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\Omega(\hat{y}(\mathbf{x}))\right] = \int_{\mathcal{X}} \Omega(\hat{y}(\mathbf{x}))p(\mathbf{x})d\mathbf{x}. \tag{5.1}$$

There are many useful choices of Ω over a variety of applications. If it is known what the model output should be on average ($\bar{y}$), we can construct Ω to encourage that behavior, *e.g.*, $\Omega(\hat{y}) = (\bar{y} - \hat{y})^2$. Minimizing consistency metrics [181] are a common method to improve learning in label-starved problems. These encourage the model to produce similar outputs over neighborhoods around (labeled) examples from $\mathcal{T}_x$ where neighborhoods are created by random or domain-specific augmentations. This process can be viewed as an approximation to an integral over the neighborhood,

$$\mathbb{E}_{\epsilon\sim p(\epsilon)}\left[\mathcal{L}(y, \hat{y}(\mathbf{x}+\epsilon))\right] = \int \mathcal{L}(y, \hat{y}(\mathbf{x}+\epsilon))p(\epsilon)d\epsilon \tag{5.2}$$

where $\mathcal{L}$ is a distance-like metric, and ϵ is the neighborhood. Equation 5.2 can be generalized to other neighborhoods. We can recast the standard classification problem as a discrete approxi-

mation to an integral. Typically, we minimize the cross-entropy between $\hat{y}(\mathbf{x};\theta)$ and y over the model parameters, θ, for all $(\mathbf{x}, y) \in \mathcal{T}$ which becomes an integral over class-conditioned distributions, $p(\mathbf{x}|c)$,

$$\min_{\theta} -\sum_{x,y\in\mathcal{T}} \sum_{k} y_k \log\left(\hat{y}_k(x;\theta)\right) \Rightarrow \min_{\theta} -\sum_{k} \int_{\mathcal{D}^{(k)}} y_k \log\left(\hat{y}_k(x;\theta)\right) p(\mathbf{x}|k) d\mathbf{x}. \tag{5.3}$$

Equation 5.3 can be simplified when y_k is a one-hot vector by collapsing the sum over k to retain only the term corresponding to the (single) true class. We choose not to perform this simplification to allow for more general cases where there may be some class ambiguity within the volume, such as when label smoothing is in use.

Unfortunately, integration in high dimension is difficult. Naive gridded solutions require an exponential number of points with error decreasing as $O(G^{-M})$, for G, M-dimensional points. Monte Carlo (MC) methods theoretically have better performance with error that decreases as $O(G^{-1/2})$. However, the rate of convergence for MC methods depends on the variance of samples [169], which may make for poor approximations in practice. Importance sampling [19] can improve the performance of Monte Carlo methods to adapt to the regions with the greatest contribution.

We choose to model the data using a separable function. Separable functions have the key benefit of decomposing M-dimensional integrals into a combination of M one-dimensional integrals. Each of these one-dimensional integrals can then be solved using any number of highly-accurate solvers (*e.g.*, Runge-Kutta [151], Dormand-Prince [48], Tsit [164], etc.) that have error rates better than $O(G^{-4})$ but are unavailable in high dimensions. The use of separable functions is a component of the VEGAS algorithm [140] and is utilized in conjunction with adaptive Monte Carlo sampling to approximate high-dimensional integrals.

The use of a separable function over the input space may make estimation of integrals over the model more accessible; however, they impose a strong, inappropriate inductive-bias. The obvious approach of utilizing a standard neural network as a feature extractor and integrating

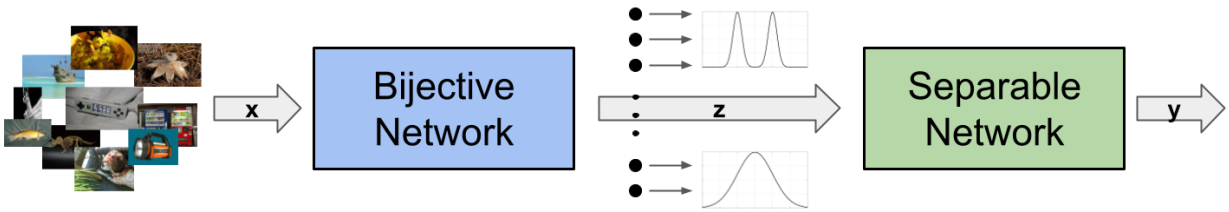


Figure 5.1: Depiction of the overall network with intervening distributions over the latent space, $\mathcal{Z}$

over learned features means that we would no longer have a valid integrator over the input space. We propose to solve this problem by utilizing bijective transforms prior to the separable network. The bijective transform allows us to decouple the data into a latent space where the data can be modeled using a separable network and guarantees equality between integrals in the latent space and integrals in the input space.

5.3 Background

We perform integration over the input space by splitting neural network models down into two key components: (1) a *bijective* feature extractor, (2) a *separable* task network, see Fig. 5.1. For simplicity, we only consider classification tasks in this work. This makes our total network analogous with the common architecture where a classifier, often a linear layer or an MLP, is constructed on a feature extractor, such as a CNN. Unlike the typical process, we must place constraints on both networks so that we can integrate over the input domain. This network breakdown is similar to hybrid networks [30, 126] except for the separability requirement on the classifier.

5.3.1 Separable Functions

Separable functions have long been used in mathematics and physics to solve simple partial differential equations such as the homogeneous wave and diffusion equations [159]. We consider two types of separable functions, additive and multiplicative. All proofs can be found in Appendix A.

5.3.1.1 Additively Separable Functions

Definition 5.3.1. A function, $f : \mathbb{C}^M \rightarrow \mathbb{C}^K$, is additively separable if it is composed as a summation of element-wise functions operating independently on each dimension of the input:

$$f(\mathbf{v}) = \sum_{m=1}^{M} f_m(v_m; \phi_m) \tag{5.4}$$

Theorem 5.3.1 (Additive Independent Integration)**.** *Given an additively separable function, $f(\mathbf{v})$, an independent likelihood, $p(\mathbf{v}) = \prod_{m=1}^{M} p_m(v_m)$, and domain, $\mathcal{D}_v = \mathcal{D}_{v_1} \times ... \times \mathcal{D}_{v_M}$:*

$$\mathbb{E}_{\mathbf{v}\sim p(\mathbf{v})}[f(\mathbf{v})] = \sum_{m=1}^{M} \int_{\mathcal{D}_{v_m}} f_m(v_m)\, p_m(v_m)\, dv_m \tag{5.5}$$

5.3.1.2 Multiplicatively Separable Functions

Definition 5.3.2. A function, $g : \mathbb{C}^M \rightarrow \mathbb{C}^K$, is multiplicatively separable if it is composed as a product of element-wise functions operating independently on each dimension of the input:

$$g(\mathbf{v}) = \prod_{m=1}^{M} g_m(v_m; \psi_m) \tag{5.6}$$

Theorem 5.3.2 (Multiplicative Independent Integration)**.** *Given a multiplicitively separable function, $g(\mathbf{v})$, an independent likelihood, $p(\mathbf{v}) = \prod_{m=1}^{M} p_m(v_m)$, and domain, $\mathcal{D}_v = \mathcal{D}_{v_1} \times ... \times \mathcal{D}_{v_M}$:*

$$\mathbb{E}_{\mathbf{v}\sim p(\mathbf{v})}[g(\mathbf{v})] = \prod_{m=1}^{M} \int_{\mathcal{D}_{v_m}} g_m(v_m)\, p_m(v_m)\, dv_m \tag{5.7}$$

5.4 Method

Both forms of separable functions allow us to decompose a single M-dimensional integral into M 1-dimensional integrals. A dense estimation of the integral without taking advantage of a separable function using G points per dimension would require $\mathcal{O}(G^M)$ network evaluations.

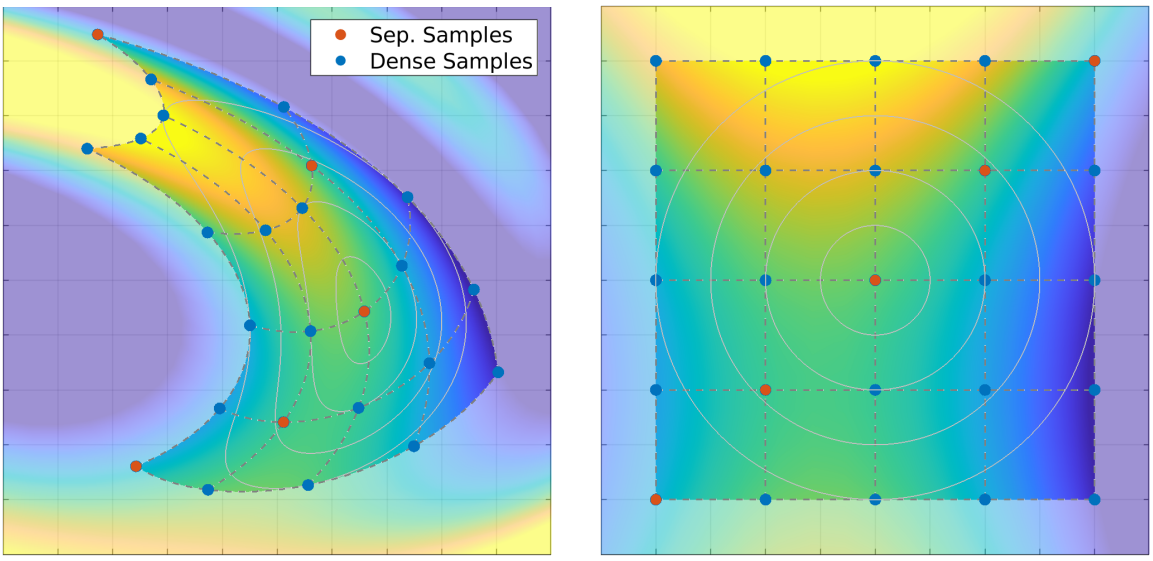
This is completely impractical for modern datasets where M is at least on the order of hundreds and G should be as large as possible. Exploiting separable functions allow us to reduce the complexity to $\mathcal{O}(GM)$.

However, it is unlikely that we could construct a separable function directly on the input space and achieve reasonable performance. Doing so would essentially require that each input dimension contribute to the final output independently of the others. Instead, we can combine Eq. 2.8 and Eq. 5.5 (alternatively, Eq. 5.7) to perform practical integration over the input domain while still allowing for inter-dependent contributions from each input dimension. To do so, we first let the latent distribution, p_Z, be independently (but not necessarily identically) distributed: $p_Z(z) = \prod_m p_m(z_m)$. This allows us to write the integral over the input space in terms of the latent space and then simplify via Eq. 5.5.

$$\begin{aligned}\int_{\mathcal{D}} f(h(\mathbf{x}))p_Z(h(\mathbf{x}))\left|\frac{\partial h}{\partial \mathbf{x}}\right| d\mathbf{x} &= \mathbb{E}_{\mathbf{x}\sim p_X(\mathbf{x})}\left[f(h(\mathbf{x}))\right] \\ &= \mathbb{E}_{\mathbf{z}\sim p_Z(\mathbf{z})}\left[f(\mathbf{z})\right] = \int_{\mathcal{Z}} p_Z(\mathbf{z})f(\mathbf{z})d\mathbf{z} = \sum_{m=1}^{M}\int_{\mathcal{Z}_m} f_m(z_m)\, p_m\left(z_m\right) dz_m\end{aligned} \tag{5.8}$$

Each 1-dimensional integral can be easily approximated using a variety of integration approaches. In addition to the requirements placed on the feature extractor and task network, this formulation also requires that we define the integration domain in the latent space as a Cartesian product of domains over each latent dimension. This may seem like a strong requirement since we apparently lose some level of interpretability; however, there are several advantages to defining the input domain in this learned space. Most notably, we know the data distribution in this space exactly and can tune the domain based on the goal of a particular integral.

Figure 5.2 contains a cartoon example that demonstrates how these methods combine to allow for integration over the complex data space. Both plots show the value of f; Fig. 5.2a shows the non-separable function in the input space and Fig. 5.2b shows the same data in the (separable) latent space, after the bijective transformation. The colored points and dotted lines are in correspondence between the two spaces and illustrate how the space is warped by the bijector



(a) Input Space $f(h(\mathbf{x}))$ (b) Latent Space $f(\mathbf{z})$

Figure 5.2: Separable functions need $O(G)$ latent samples (red) instead of $O(G^M)$ input samples (blue). Integration regions emphasized.

to create a separable, independent latent space. The light gray lines represent the contours of the underlying probability distribution. We define both the integration domain and perform the integration in the latent space. For simplicity, we illustrate the integration using five, equally-spaced points (in both dimensions). The naive procedure would require sampling f at each point in the grid (the blue points). The use of the separable functions allow us to integrate using only the red points and get the same estimate.

5.5 Practical Applications

In this section we discuss specific choices made when constructing the latent distribution, separable networks, and various integrals. For classification tasks, it is not possible to parameterize the normalized network output of class probabilities, $\hat{y}$, as a separable network since each prediction must sum to one. However, it is possible to parameterize each unnormalized logits as a separable network, *e.g.*, $\hat{y}(\mathbf{x}) = \sigma\left(f(h(\mathbf{x}))\right)$, where σ is the softmax operator and the output of f are the unnormalized logits. While this limits what quantities can be integrated over exactly, it is still possible to integrate over approximations/bounds without resorting to brute-force methods.

Out-of-Distribution Detection We construct a global integration regularizer to provide resilience to out-of-distribution (OOD) examples. We enable OOD detection by introducing a "reject" option [78] into the classification vector, increasing the number of classes by one, where the additional class indicates that the instance is OOD. An example is considered OOD if $\hat{y}_{K+1}$ exceeds a predefined threshold. We supervise the model over out-of-distribution examples by integrating the cross-entropy over the contrastive probability distribution, $q(\mathbf{z})$ (see Sec. 5.5.3.1).

Semi-supervised Learning We build a local integral to enhance consistency within latent neighborhoods and utilize it in place of pre-chosen augmentation strategies to inform a label-starved dataset. Specifically, we introduce a loss where all examples near a real, labeled example are regularized to share the real example's label as in Eq. 5.2 (see Sec. 5.5.3.2). We additionally use pseudo-labels [107] so that consistency is maintained about unlabelled points as the model grows more confident.

5.5.1 Latent Distributions

A critical component of this method is that the likelihoods over the latent space must be independent: $p_Z(\mathbf{z}) = \prod_m p_Z(z_m)$. As this is extremely unlikely to occur in the input space, we learn a bijective (flow) transformation from the input space to a latent space where the likelihood can be decomposed into independent components of our choice. Since we would like to use the latent features to discriminate between classes using a separable function, we choose to utilize a (bimodal) Gaussian mixture model. This allows the model to put different classes into one of two "buckets" per feature and enables classification through multiple features. This results in an exponential number of components (with respect to dimension) in the full latent space, *e.g.*, 2^M components. However, we can easily offset this by choosing some dimensions to

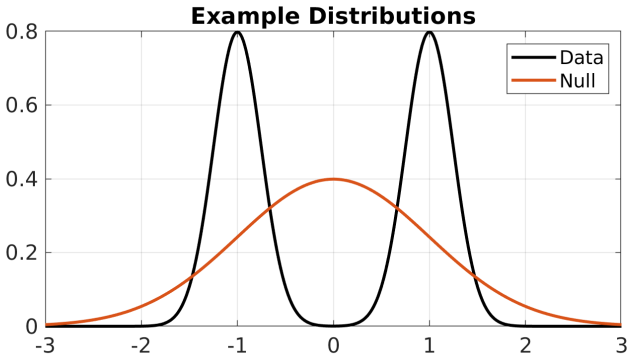


Figure 5.3: Data and contrastive distributions

use a unimodal latent distribution while the remaining dimensions are still bimodal:

$$p_{Z_m}(z_m) = \begin{cases} 0.5\,\mathcal{N}(-\mu, \sigma_1^2) + 0.5\,\mathcal{N}(\mu, \sigma_1^2), & \text{if } m \leq L \\ \mathcal{N}(0, \sigma_2^2), & \text{else} \end{cases} \tag{5.9}$$

In addition to the typical latent data distribution, $p_Z(\mathbf{z})$, we explicitly include a contrastive distribution, $q(\mathbf{z})$. This distribution is critical if we desire to regularize our model outside the data distribution, *i.e.*, setting some background/default behavior. When using a bimodal data distribution, we additionally desire that latent vectors are more likely under this distribution once the vector is sufficiently far away from any single in-distribution mode. Therefore, we select this distribution so that it fully encapsulates all the components of the data distribution, *e.g.*, $q_m(z_m) > p_{Z_m}(z_m)$ for $|\,|z_m| - \mu| > \gamma$. When the data distribution is unimodal we choose $p_m = q_m$ so that the feature is uninformative.

We utilize a standard Gaussian for the contrastive distribution and set μ to 1 and experiment with $\sigma_1 \leq 0.5$. This choice allows us to formulate OOD examples that exist between the in-distribution data as near zero, and outside the in-distribution data, near the tails. Figure 5.3 illustrates the data and contrastive distributions for a latent feature for convenience.

5.5.2 Separable Networks

We explore three different formulations of separable networks. The first formulation learns a distinct quadratic function, the second learns a symmetric hinge, and the third learns a multi-layer perceptron. These functions have distinct parameterizations for each latent feature and each in-distribution class. The out-of-distribution logit is held fixed at zero: $l_{K+1}(\mathbf{x}) = 0$.

$$f_{k,m}^{(\text{quad})}(z_m; \alpha_{k,m}, u_{k,m}, \nu_{k,m}) = \alpha_{k,m} - \frac{(z_m - u_{k,m})^2}{2e^{2\nu_{k,m}}} \tag{5.10}$$

$$f_{k,m}^{(\text{hinge})}(z_m; \alpha_{k,m}, u_{k,m}, \nu_{k,m}) = \alpha_{k,m} - |z_m - u_{k,m}|\, e^{\nu_{k,m}} \tag{5.11}$$

where k is the logit class index and m is the feature dimension so that the total unnormalized logit, $l_k(\mathbf{z})$, is $l_k(\mathbf{z}) = \sum_m f_{k,m}(z_m)$. The separable MLP is constructed by concatenating a learned vector per feature and class to each 1-dimensional feature and constructing a standard MLP that operates on that augmented state. This formulation provides the greatest flexibility as it leverages all the benefits of the universal approximator theorem [83]. Unfortunately, we find the MLP version difficult to train in practice, often resulting in degenerate solutions. Fixing the OOD logit at zero provides a fixed reference point for the other logits. We interpret this as each latent feature voting on how in-distribution the example is on a per-class basis.

5.5.3 Integrals

The cross-entropy, $\text{CE}(\hat{y}(\mathbf{x}), y)$, between the normalized model output, $\hat{y}(\mathbf{x})$, and the true labels, y, is commonly used to train classification models. In this section, we explore global and local integrals of the cross-entropy loss used to train most classifiers (see Eq. 5.3) when $\hat{y}$ is given as the composition of the softmax function, a separable function and a bijective function, *e.g.*, $\sigma(f(h(\mathbf{x})))$. We can express this as $\min_\theta \sum_{\mathbf{x},y\in\mathcal{T}} \text{CE}(\hat{y}(\mathbf{x}), y)$ owing to the finite number of elements in $\mathcal{T}$. Ideally, we would minimize the model cross-entropy over all possible examples in $\mathcal{D}$. We accomplish this by defining the subspace, $\mathcal{D}^{(c)} \subset \mathcal{D}$, that has a given label and corresponding conditional distribution, $p(\mathbf{x}|c)$

$$\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|c)}\left[\text{CE}\left(\hat{y}(\mathbf{x}), \mathbf{c}\right)\right] = -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|c)}\left[\log(\hat{y}_c(\mathbf{x}))\right] = -\int_{\mathcal{D}^{(c)}} \log(\hat{y}_c(\mathbf{x}))\, p(\mathbf{x}|c) d\mathbf{x}. \tag{5.12}$$

It is not possible to represent $\hat{y}(\mathbf{x})$ as separable network due to the softmax operation but it is possible to parameterize the unnormalized logits as a separable network. Substituting this parameterization into Eq. 5.12 and transforming to the latent space yields a bound for the expected

cross-entropy

$$\begin{aligned}\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|c)}\left[\mathrm{CE}\left(\hat{y}(\mathbf{x}), y\right)\right] \leq &-\sum_{m=1}^{M}\int_{\mathcal{Z}^{(c)}} f_{c,m}(z_m)p_m(z_m|c)dz_m \\ &+\log\left(\sum_{j=1}^{K+1}\prod_{n=1}^{M}\int_{\mathcal{Z}^{(c)}}\exp\left(f_{j,n}(z_n)\right)p_n(z_n|c)dz_n\right).\end{aligned} \tag{5.13}$$

See Appendix E for a full derivation. We will utilize this formulation in conjunction with a contrastive prior to supervise OOD examples, Sec. 5.5.3.1, and with a local prior to enforce consistency, Sec. 5.5.3.2.

5.5.3.1 Out-of-Distribution Supervision

We can utilize the cross-entropy integral in different ways by choosing the label we would like to apply over a domain. For example, we can integrate the cross-entropy over the OOD latent space, $\mathcal{U}$, with the true label fixed to the reject class ($c{=}K{+}1$), and the latent distribution over codes is the contrastive distribution (Sec. 5.5.1), $p(z|c{=}K{+}1) = q(z)$; using Eq. 5.13 with $f_{K+1,n}{=}0$:

$$\mathcal{L}_{\text{GLBL}} = \log\left(\sum_{j=1}^{K}\prod_{n=1}^{M}\int_{\mathcal{U}}\exp\left(f_{j,n}(z_n)\right)q_n(z_n)dz_n\right). \tag{5.14}$$

In effect, Eq. 5.14 discourages OOD data from being labeled as any in-distribution label. Since the data and contrastive distributions overlap, the model could degenerate and always make OOD decisions; however, this would be in conflict with the standard cross-entropy loss applied to each example in the training set. So long as $\mathcal{L}_{\text{GLBL}}$ is not weighted too strongly, the model achieves equilibrium by making OOD predictions over regions where the contrastive distribution is more likely. In effect, we supervise OOD training without requiring any OOD data.

5.5.3.2 Local Consistency

We may also perform integration over neighborhoods of data points in $\mathcal{T}_x$ and utilize that point's label over the entire region. This integral serves to improve consistency around observa-

tions. Adversarial attacks [64, 116] are often constructed as perturbations on real data points and adversarial training finds one such point and includes it in the training set. We can interpret local consistency integrations as a form of *average* adversarial training. We reformulate the integral to be centered around a particular datum, $\mathbf{x}_0$ and its class label, y

$$\mathcal{L}_{\text{LCL}} = -\sum_k y_k \int_{\mathcal{V}} \log(\sigma\left(f_k(h(\mathbf{x}_0) + \mathbf{v})\right)) \, p_V(\mathbf{v}) d\mathbf{v} \tag{5.15}$$

A complication resulting from local integration is how to select the local distribution, $p_V(\mathbf{v})$, and the neighborhood, $\mathcal{V}$. There are many reasonable choices depending on the goal of the local integration. For simplicity, we choose each $\mathcal{V}_m \in [-\varepsilon, \varepsilon]$ and $p_m(v_m)$ to be uniformly distributed.

5.5.4 Loss Components

The final training loss used to evaluate the model is composed of different combinations of the standard cross-entropy loss over class predictions, $\mathcal{L}_{\text{CE}} = -\sum_k y_k \log\left(\hat{y}\right)$, the negative log-likelihood loss over in-distribution data points, $\mathcal{L}_{\text{NLL}} = -\log(p_{\mathcal{Z}}(h(\mathbf{x}))) - \log\left|\frac{\partial h}{\partial \mathbf{x}}\right|$, and the integration losses, $\mathcal{L}_{\text{GLBL}}$ and $\mathcal{L}_{\text{LCL}}$. We always include $\mathcal{L}_{\text{CE}}$ and $\mathcal{L}_{\text{NLL}}$. Based on results from previous hybrid networks [30, 126], we weight the negative log-likelihood by $1/M$, which is analogous to using bits per dimension and introduce an additional weight over the cross-entropy, λ, which controls the relative importance of the generative and classification tasks. When included, each integration penalty shares the same weight as $\mathcal{L}_{\text{CE}}$ with additional binary weight, π

$$\mathcal{L}_{\text{total}} = \frac{1}{M}\mathcal{L}_{\text{NLL}} + \lambda\left(\mathcal{L}_{\text{CE}} + \pi_{\text{GLBL}}\mathcal{L}_{\text{GLBL}} + \pi_{\text{LCL}}\mathcal{L}_{\text{LCL}}\right). \tag{5.16}$$

5.6 Related Work

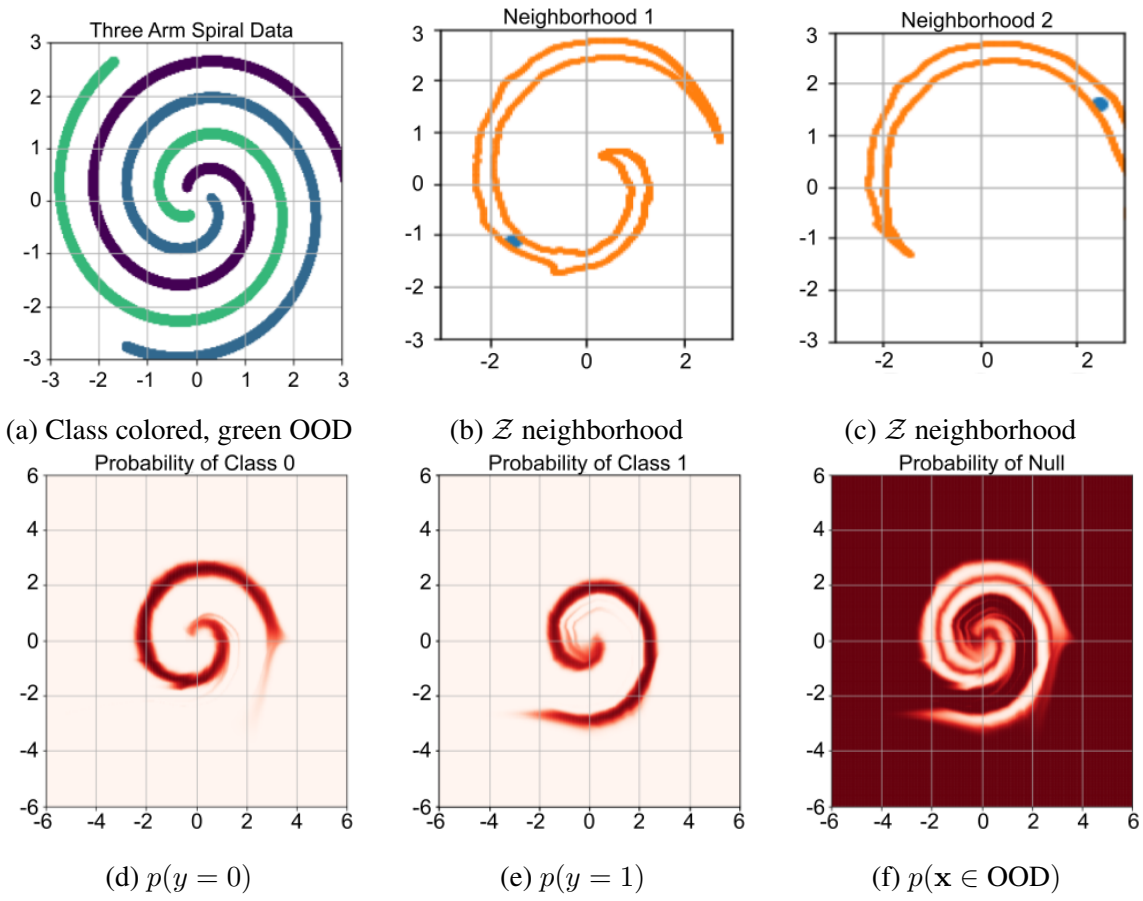
Noise Contrastive Distributions Noise contrastive methods introduce a distribution that is distinct from the data distribution. The constrastive distribution can be learned and provides a mechanism to supervise the model to discriminate between true and contrastive samples [71]. This

forces the model to learn discriminative statistical properties of the data. We utilize this idea to inform the null-space over which we will integrate. We fix the data and contrastive distributions and learn the bijective map between input and latent spaces.

Hybrid Models Normalizing flows are a family of flexible, tractable likelihood estimators based on the application of learnable, bijective functions [47, 67]. These methods have shown strong performance, both as likelihood estimators and as generative processes. Recent work has coupled advances in bijective networks to construct invertible feature extractors that are capped by a more conventional classifier. The combination bijective/classifier structure are called *hybrid models* and show reasonable performance as likelihood and discriminative models [30, 126]. Unfortunately, the discriminative performance of these models is lower than traditional methods. We utilize hybrid models with constraints that enable tractable integration. Other hybrid models architectures are less restricted but prohibits practical integration.

Out of Distribution Detection OOD detection is a challenge for many modern ML algorithms. Recent work has demonstrated that OOD data can achieve higher likelihoods than the training data [80, 125]. Several recent methods have been developed to detect OOD examples including Maximum Softmax Probability (MSP) [79], Outlier Exposure (OE) [80], Multiscale Score Matching (MSMA) [117], ODIN [111], and Certified Certain Uncertainty (CCU) [119]. [109] utilize a GAN trained with a classifier to produce examples near but outside the training distribution. These methods are constructed specifically for OOD detection whereas our method is applicable to a variety of problems.

Semi-supervised Learning Semi-supervised learning is a burgeoning research area that learns from a large data corpus when only a small subset are labeled. The problem setting is very pertinent as it is often easy to acquire data examples but can be extremely time consuming to create the corresponding labels. Many modern methods can achieve very strong performance with very few labels [189, 33]. However, most of these methods rely on domain-specific augmentation strategies that are difficult to replicate in new data regimes, *i.e.*, for non-image data. Fortunately,



(a) Class colored, green OOD (b) $\mathcal{Z}$ neighborhood (c) $\mathcal{Z}$ neighborhood

(d) $p(y = 0)$ (e) $p(y = 1)$ (f) $p(\mathbf{x} \in \text{OOD})$

Figure 5.4: Three-arm spirals results

methods such as SSVAE [95] and VIME [182] are domain agnostic. These methods are built exclusively for semi-supervised learning but we only require an additional regularizing penalty to achieve comparable performance on tabular datasets.

5.7 Experiments

All models were constructed using PyTorch [137], trained using PyTorch-Lightning [55], utilized bijectors and distributions from Pyro [18], and were trained using Adam [93]. We assess the integrable model's performance in a semi-supervised regime and against OOD examples. See Appendix B and C for additional experiments and training details.

5.7.1 Spirals

We construct a synthetic, 2D dataset composed of three intertwined spirals, see Fig. 5.4a. Suppose that, due to an unknown sampling bias, we only observe two of the three arms (missing the green arm) and constructed our model as a two-class problem with a third, reject class.

We sample points in a dense 2-dimensional grid at twice the range of the data in both dimensions and evaluate the probability that each point belongs to the two known classes and the OOD class. Figures 5.4d and 5.4e illustrate the probability of belonging to the two known classes and Fig. 5.4f contains the probability that each point is OOD. Red and white values indicate high and low probabilities, respectively. The model successfully identifies the two in-distribution regions. Unsurprisingly, the model also identifies data outside of the training data range as being OOD and, impressively, identifies the unobserved spiral and the region between spirals as being OOD.

Finally, we sample a square box (orange) around a data point (blue) in the latent space and invert the box to assess the appropriateness of the neighborhood in the input space and plot the result, Figs. 5.4b and 5.4c. As expected, the bijector converts the latent Euclidean neighborhood into a semantically meaningful neighborhood. Had we included the local cross-entropy integral in this training process, all points within the orange boundary would have received the same label as the data point.

5.7.2 Out of Distribution Detection

We test how our models respond to OOD examples when trained with global integrals over the noise-contrastive prior and consistency integrals and compare to methods designed cfor OOD detection. See Appendix B.3 for standard performance and Sec. 5.6 for a discussion of the baselines. Table 5.1 contains the AUPR for the various methods against similar but different datasets (see Appendix B.4 for the AUROC). We juxtapose SVHN vs CIFAR10 and MNIST vs FMNIST plus Extended MNIST (EMNIST) [36]. We include a separable, hybrid model without integral regularizations (hybrid) and the same model with regularizations (Int. Reg.) to illustrate the utility of learning over hypervolumes. When MNIST or FMNIST are the in-distribution set, the

regularized, integrable network performs on par with the best baseline methods. SVHN and CIFAR10 show reasonable OOD performance but are not as strong as the baselines. This is not surprising since the integrals rely on a reasonable estimate of $p(\mathbf{x})$, which both datasets have failed to achieve, Table C.3.

Table 5.1: Area under the PR curve (percentage).

In	Out	GAN	ODIN	MSMA	OE	CCU	Hybrid	Int. Reg.
MNIST	FMNIST	99.4	98.8	-	99.9	99.9	93.7 ± 6.8	99.7 ± 0.17
	EMNIST	84.5	78.4	-	91.4	84.3	97.2 ± 1.9	99.8 ± 0.03
FMNIST	MNIST	99.9	99.2	80.8	97.0	98.3	63.9 ± 8.3	95.4 ± 0.04
	EMNIST	100.	99.3	-	98.6	99.1	93.8 ± 3.5	98.8 ± 0.47
SVHN	CIFAR10	98.6	97.3	92.5	100.	100.	71.8 ± 2.0	76.8 ± 1.4
CIFAR10	SVHN	80.5	92.7	99.0	98.5	97.5	84.6 ± 0.8	87.0 ± 2.1

5.7.3 Semi-Supervised Learning

We apply the local integral to the separable hybrid model and train on several tabular datasets with 10% of the labels and domain-specific augmentation strategies are unavailable. These models do not utilize the OOD class or global penalty. We apply pseudo-labelling with thresholds of 0.9-0.95. Table 5.2 compares the performance of the integrated hybrid models to several standard (non-image) semi-supervised baselines on flat MNIST (MNIST, flattened to a vector), MiniBooNE [12], and HepMass [11]. We utilize a CNF [67] as the bijector and the hinge as the classifier. We see that the integrated model achieves similar-to-better performance than the other methods on all datasets, showcasing the generality of integrated regularizers in different problem spaces.

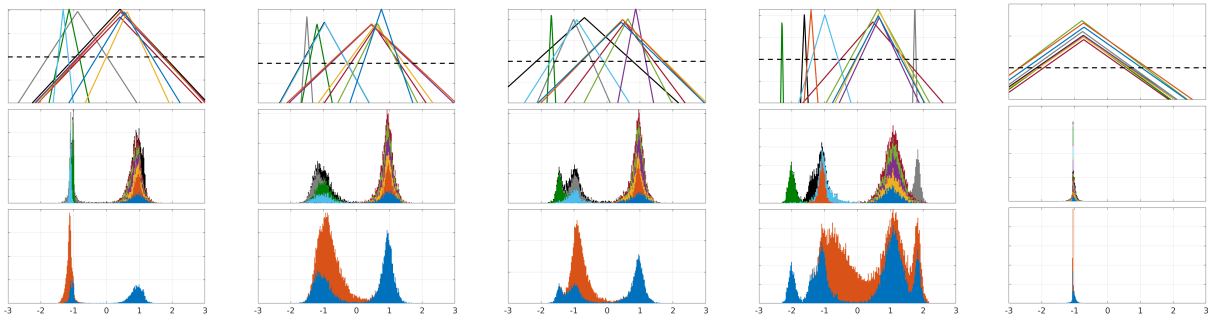
5.7.4 Interpretability

The separably model and independent latent dimensions allows us to reason about how the model is making decisions in the latent space. Unfortunately, for most bijectors, it is not possible

to carry this interpretability back to the input space. Figure 5.5 demonstrates the final state of the model trained on Fashion MNIST for several features with respect to the latent distribution, per in-distribution class and juxtaposed with the out of distribution data. Specifically, the top row contains the logit components, $f_{k,m}$, learned by the separable network, color-coded by class; the middle row contains the distribution of each class (colors matched to logits); and the bottom row contains the distribution of the in-distribution data (blue) and OOD data (red). The top row illustrates how the classification network makes its decisions per feature over the distribution presented in the middle row. We see that the logit components map well to the distribution of the data in each feature and provides some intuition for how certain the classifier is over a feature. This demonstrates how this architecture allows for reasonable interpretibility from the latent space. Any value above zero (the dotted black line) is considered in-distribution and the most likely class is the line with the greatest value at that point. The features were chosen to demonstrate the diversity of the learned solution over features. Generally, the data maps reasonably well to the bimodal distribution, though we do occassionally see mode collapse as in Fig. 5.5e. Fortunately, in these cases the logits tend to be fairly uninformative and only introduce a small bias. Figures 5.5a through 5.5c show a common trend where the OOD data has heavy overlap with one of the two clusters but not the other. While we do see some diversity amongst the in-distribution classes that overlap with the OOD data the "bags" (gray) and "sandals" (cyan) class overlap most often. Finally, Fig. 5.5d demonstrates a latent feature where the OOD data falls in the region between the two data components.

Table 5.2: Tabular dataset semi-supervised accuracy (%).

	SSVAE	VIME	Local Int.
Flat MNIST	88.9	95.8	94.9 ± 0.16
MiniBooNE	92.2	91.7	93.5 ± 0.074
HepMass	83.1	82.0	85.4 ± 1.2



(a) Feature 1 (b) Feature 2 (c) Feature 3 (d) Feature 4 (e) Feature 5

Figure 5.5: Each subplot corresponds to a single latent feature. The top and middle rows contain the unnormalized logit components and distributions per class, with matching colors. The bottom row compares the distribution of Fashion MNIST (blue) and MNIST (red). x-axis shared across all rows.

5.8 Limitations

There are two primary limitations with this method for delivering tractable integrals. First, the network relies on a bijective feature extractor to relate latent and input spaces in a principled manner that allows us to equate integrals in one space to integrals in the other space. Unfortunately, bijective feature extractors are known to provide inferior performance relative to their injective counterparts. This is demonstrated empirically by Chen et al. [30] where they demonstrate that hyrbid networks struggle with a trade-off between generative and discriminative tasks. This result is consistent with our efforts, especially when we introduce additional objectives in the form of volumetric penalties. As discussed in Sec. 5.7.2 and C.2.3, we expect we could improve the performance of our models by utilizing more sophisticated (and more expensive) bijectors such as Spline-Coupling [50] or Residual Flows [30].

The second limitation of the method pertains to the output of the separable network. Namely, when the output of the model requires some constraint across output features (*e.g.*, must sum to one or have unit norm), the separable network will not generally be able to maintain this constraint. Fortunately, it is usually possible to bypass this limitation with special tricks or by constructing approximations or bounds for the output (given the constraint and separable network) so that we can still achieve some reasonable training objective. In fact, this is the case for the

classification problems and the reason we explore them in this chapter. We demonstrate both how to construct a bound for training and the effectiveness of training given that bound.

5.9 Conclusions

In this chapter we have demonstrated how to incorporate a learned density (*e.g.*, a normalizing flow) in conjunction with a separable network to enable tractable integration over the data space. This is a capability that does not currently exist within existing architectures without relying on high-sample count Monte Carlo estimators that would be too memory intensive to utilize in training for high dimensional/high complexity data sets. We demonstrate that the ability to supervise regions and not isolated points encourages the model to learn better representations and be less likely to degenerate. We consider several formulations that allow us to regularize the model's behavior based on consistency and contrast without relying on augmentations of and sparse comparisons between a finite collection of data points. We experiment with the various integrals to obtain promising out-of-distribution detection. Through now tractable integrals, this work enables future methods and applications for learning over continuous regions.

CHAPTER 6: CONTINUOUSLY PARAMETERIZED MIXTURE MODELS

We now look to increase the transparency of deep learning generative models by parameterizing a mixure of Gaussians with a neural ODE, Sec. 2.1.2. Mixture models are universal approximators of smooth densities but are difficult to utilize in complicated datasets due to restrictions on typically available modes and challenges with initialiations. We show that by continuously parameterizing a mixture of factor analyzers using a learned ordinary differential equation, we can improve the fit of mixture models over direct methods. Once trained, the mixture components can be extracted and the neural ODE can be discarded, leaving us with an effective, but low-resource model. We additionally explore the use of a training curriculum from an easy-to-model latent space extracted from a normalizing flow to the more complex input space and show that the smooth curriculum helps to stabilize and improve results with and without the continuous parameterization. Finally, we introduce a hierarchical version of the model to enable more flexible, robust classification and clustering, and show substantial improvements against traditional parameterizations of GMMs.

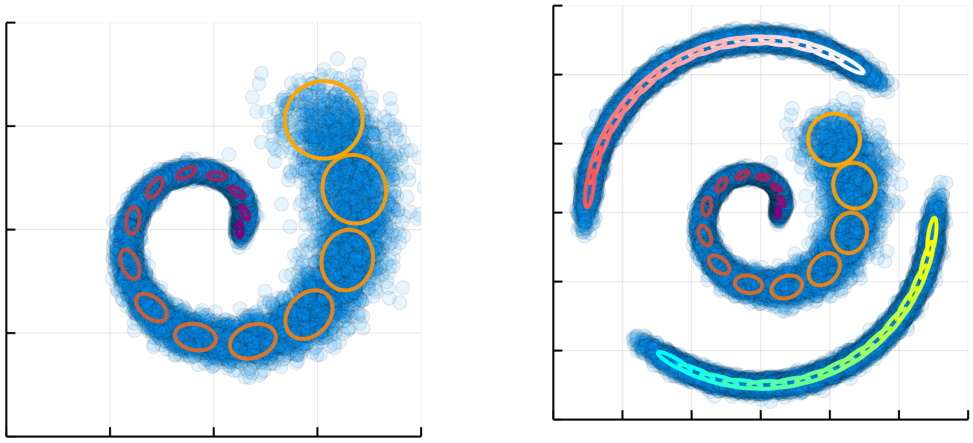
6.1 Overview

Mixture models have long served as reliable tools for both modeling (density estimation) and summarizing (cluster analysis) datasets. Through their probabilistic nature, mixture models provide an estimate of the underlying data distribution; through a parsimonious collection of components, mixture models succinctly summarize the major patterns found in a dataset. Gaussian mixture models (GMMs), which provide a universal approximation for smooth densities [65], have been effectively deployed for modeling and clustering in a wide range of applications [22, 58, 90].

There are two major challenges in applying GMMs to complicated, high dimensional data. First, the partitions (modes) that we wish to characterize in data are rarely spherical or simple enough in nature to be faithfully captured with Gaussian components. As an alternative, one may consider more complicated components [82], however, this worsens identifiability issues and yields partitions that do not adequately summarize populations of interest in the data [22]. Second, mixture models converge to local minima and thus their optimization is *extremely* sensitive to initialization. Some approaches propose to initialize via local fitting methods such as k-means [146]. However, the distance metrics, for example Euclidean distance, implied in those local fitting methods is rarely appropriate in high dimensions, imposing initial partitioning that is arduous to escape.

We propose to offset the first difficult by taking a mixture of mixtures in a hierarchical approach by considering a *mixture of distinct dynamical systems*. In effect, we propose to learn to evolve the components of one partition (see Fig. 6.1a) in a fashion that results in an infinite mixture model in the limit. This alleviates the burden of learning a large number of separate modes through a shared generator of components in an approach that is akin to hypernetworks [72]. We can then combine multiple partitions to cover the entire support (see Fig. 6.1b). Specifically, each partition is itself defined by multiple components that are spanned through smooth dynamics to represent complex, multi-modal distributions. We sample discrete components from this continuous parameterization during inference to construct a cheap, lightweight model that dramatically reduces the computational cost, maintains improved performance over typical mixture methods, and simplifies model interpretation.

We alleviate the second difficulty by creating a learned curriculum [16] over the data. We construct the curriculum by transforming the data into a standard Gaussian through a normalizing flow [30, 47, 67, 96] and slowly annealing back to the original space. This removes the difficulty with initialization since the distribution early in the training process is *known*. As we advance through the curriculum, the model only requires minor perturbations from its previous state. We



(a) Evolution of a single continuous mixture model.

(b) Evolution of several continuous mixtures. Color schemes represent different mixtures.

Figure 6.1: We illustrate how a continuously parameterized mixture model can evolve components to model the data distribution (samples in blue). Each ellipse corresponds to equal likelihood contours for a different component. Colors across components implicitly indicates the arrow of pseudotime. (a) A single *trajectory* through the space provides a smooth probabilistic model. (b) A hierarchy of three different partitions allows for different manifolds and enables flexible clustering or classification.

find that this process results in considerably improved performance regardless of the mixture parameterization.

Main Contributions We propose a parameterization of Gaussian mixture models through the output of a neural ordinary differential equation (NODE). This formulation induces a smoothly-varying trajectory through the data, in essence, learning a probabilistic sheath around the trajectory and across the data manifold. Once trained, the components of the GMM can be extracted and the NODE discarded, rendering storage and inference notably cheaper than most other modern methods. We include a hierarchical component to the model by considering multiple trajectory starting points to allow us to utilize simple Bayesian methods for clustering or classification. Finally, we demonstrate a curriculum-based training method to significantly improve model performance.

6.2 Background

6.2.1 Mixture of Factor Analyzers

Mixture of factor analyzers [62, 146] is a popular exploratory statistical model commonly used in applied disciplines such as psychology. The model for one factor analyzer is

$$\mathbf{x} = A\,\mathbf{z} + \mu + \boldsymbol{\epsilon} \tag{6.1}$$

where $A \in \mathbb{R}^{M \times R}$ is the loading matrix with $R \ll M$, $\mathbf{z} \sim \mathcal{N}(0, I)$ is the latent representation drawn from an isotropic Gaussian, $\epsilon \sim \mathcal{N}(0, D)$, and D is a diagonal matrix. Due to the distributional assumptions on $\mathbf{z}$ and $\boldsymbol{\epsilon}$, we have $\mathbf{x} \sim \mathcal{N}\left(\mu, AA^T + D\right)$. It then follows that a mixture of factor analyzers is a special case of the general GMM with the density

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}\left(\mathbf{x};\, \mu_k, A_k A_k^T + D_k\right) . \tag{6.2}$$

This choice allows for a reduction in the number of parameters from $\mathcal{O}(M^2)$ to $\mathcal{O}(MR)$. When M is large (*i.e.*, when the dimension of the input space is high), the naive evaluation of the mixture density, $p(\mathbf{x})$, is challenging as it involves inverting and calculating the determinants of large covariance matrices each of which is $\mathcal{O}(M^3)$. Fortunately, the Woodbury matrix inversion lemma and the matrix determinant lemma can be applied to significantly speed up the calculations by reducing the complexity to $\mathcal{O}(MR^2)$. We refer readers to [146] for more details.

6.3 Methods

In this section we discuss how we utilize neural ordinary differential equations (NODE) to parameterize mixture models over continuous indices and how to improve the training process by instituting a curriculum to guide the model from an easy-to-learn space to the true data space. The NODE allows for a hyper-network [72] type approach, which shares weights of a network

to output parameters over multiple components, and whose limit is an infinite GMM. We then extend a continuously parameterized mixture model by allowing for multiple trajectories in a hierarchical structure that allows us to perform clustering or classification.

6.3.1 Continuously Parameterized Mixture Models

Inspired by the universal approximator properties of Eq. 2.7, where the parameters of each component depend on the latent variable, s, we chose to parameterize the MFA via a neural network. We begin by constructing a joint state across the MFA parameters. When the data is tabular, this amounts to flattening each parameter and concatenating them together:

$$h(t) = \left[\mu(t), \bar{A}(t), \log(d)(t), \log(\pi)(t)\right] \tag{6.3}$$

where the bar over A indicates the matrix has been flattened, d is the diagonal portion of D, and $\log$ is applied element-wise. We choose to use a shared state across parameters rather than a separate ODE for each parameter so that the model can better coordinate and share relevant information. We learn the $\log$ of d and π instead of the parameters directly to ensure that the covariance is positive definite and that the weights are non-negative. As a result, $h \in \mathbb{R}^J$ where $J = (R+2)M + 1$. In most cases, we set the value of π to a constant and exclude it from the state to encourage contiguous modes and prevent a trajectory from covering a zero-probability region.

There are two possible ways to proceed. If our goal was to best match Eq. 2.7, we would construct the NODE based on the joint state and solve the system of ODEs that includes the continuous mixture

$$\begin{aligned}
\frac{dh(t)}{dt} &= f(h(t), t; \theta) \\
\frac{dp(x)}{dt} &= (2\pi)^{-M/2} \left|\mathfrak{Σ}(t)\right|^{-0.5} \exp\left(-\left(\mathbf{x} - \mu(t)\right)^T \mathfrak{Σ}^{-1}(t) \left(\mathbf{x} - \mu(t)\right)/2\right) \pi(t)
\end{aligned}$$

where we evaluate each component in accordance with Eq. 2.7 on the instantaneous value of t within the ODE solver and set $p(x) = 0$ at $t = 0$ and take the integral over the range of t without producing intermediate values and require that $\int \pi(t)\, dt = 1$. Unfortunately, while this formulation is a better match to the continuous mixture model, integrating $p(x)$ is numerically unstable and often causes f to become stiff [20] or for the integrator to underflow, see Appendix for additional details. Additionally, this form would require that we keep the NODE once training is complete and resolve it for every new example encountered.

We instead choose to solve for the joint state directly and return the parameters at a (possibly variable) number of pseudotimes. That is, for a (uniformly drawn) set of pseudotimes, $\{t_j\}_{j=1}^{G}$, we model the density as $p(\mathbf{x}) = \sum_{j=1}^{G} \pi(t_j)\, \mathcal{N}\left(\mathbf{x};\, \mu(t_j), A(t_j)A(t_j)^T + D(t_j)\right)$, where $\pi(t_j), \mu(t_j), A(t_j), D(t_j)$ are the respective continuously indexed parameters and $\sum_j \pi(t_j) = 1$. Concretely, we solve the NODE, $\frac{dh(t)}{dt} = f(h(t), t; \theta)$, for each t_j to extract the j-th component parameters and evaluate $p(x)$ using the finite sum. This approach essentially performs numerical integration to approximate Eq. 2.7, which is feasible given the $1d$ integral. This leaves us with a *mixture model* whose parameters are derived from a *continuous process*. We will refer to this model as a continuously parameterized mixture model (CPMM). Figure 6.1 illustrates a dataset overlayed with the components extracted from a CPMM.

We have specifically chosen f so that the ODE is not autonomous [20]. From a practical perspective, this means that our models are capable of learning loops and cycles. However, to further increase the flexibility of the model, we additionally augment the joint state

$$h(t) = \left[\mu(t), \bar{A}(t), \log(d)(t), \alpha(t)\right] \tag{6.4}$$

where $\alpha \in \mathbb{R}^L$ and we have excluded $\log(\pi)$ from the state. α is not directly used in evaluating any portion of the mixture but instead provides an unconstrained pathway that the network can use to pass information along. Without the augmented state, the network must encode information relevant to future times into the parameters of the current time, overloading those parame-

ters and notably decreasing performance. We generally choose L to be 2-4$\times$ larger than J, so $h \in \mathbb{R}^{5J}$.

In this form, the NODE, $\frac{dh(t)}{dt}$, does not depend on the evaluation data point, $\mathbf{x}$, directly and can be solved independently. In fact, the NODE can be evaluated with only the initial value and the parameter times. We can therefore consider that our model is a type of hypernetwork [72]. We train the model by evaluating/solving the hypernetwork against the *learnable* initial state and then evaluating the likelihood for each example in the batch. The initialization and hypernetwork dynamics are then updated to maximize the likelihood of the batch. During evaluation, we solve the hypernetwork *once* and cache the parameters (for a uniformly drawn set of pseudotimes $\{t_j\}_{j=1}^G$) against future executions. This places the total computational cost for inference at $\mathcal{O}(KMR)$.

When the input data is an image, we construct the joint (augmented) state by keeping the parameters in the image shape for the NODE and concatenating over channels which allows us to utilize CNNs for the function underlying the NODE. We then flatten the data and parameters to calculate the likelihood. This means that the unaugmented portion of the channel dimension is increased by a factor of $R + 2$ relative to the image channel count (when π is held fixed).

6.3.2 Hierarchical Mixture of Factor Analyzers

To facilitate clustering and classification, we extend the mixture of factor analyzers (MFAs) in a hierarchical manner. More specifically, we model the data with a *mixture of MFAs*

$$p(\mathbf{x}) = \sum_{c=1}^{C} \eta_c \, p(\mathbf{x} \,|c) = \sum_{c=1}^{C} \eta_c \sum_{j=1}^{G} \pi_c(t_j) \, \mathcal{N}\left(\mathbf{x};\, \mu_c(t_j), A_c(t_j) A_c(t_j)^T + D_c(t_j)\right) , \tag{6.5}$$

where C denotes the number of MFAs and, therefore the number of clusters/classes, $\sum_c \eta_c = 1$, and $\{t_j\}_{j=1}^G$ is a set of pseudotimes. We construct the mixture of the C MFAs by providing C different initial values to the NODE. Alternatively, we could learn a separate NODE and initial state for each MFA. This should allow for greater flexibility in each trajectory. However, we generally

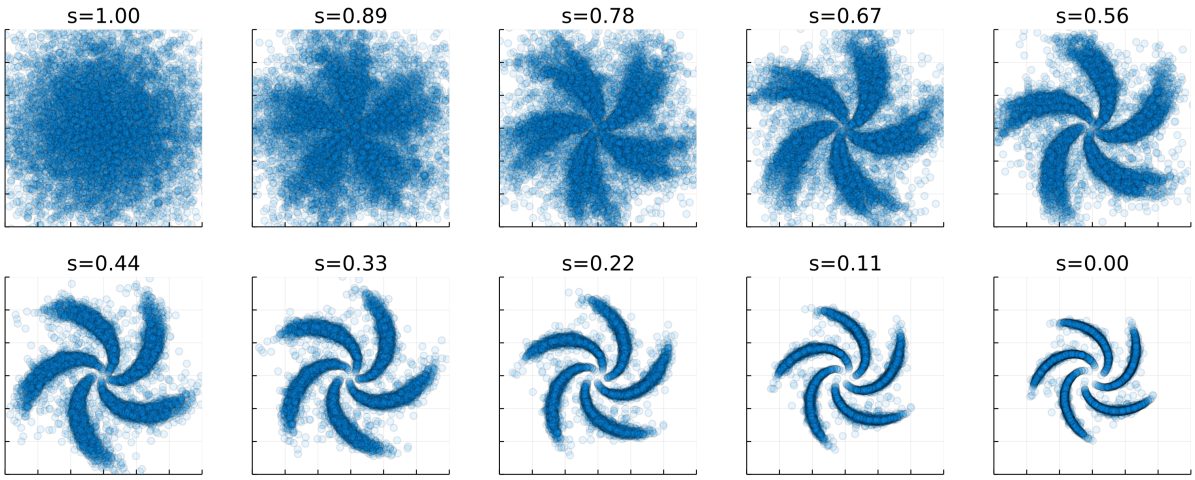


Figure 6.2: Evenly distributed transitions through different spaces. The data begins in a latent space as a standard Gaussian (top left) and ends in the true space (bottom right).

find that the improvement is not appreciable while the increase in compute is significant. Sharing the ODE means that the trajectories will learn from one another and share certain dynamic characteristics.

The hierarchical mixture allows the model to learn disparate data partitions without requiring that a single trajectory transition through low probability regions. Additionally, we can utilize Bayes rule to estimate $p(y = c|\,\mathbf{x})$ from Eq. 6.5 to perform clustering or classification. In the unsupervised setting we can utilize this to impose different forms of regularization. In the supervised setting, $p(y|\,\mathbf{x})$ can be directly supervised.

6.3.3 Curriculum Through Spaces

Unfortunately, training large mixture models in high dimensions often gets stuck in local minima and is *extremely* sensitive to initialization. Direct mixtures (mixtures with parameters that do not come from the same generative process such as the NODE) are often initialized via a combination of kmeans and local fitting prior to gradient descent [146]. However, for CPMMs, the parameters are the product of a learnable process and cannot be directly initialized. As a heuristic, one may initialize the ODE initial value based on kmeans or labeled examples, however

this only provides a limited signal and one must resolve how to initialize other parameters such as A or $\log d$.

In order to circumvent this difficulty, we borrow ideas from curriculum learning [16]. We begin by defining a continuously indexible map, $\forall v \in [0,1],\ \mathcal{M}_v : \mathbb{R}^M \mapsto \mathbb{R}^M$ such that $\mathcal{M}_0(\mathbf{x}) = \mathbf{x}$, and $\mathcal{M}_v$ progressively maps to a smoother, simpler space as v increases. We propose to leverage $\mathcal{M}$ to construct a curriculum for learning the CPMM. Intuitively, we may begin training on the simplest space induced by $\mathcal{M}_1$ and slowly progress to training on our target input space $\mathcal{M}_0$. That is, for a sequence $1 = v_1 > \ldots > v_T = 0$, we progressively train on respective datasets $\mathcal{D}_t = \{\mathcal{M}_t(\mathbf{x}_i)\}_{i=1}^N$. An accessible choice is to define $\mathcal{M}_v$ as a continuous normalizing flow [67] (see Sec. 2.2.2.4), $\mathcal{M}_v = \mathbf{u}_v$, where

$$\mathbf{u}_v(\mathbf{x}) = \int_0^v g(\mathbf{u}_s(\mathbf{x}), s; \phi) ds$$
$$\log p_{\mathbf{u}_v}(\mathbf{u}_v(\mathbf{x})) = \log \mathcal{N}(\mathbf{u}_1(\mathbf{x}); 0, I) + \int_v^1 \left(\frac{\partial g}{\partial s}\right) ds,$$

$\mathbf{u}_0(\mathbf{x}) \equiv \mathbf{x}$ and $\mathbf{u}_v(\mathbf{x})$ is defined by a learnable function, g, which is trained via MLE so that $\mathbf{u}_1(\mathbf{x}) \sim \mathcal{N}(0, I)$.

We construct the curriculum for the CPMM by first training the CPMM on $\mathbf{u}_1$. Since the data is (ideally) a standard Gaussian in this space, it is trivial to learn a good CPMM and the initialization of the CPMM parameters can be created as perturbations from the standard Gaussian. We then marginally perturb the space by taking a small step in v towards the input, *e.g.*, we train the CPMM on $\mathbf{u}_{1-\epsilon}$ where, for sufficiently small ϵ, $p_{\mathbf{u}_{1-\epsilon}}(\mathbf{u}_{1-\epsilon}) \approx p_{\mathbf{u}_1}(\mathbf{u}_1)$. We repeat this process of perturbing the space and then updating the CPMM until we have arrived back at the input space. Figure 6.2 shows this smooth transition on a two-dimensional dataset as the map transitions between the latent space, through several intermediate spaces, before arriving back at the input space.

One interpretation of this curriculum is that we learn the initialization for one space based on training over a marginally simpler version of the data. This interpretation applies to both the

parameters of the NODE hypernetwork and the initial values of each trajectory. Without this slow update procedure, even with a good initial value, the CPMM can be hard to train since the initial trajectories can point in the wrong directions and the model can struggle to shift the probability mass appropriately.

An important distinction between our use of a curriculum and the typical form of curriculum learning is that we do not want to remember earlier "tasks." Doing so would mean that the CPMM would still capture the intermediate spaces between the Gaussian noise and the input space. Since these spaces are only used as a guide and have no intrinsic meaning, maintaining them has no direct value.

6.4 Related Work

Mixture Models Mixture models can be applied to solve an array of ML problems, *e.g.*, clustering and density estimation [73], and are widely deployed in modern ML methods. Viroli et al. [171] model the variables at each layer of a deep network with a Gaussian mixture model (GMM), leading to a set of nested mixtures of linear models. Izmailov et al. [86] use a GMM as the base distribution in conjunction with normalizing flows for semi-supervised learning. Richardson et al. [146] demonstrate that GMMs better capture the statistical modes of the data distribution than generative adversarial models (GANs), while Eghbal-zadeh et al. [52] incorporate a GMM into the discriminator of a GAN to encourage the generator to exploit different modes in the data.

Tractable Likelihood Models Normalizing flows and autoregressive models are two common types of extremely effective tractable likelihood estimators. Normalizing flows (NF) learn invertible transformations to a latent space where the data has a known, prechosen, distribution, typically the standard normal. There exist considerable variations between models based on the families of invertible functions allowed [30, 47, 67, 96]. Autoregressive (AR) models [132, 135, 167] exploit the probabilistic chain rule and learn a distribution for each dimension conditioned on the

previous features. Despite their impressive performance as likelihood estimators and as generative methods, these models are surprisingly brittle. In particular, they show higher likelihoods for completely different datasets than the ones they were trained on which prevents them from being utilized for outlier detection [80, 125].

Deep Clustering Several approaches to apply deep networks for clustering have been proposed in the past few years [26, 180], centering around the concept that the input space in which traditional clustering algorithms operate is of importance. There have also been works on incorporating traditional clustering methods, such as spectral clustering or hierarchical clustering, directly into deep networks [28, 104, 150]. Mukherjee et al. [122] extend GANs for clustering by using a combination of discrete and continuous latent variables.

Mixture Models + Deep Nets for Clustering Since mixture models are the traditional models of choice for clustering tasks, arming them further with the recent development in deep learning is only natural. Zhang et al. [188] directly deployed a mixture of autoencoders with the assumption that different clusters are effectively different local data manifolds, and thus could be parametrized and learned by autoencoders. In the same vein, Pires et al. [141] model the data using a mixture of normalizing flows, where the mixture weights are computed through optimizing the variational lower bound. Jiang et al. [89] use a GMM as the prior distribution for the latent code in a variational auto-encoder (VAE), thus allowing for clustering of the input data in the latent space.

Non-parametric Bayesian Mixture Models Non-parametric Bayesian models [124] assume the number of parameters of the underlying model grows with the number of data samples. In the case of the underlying model being a non-parametric Bayesian mixture model, this indicates that for every new data sample, the model can either group it with the already-discovered mixture components or initiate a new mixture component for that data sample, increasing the number of mixture components deployed and hence the number of parameters utilized [66]. This is accomplished by using a Dirichlet process for the prior distribution of the parameters of non-parametric

Bayesian mixture models [59]. Despite effectively deploying an infinite number of mixture components during the training stage, the number of components eventually learned is determined by the finite data samples available, resulting in a learned finite mixture distribution at test time.

Sum-Product Networks Sum-product networks (SPN) [138, 139, 142] are a class of probabilistic circuits [37] that produce a tractable likelihood estimate akin to normalizing flows and autoregressive methods. SPNs are constructed as a directed acyclic graph composed of alternating "sum" and "product" operations over the evaluation of (typically) one-dimensional densities at each leaf node. Learning is often performed through the EM algorithm by updating the parameters of each leaf distribution and the weights at each sum node. This structure allows SPNs to cheaply evaluate a variety of statistical quantities (*e.g.*, marginals) without approximation. When leaf densities are Normal distributions, the SPN can be expressed as a GMM with many components [87].

6.5 Experiments

We construct our models in PyTorch [137] and train using PyTorch Lightning [55]. We used Adam [93] as the optimizer in all cases. Unless otherwise stated, we extract 25 components per trajectory for each hierarchical CPMM. Curriculum models were constructed using simple CNFs from FFJORD [67]. Since the execution of a CNF is computationally expensive, we extracted the smoothly varying datasets (see Sec. 6.3.3), $\mathcal{D}_t$, immediately after training and saved them to disk. In general, we choose to use 51 total spaces, $\mathcal{M}_t$, (including the latent space, $\mathbf{u}_1$, and the input space, $\mathbf{x}$) and allocate one epoch per space. We then fine-tune on the input space. All models trained directly in the input space utilize standard data augmentations (see Appendix for further details). Models trained with the curriculum are initialized randomly; other models are initialized via kmeans or a combination of kmeans and fitting a local Factor Analyzer as in [146]. When training a hierarchical model, we often find it helpful to include entropy regularizations across trajectories.

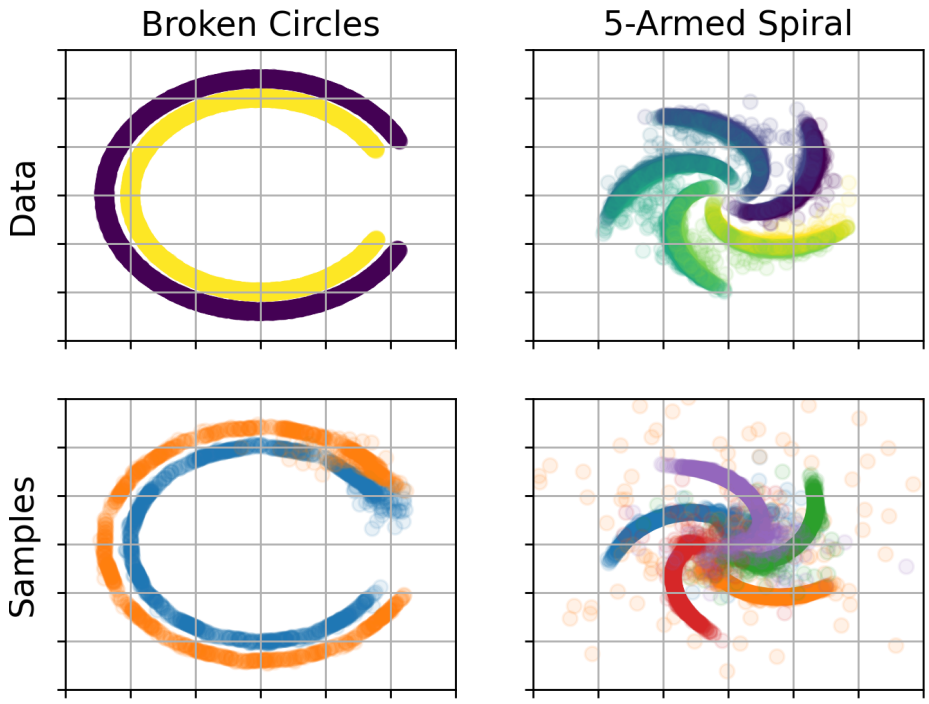


Figure 6.3: Two synthetic datasets designed to assess the limitations of the CPMM.

6.5.1 Synthetic Data

We demonstrate the effectiveness and weaknesses of CPMM on two toy datasets. The first dataset consists of two concentric, broken circles. The radii of both circles are chosen to differ by 10% and have similar thicknesses. The second dataset consists of a noisy five-armed spiral that begins near the origin before curving outward. This dataset is intentionally constructed with "outliers" (points between the arms without a clear cluster identity) to assess how the CPMM will handle examples "far" from the manifold. Both CPMMs are trained using a curriculum.

Figure 6.3 contains true data points in the top row and samples drawn from the CPMM on the bottom row. Since the model is trained unsupervised, we use different colors between true labels and cluster labels (*e.g.*, between the top and bottom row). We see that the CPMM does a reasonable job of capturing the general manifolds of both datasets. However, the model seems to struggle with the end points of the trajectory and the samples are less precise than during the other portions of the trajectory. We observe this trend on all datasets: the CPMM often struggles with endpoints, typically the early pseudotimes. In the spiral data, we see that the model has

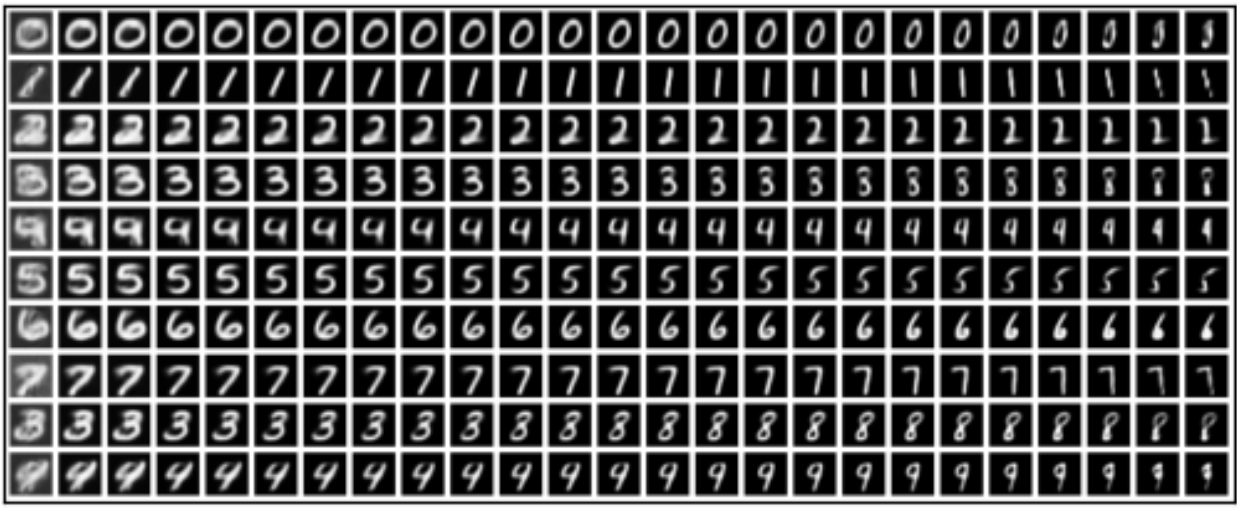


Figure 6.4: Means from a hierarchical CPMM trained on MNIST with a space-based curriculum. Each trajectory is evaluated at evenly spaced pseudotimes between zero and one and displayed from left to right. Despite the 3/8 and 4/9 confusion, the model does an excellent job of learning smoothly-varying transitions between digits. The initial component (far left cases) are never the most likely component and could be discarded.

attempted to capture the outliers, though they have larger spread than the true outliers and it attributes them all to the same cluster.

6.5.2 Images

To process image datasets, we first rescale the input pixels to be between zero and one and then apply an element-wise logit transform, a standard preprocessing technique (*e.g.*, [67]). We perform these transformations so that the "input" data has support over $\mathbb{R}$. These transformations are applied before we train any model and the corresponding $\log\det$ `Jacobian` is accounted for when estimating bits per dimension (BPD) as in a normalizing flow.

We test training CPMMs on MNIST [40] and Fashion-MNIST [178] and compare to two different GMM baselines along with several standard likelihood models and clustering models. CPMM NODEs are constructed using CNNs with a depth of 3, a hidden channel width of 64, and utilize $\sin$ activations. We additionally augment the state space by a factor of four and explicitly condition the ODEs on pseudotime by concatenating it as an extra channel. To avoid degeneracies, we soft-clip $\log d$ so that the minimum value cannot be less than -6.

Table 6.1: Bits per dimension (BPD) and clustering accuracy (Acc.) for several image datasets.

	MNIST		Fashion-MNIST	
	BPD	Acc.	BPD	Acc.
Matched GMM	6.61	61.75	6.58	55.00
GMM w/ S.Clust.	5.21	48.91	5.71	48.52
EiNet (Matched)	2.19	-	4.18	-
EiNet (Large)	1.95	-	3.98	-
GMM w/ Curr.	2.61	64.47	4.35	58.41
CPMM	2.21	80.07	3.91	65.01
Teacher	1.04	-	2.85	-
FFJORD	1.01	-	2.75	-
VADE	-	94.5	-	57.8
SPC	-	99.21	-	67.94
DLS	-	97.6	-	69.3

We consider two baseline GMMs for comparison. In the first model, we choose the total number of components equal to the number of clusters (*e.g.*, 10). This simple procedure allows for easy cluster assignments; each component corresponds to exactly one cluster. In the second case, we train a GMM with 250 components since this corresponds exactly to the number of components extracted across trajectories when using a CPMM. However, attributing components to clusters is less obvious in this case. We choose to use spectral clustering [57] over components based on the Fréchet distance over Gaussians [49]. Specifically, we attribute examples to components and components to clusters based on a spectral clustering model that constructs an affinity matrix through the Fréchet distance. Both models are initialized and trained as discussed in Sec. 3.3.

Table 1 summarizes the results for the different mixture models in juxtaposition to standard baselines [3, 44, 118, 138]. The matched GMM models (one component per cluster) achieves surprisingly high clustering accuracies but subpar BPD. Unsurprisingly, introducing more components (GMM w/ S.Clust.) improves the BPD. However, spectral clustering across components is only marginally successful and the clustering accuracy drops relative to the baseline GMM. We

additionally train two EiNets [138], a type of sum-product network. The first EiNet (Matched) is constrained to have the same total number of parameters (not components) as the the CPMM. The second EiNet (Large) utilizes an order of magnitude more parameters. Finally, we train a standard GMM with 250 components that additionally utilizes the curriculum. The CPMM trained with the curriculum enjoys significantly improved BPD and clustering accuracy relative to the GMM baselines, better performance compared to the matched EiNet and standard GMM with the curriculum, and similar performance against the large EiNet. (See Appendix for additional ablation experiments.) Our results indicate that our proposed methodology closes the considerable gap between mixture models and modern neural baselines (listed in the bottom of Table 6.1). An especially notable case is the Fashion MNIST clustering accuracy, where the CPMM results in comparable-to-better than the standard baselines. Thus, CPMM allows one to retain the interpretability and inference speed of mixture models with improvements in modeling performance over tradition mixture approaches.

Interpretability Figure 6.4 shows the trajectories learned by a CPMM where we have manually ordered the clusters. The figure shows smooth transitions along the trajectory for all digits, in general transitioning from fatter, curvier digits to slimmer, straight digits. The early pseudotimes (far left) are never the most likely component and could be excluded for a slight increase in the BPD. We have not done so here for the sake of transparency and to avoid arbitrary post-hoc operations. This particular model does an excellent job of isolating different digits into different trajectories with the exception of some 3/8 and 4/9 confusion that results in a clustering accuracy of 80%. This confusion is understandable given the similarities in the digit pairs but more importantly, this analysis highlights the *interpretability* of our model. That is, it is simple to resolve the model's confusion from the trajectories since CPMM directly operates over input features. In contrast to more opaque clustering models, a quick visual inspection reveals CPMM's partioning of the space. Similar results and discussion can be found for Fashion-MNIST in the Appendix.

OOD Detection Finally, we test the CPMM's ability to distinguish between different datasets based on likelihood. This is a known shortcoming of neural likelihood models [80, 125] where

Table 6.2: OOD Detection via Likelihood Thresholding

		CPMM		FFJORD	
In	Out	AUROC	AUPR	AUROC	AUPR.
MNIST	Fashion	99.95	99.95	99.95	99.95
Fashion	MNIST	93.38	93.28	8.98	31.65

the networks unfortunately predict that simpler datasets are more likely than the data the model was trained on; *e.g.*, MNIST instances yield a higher likelihood than Fashion-MNIST instances for models trained on Fashion-MNIST. This surprising result limits the applicability of modern likelihood models for detecting out-of-distribution (OOD) and anomalous instances, despite their intuitive capabilities. Table 6.2 illustrates the performance of our model against a CNF for detecting instances that are out-of-distribution (Out) when trained on a distinct inlier distribution (In) with a simple thresholding of likelihoods. Specifically, we evaluate the likelihood of each image using the CPMM and FFJORD when images come from the inlier distribution and when they come from an outlier distribution (e.g., In: Fashion, Out: MNIST) and call an example in-distribution if the likelihood exceeds some threshold and call it out-of-distribution otherwise. Results are given as areas under the receiver operating characteristic and precision/recall curves as a percent. Higher is better. Although both models perform nearly perfectly when MNIST is in-distribution and Fashion-MNIST is out-of-distribution, the CPMM performs quite well on the reverse problem where the CNF fails miserably. Notwithstanding a gap in the likelihood that is obtainable with CPMM as compared to CNF (Table 6.1), this experiment illustrates an advantage to our simplified approach of directly modeling the input data through a mixture of components.

6.6 Limitations

The primary difficulty of the proposed method centers on the complexity of the underlying trajectories and how to construct a space-filling curve. This is compounded by our desire for a trajectory to be matched to a particular class or cluster of the data and the limitation that we

forced each component to be equally likely, *e.g.*, $\pi_c(t)$ is constant for all c and t. This limitation is necessary to restrict the trajectories from crossing through low-probability regions and maintain a coherent cluster. Without this requirement, the 3/8 confusion we observed would be better accepted as the intermediate cases (mid-times/components) would simply have little-to-no probability. This has the advantage of increasing the effectiveness of the generative portion of the model but would significantly diminish its discriminative properties. Unfortunately, more complex data manifolds may require clusters with sophisticated trajectories that would be aided by the ability to perform complex corrections in low-likelihood regions. This is especially true if we wish to consider clusters of objects that can take different evolutionary paths.

Two possible solutions exist to rectify this complication: additional trajectories for each diverging path or the ability for the model to bifurcate at branch points. The up front addition of more branches is naive and requires advanced knowledge of how many branches exist and how to group independent trajectories back into a tree-like structure. Introducing the ability for the model to bifurcate alleviates the difficulty with grouping but still requires a knowledge of how many branching opportunities should be allowed. Fortunately, it may be possible for a model to choose where to branch. We consider this an interesting direction for future research.

6.7 Conclusions

We have demonstrated that even simple likelihood models can provide additional benefits beyond the simple generative properties they are usually ascribed when properly parameterized and trained. Speficially, mixture models are interpretable, well-behaved, computationally-cheap likelihood models that are, unfortunately, increasingly difficult to employ as the data complexity and dimensionality increases. These difficulties are largely due to the relatively simple components and sensitivities to model initializations. Neural networks have proven themselves to be incredibly powerful models but their performance is inexplicable and they exhibit bizarre failure modes. In this chapter, we have proposed a hierarchical method to parameterize a continuum of mixture models from neural ODEs to create a rich, multi-modal density over disparate partitions,

essentially constructing a mixture of mixtures where the outer mixture encompasses different data regions and the inner mixture can be arbitrarily complex.

We have additionally innovated a curriculum that alleviates many of the difficulties associated with initialization. The curriculum utilizes an annealing process from a prescribed latent space chosen for its simplicity towards the true, nuanced data space. The transitions are learned via maximum likelihood estimation through a continuous normalizing flow. Since the initial distribution is known, it is trivial to initialize a model that is well-matched. Finally, we sample a finite number of components from the dynamic NODE to achieve an effective, light-weight model that significantly outperforms mixtures with a similar number of components, indicating that the combination of the curriculum and dynamic system allows for a more efficient use of the available components than traditional methods.

While this method provides notable improvements in terms of likelihood predictions and clustering accuracies in comparison to typical mixture methods, it unsurprisingly, fall short of neural network methods that utilize learned latent spaces on the primary metrics typically used in model assessment. However, we do not exhibit the same brittleness (*e.g.*, larger likelihoods on OOD data), require significantly less computation (once trained), and are completely interpretable. We additionally have the ability to arbitrarily condition our models if a subset of features is known a priori. Similarly, we can easily marginalize a subset of features to produce a completely consistent model without requiring any additional training. Neither of these benefits are available to pure neural network models further demonstrating the additional capabilities that likelihood models can provide for alternate processes and functionality.

CHAPTER 7: CONCLUSION

In this dissertation we have addressed shortcomings in likelihood models over less structured data by introducing a clever mechanism to link exchangeable likelihood models to sequential models, enabling practitioners to exploit the huge corpus of existing research from normalizing flows and autoregressive models. Bridging this gap is of increasing importance as the proliferation of unordered collections across a number of domains continues to increase. For example, unordered collections are inherent for the lidar arrays utilized by driverless cars, land surveys, and national defense. Our method remains the only model that allows for sophisticated likelihood modeling without relying on *i.i.d.* assumptions in the input or latent spaces or without requiring impractical marginalizations over all permutations. We have empirically demonstrated that this model provides notable improvements in likelihood scoring and sample quality using a variety of data domains.

We have additionally demonstrated several methods that utilize a learned density to improve the performance of a larger system. Specifically we showed that by constructing a generative process over neural networks to improve adversarial robustness. This is an important problem as artificial agents become increasingly prevalent in areas pertaining to human safety and security. We have additionally proposed a method for enabling high-dimensional integrals to estimate and regularize the expected behavior of a network. The ability to estimate the expected behavior of a network has a huge number of applications ranging from general semi-supervised learning to average adversarial training. We can also introduce more custom integrals to help regularize a network to better match expected behavior derived from domain expertise. A critical component of this method exploits the transformative portion of a normalizing flow and without this underlying likelihood model, the construction would not be practical. Finally, we have proposed a universal

approximator of smooth denisities by constructing a continuously parameterized mixture model that maintains many of the benefits of traditional density estimates without a significant loss of performance relative to state of the art methods. We have illustrated how this flexible model can be used for explainable decision making when performing clustering and classification. Overall, we have clearly shown that likelihood models have significant utility in addition to their direct generative mode.

Future Works Our efforts have also enabled several interesting directions for future efforts. In particular, we hypothesize that it may be possible to improve the bridge between exchangeable and sequential models by utilizing a learnable ordering function instead of relying on a sorting operation. This should ease the learning process of the permutation equivariant transforms by explicitly providing gradient information about how the scan is making its decisions. Additionally, we expect that utilizing more sophisticated flow transformations should result in across-the-board performance improvements when utilizing a composition of bijective and separable functions to enable integrations. Finally, we expect that allowing for bifurcation within the trajectories of the continuously parameterized mixture should allow for more flexible groups.

APPENDIX A: FLOWSCANS

A.1 Proof of Prop. 1

Proof. First, note that $|\det\frac{\mathrm{d}\hat{q}}{\mathrm{d}\mathbf{x}}|$ is invariant to Γ since one may compute the Jacobian of $\hat{q}(\Gamma\mathbf{x})$ as the composition of $\hat{q}$ followed by a permutation and the determinant of the Jacobian of a permutation is one. Furthermore, $\hat{f}(\hat{q}(\Gamma\mathbf{x})) = \hat{f}(\Gamma\hat{q}(\mathbf{x})) = \hat{f}(\hat{q}(\mathbf{x}))$, by the permutation equivariance of $\hat{q}$ and permutation invariance of $\hat{f}$. Hence, the total likelihood $\hat{p}(\mathbf{x}) = \left|\det\frac{\mathrm{d}\hat{q}}{\mathrm{d}\mathbf{x}}\right|\hat{f}(\hat{q}(\mathbf{x}))$ is permutation invariant. □

A.2 Experiment Details

Experiments were implemented in Tensorflow [2]. We use multiple stacked Real NVP transformations with a Gaussian exchangeable process for BRUNO. For both BRUNO and NS, we experimented with publicly available implementations in addition to our own. We observed superior performance for each using our own implementations and thus report these results. All results reported for BRUNO are the best-of-six validated trained cases. We validated eighteen different modifications of the Neural Statistician where we toggled if the variance of the code was fixed (learned or fixed at one), the hidden layer sizes (64, 128, or 256), and the code size (16, 64, or 256). See [131], and [51] for further details. (Largest models were typically not best.) Results are reported on the best of the eighteen models. Additionally, each model was initialized six different times and the best model was then trained to convergence. The best model was selected based on a held-out validation set. For FlowScan, we used equivariant pointwise transformations by stacking RNN-coupling and invertible leaky-ReLU transformations [131]; after the scan we implemented the autoregressive likelihood with a 2-layer GRU (256 units), which conditioned a TAN density [131] on points. Models were optimized for 40k iterations on TitanXP GPUs. Modelnet data was gathered as in [184], brain data was gathered as in [35]. All datasets used a random 80/10/10 train/validation/test split.

A.2.1 ModelNet10 Ablation Study

We performed an ablation study using `ModelNet10`. We begin with a full flow scan model, which performs an equivariant flow transformation, scans, does corresponding coupling transformations, and uses an auto-regressive model. Next, we omit the correspondence coupling transformation. After, we also remove the equivariant flow transformation. Finally, we considered a basic model without an autoregressive likelihood, that only scans and has a flat-vector density estimate on the vector of concatenated covariates. The models achieve per point log likelihoods of 3.01, 2.67, 2.34, and 2.27, respectively. We see that each component of FlowScan is improving the likelihood estimate. It is also interesting that the basic scan model is still outperforming the NS likelihood bound.

A.3 Synthetic

The synthetic data in Sec. 3.3.1 is generated as follows:

$$\begin{aligned} x_1^{(1)} &\sim \mathcal{N}(2, n^{-2}) \\ x_1^{(2)} &\sim \mathcal{N}(0, n^{-2}(1 + (\pi/3)^2)) \\ x_k^{(1)} &\sim \mathcal{N}(x_1^{(1)} \cos(\pi k/n), n^{-2}) \\ x_k^{(2)} &\sim \mathcal{N}(\cos(\pi k/n + x_1^{(2)}), n^{-2}) \end{aligned}$$

A.4 Permutation Equivariant Transformations

For the sake of completeness, we develop several novel permutation equivariant transformations which do not transform each set element independently.

Recall that a transformation $q : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$ is permutation equivariant if for any permutation matrix Γ, $q(\Gamma \mathbf{x}) = \Gamma q(\mathbf{x})$. Furthermore, recall that one may construct a simple permutation

equivariant transformation by transforming each element of a set identically and independently:

$$(x_1, ..., x_n) \mapsto (q(x_1), ..., q(x_n)). \tag{A.1}$$

However, this transformation is unable to capture any dependencies between points, and operates in a *i.i.d.* fashion. Instead, we propose equivariant transformations that transform each element of a set in a way that depends on other points in the set, yielding a richer family of models. In other words, transforming as

$$(x_1, ..., x_n) \mapsto (q(x_1, \mathbf{x}), ..., q(x_n, \mathbf{x})). \tag{A.2}$$

Below, we propose several novel equivariant transformations with intra-set dependencies.

A.4.1 Linear Permutation Equivariant (L-PEq)

We start with a linear permutation equivariant transformation. It can be shown [184] that any linear permutation equivariant map of one-dimensional points can be written in the form, $\mathbf{x} \mapsto (\lambda\mathbf{I} + \gamma\mathbf{1}\mathbf{1}^T)\mathbf{x}$ for some scalars λ and γ, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$. Specifically, a linear permutation equivariant transformation is the result of a matrix multiplication with identical diagonal elements and off diagonal elements.

Such a transformation captures intradependencies by mapping the jth dimension of the ith point as

$$x_i^{(j)} \mapsto \lambda^{(j)} x_i^{(j)} + \frac{\gamma^{(j)}}{n} \sum_{k=1}^{n} x_k^{(j)}, \tag{A.3}$$

incorporating the mean of other points in the set. We use the mean rather than the sum as in [184] because it allows for better symmetry with our proposed generalization in Sec. A.4.2. It is trivial to go between the two formulations by scaling $\gamma^{(j)}$ by n. The log-determinant of the transformation equation equation **??** can be show to be $(n-1)\log|\lambda^{(j)}| + \log|\lambda^{(j)} + \gamma^{(j)}|$, and is

invertible whenever $\lambda^{(j)} \neq 0$ and $\lambda^{(j)} + \gamma^{(j)} \neq 0$ with inverse:

$$z_i^{(j)} \mapsto \frac{z_i^{(j)}}{\lambda^{(j)}} - \frac{\gamma^{(j)}}{n\lambda^{(j)}(\lambda^{(j)} + \gamma^{(j)})} \sum_{k=1}^{n} z_k^{(j)}$$

A.4.2 Nonlinear Weighting (NW-PEq)

We propose a generalization of the linear permutation equivariant transformation equation equation **??** here. Instead of a direct mean, we propose to weight each element by some nonlinear function that depends on the element's value relative to a global operation over the set:

$$\begin{aligned} x_i^{(j)} &\mapsto \lambda^{(j)} x_i^{(j)} + \gamma^{(j)} \frac{\sum_k x_k^{(j)} w\left(x_k^{(j)}\right)}{\sum_m w\left(x_m^{(j)}\right)} \\ &\mapsto \lambda^{(j)} x_i^{(j)} + \gamma^{(j)} \eta^{(j)} \end{aligned} \tag{A.4}$$

where w is the nonlinear weighting function and $\eta^{(j)}$ is the weighted mean. The log-determinant of the Jacobian can be expressed as

$$\begin{aligned} \log|J| &= (n-1)\log|\lambda^{(j)}| \\ &+ \log\left|\lambda^{(j)} + \gamma^{(j)}\left(1 + \frac{\sum_k \left(x_k^{(j)} - \eta^{(j)}\right) w'\left(x_k^{(j)}\right)}{\sum_m w\left(x_m^{(j)}\right)}\right)\right| \end{aligned} \tag{A.5}$$

where w' is the first derivative of w. It is clear that A.5 simplifies to the linear determinant for constant w. Attempting to invert A.4 results in an implicit function for $\eta^{(j)}$

$$\eta^{(j)} = \left(1 + \gamma^{(j)}\right)^{-1} \frac{\sum_k x_k^{(j)} w\left(x_k^{(j)} - \gamma^{(j)}\eta^{(j)}\right)}{\sum_m w\left(x_m^{(j)} - \gamma^{(j)}\eta^{(j)}\right)} \tag{A.6}$$

(where $\lambda^{(j)}$ has been dropped for brevity) that could be solved numerically to perform the inverse for a general nonlinear weight. This formulation implies that the weighted permutation equivariant transform can be inverted even if w is not an invertible function. Thus, allowing for a

larger family of nonlinearities than is typically included in transformative likelihood estimators [45, 46, 97].

The simplest method forward is to choose a weighting function such that a sum of function inputs decomposes into a product of outputs, e.g. $w(a+b) = f(a)f(b)$ where f is some nonlinear function. In this case, A.6 simplifies to

$$\eta^{(j)} = \left(1+\gamma^{(j)}\right)^{-1} \frac{\sum_k x_k^{(j)} f\left(x_k^{(j)}\right)}{\sum_m f\left(x_m^{(j)}\right)} \tag{A.7}$$

and the inverse transform proceeds trivially. Choosing f to be the exponential function allows for the simplification, guarantees positive weights, and results in a softmax-weighted mean,

$$x_i^{(j)} \mapsto \lambda^{(j)} x_i^{(j)} + \gamma^{(j)} \frac{\sum_k x_k^{(j)} \exp\left(\beta^{(j)} x_k^{(j)}\right)}{\sum_m \exp\left(\beta^{(j)} x_m^{(j)}\right)} \tag{A.8}$$

with inverse temperature scaling β. It is apparent that this transformation reduces to the L-PEq transformation when $\beta = 0$. Additionally, in the limit as $\beta \to \infty$ or $\beta \to -\infty$, the transformation tends to shift by the maximum or minimum of the set, respectively. The log-determinant of this transformation is identical to the linear case and the inverse comes directly from (A.7):

$$z_i^{(j)} \mapsto \frac{z_i^{(j)}}{\lambda^{(j)}} - \frac{\gamma^{(j)}}{\lambda^{(j)}\left(\lambda^{(j)}+\gamma^{(j)}\right)} \frac{\sum_k z_k^{(j)} \exp\left(\frac{\beta^{(j)} z_i^{(j)}}{\lambda^{(j)}}\right)}{\sum_m \exp\left(\frac{\beta^{(j)} z_i^{(j)}}{\lambda^{(j)}}\right)}$$

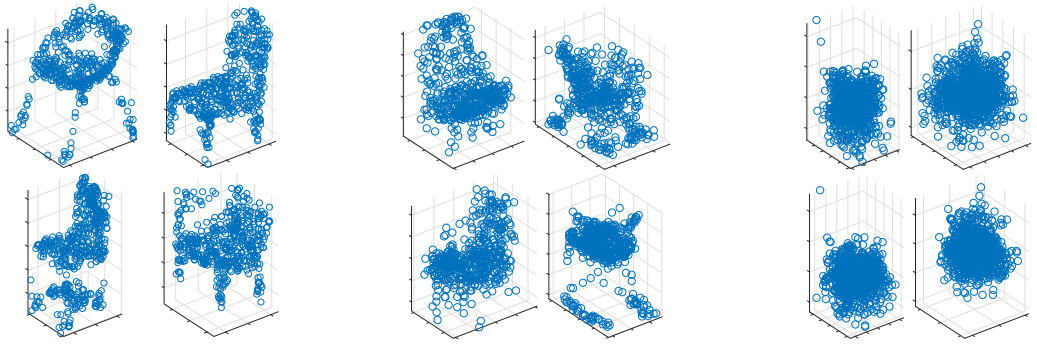
where $\lambda^{(j)}$ has been reintroduced. Other choices for the nonlinear weight function w are possible, however finding a good map that has both a closed-form log-determinant and inverse is non-trivial. Alternate weighting functions remain a direction for future research.

A.5 Generated Samples for Point Cloud Experiments

Below we plot additional sampled sets using the methods compared in our experiments in Figures A.1-A.5.

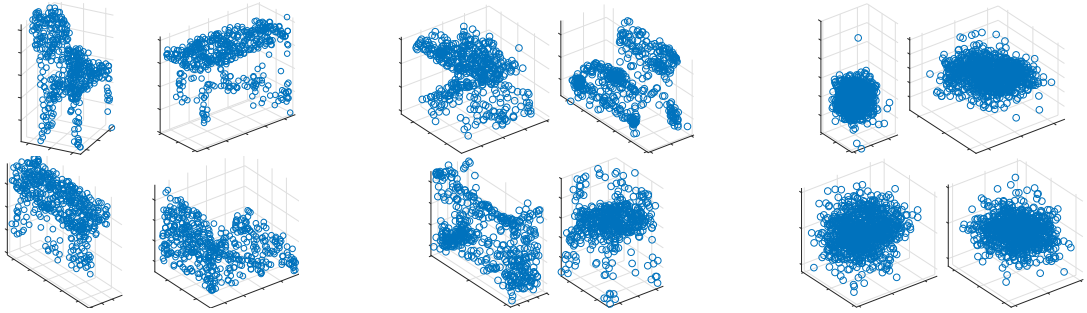
A.6 Training Examples for Point Cloud Experiments

The training data includes 10,000 points per set for all ModelNet data sets and approximately 1,000 points for both brain substructures. The various models used a randomly selected, 512 point subset during training, validation, and testing. We plot training instances in Figure A.6.



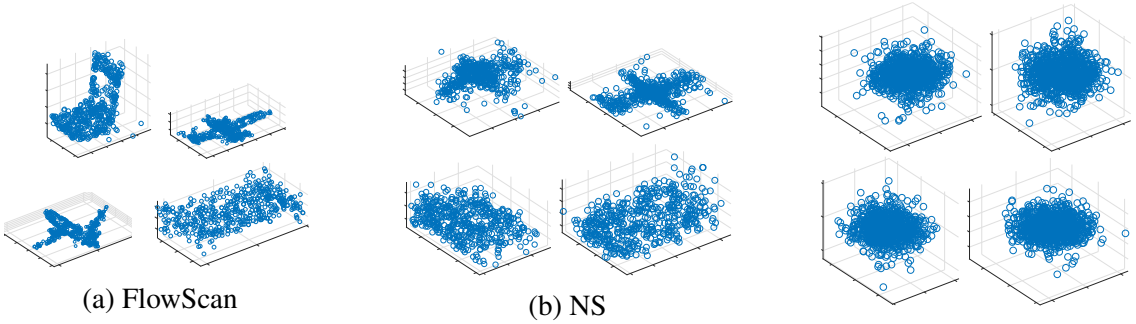
(a) FlowScan (b) NS (c) BRUNO

Figure A.1: Chair Samples



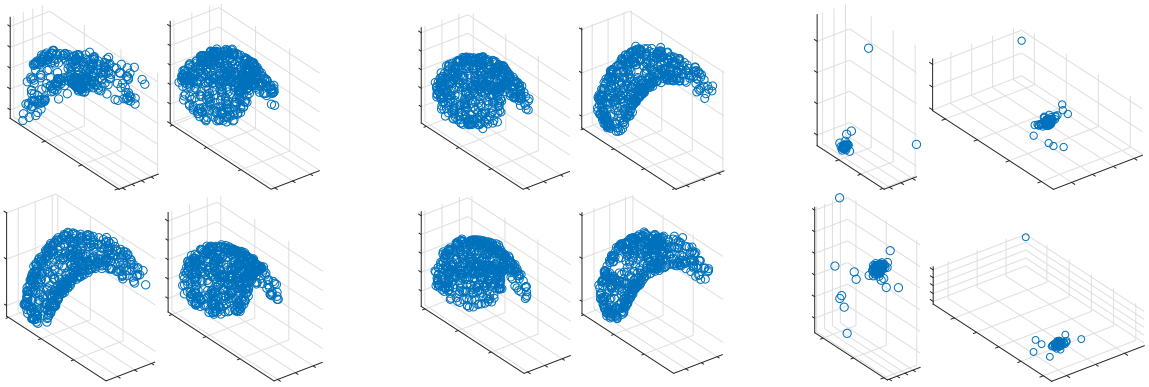
(a) FlowScan (b) NS (c) BRUNO

Figure A.2: ModelNet10 Samples



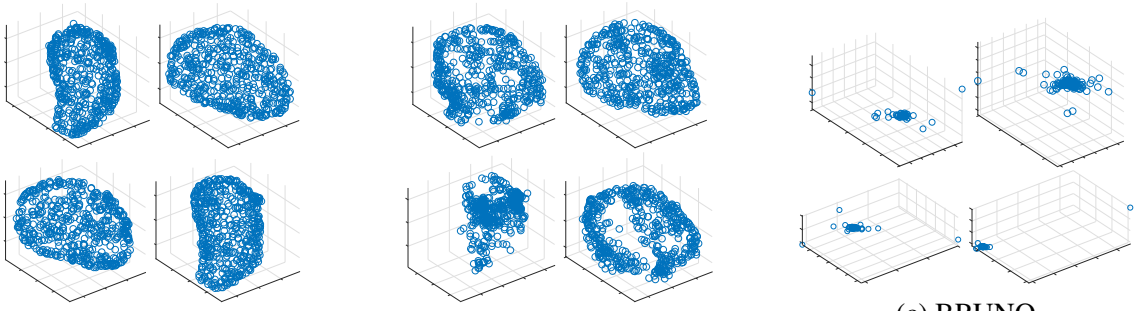
(a) FlowScan (b) NS (c) BRUNO

Figure A.3: ModelNet10a Samples



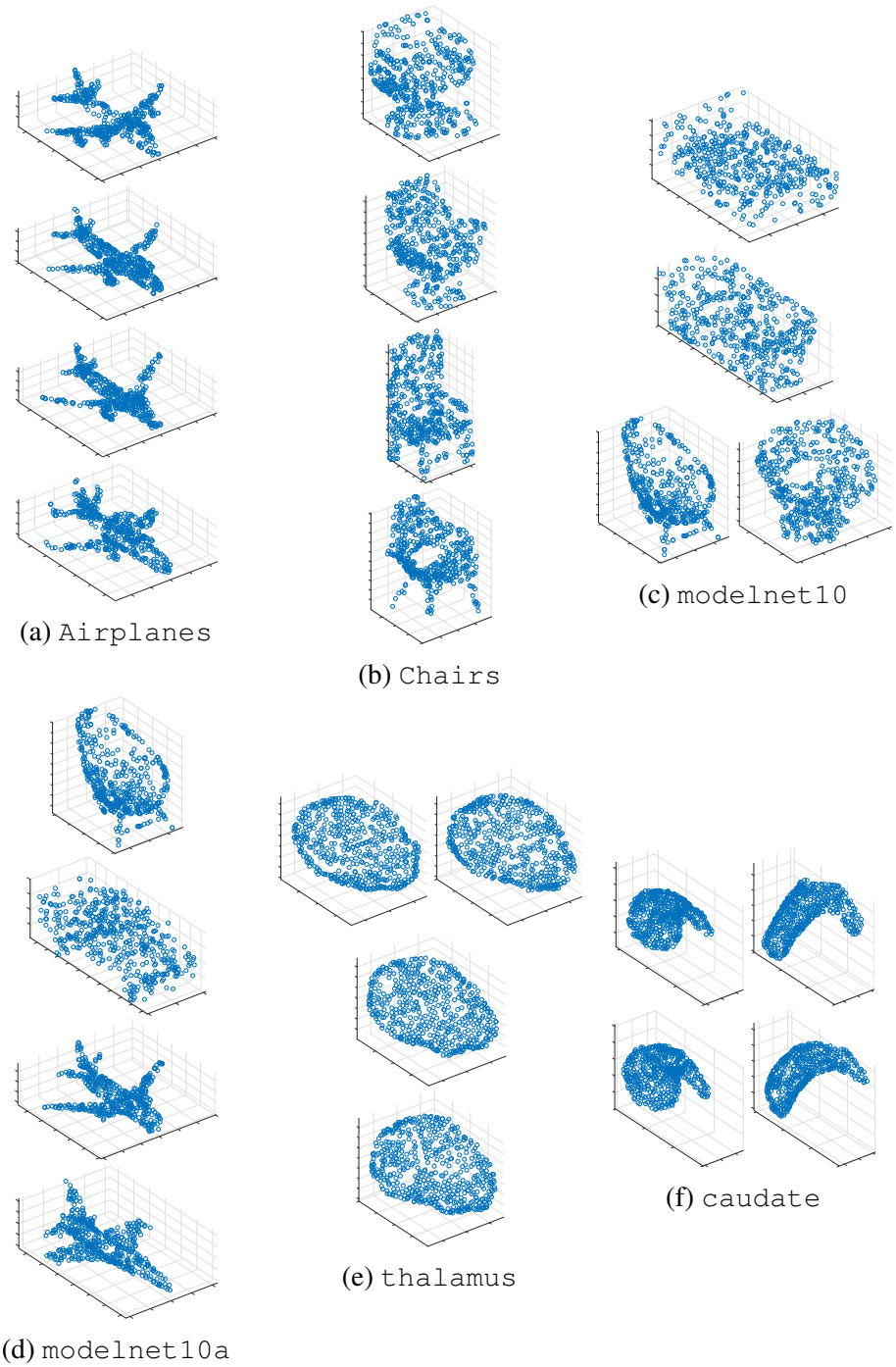
(a) FlowScan (b) NS (c) BRUNO

Figure A.4: Caudate Samples



(a) FlowScan (b) NS (c) BRUNO

Figure A.5: Thalamus Samples



(a) `Airplanes` (b) `Chairs` (c) `modelnet10` (d) `modelnet10a` (e) `thalamus` (f) `caudate`

Figure A.6: Training examples

APPENDIX B: SUPPORTING INFORMATION FOR DIVERSE DEFENSES

B.1 Proof of Proposition 1

Proof. Suppose $\mathbf{X} \sim f_{\mathbf{X}}(\mathbf{x})$. Since the elements of $\mathbf{X}$ are independent, the entropy of the random vector $\mathbf{X}$ equals the sum of the entropy of each individual element

$$\begin{aligned} H(\mathbf{X}) &= -\int_{\mathbf{x}} f(\mathbf{x}) \ \log f(\mathbf{x}) d\mathbf{x} \\ &= -\sum_i \int_{\mathbf{x}} \left[\prod_j f(x_j) \right] \log f(x_i) d\mathbf{x} \\ &= -\sum_i \int_{x_i} f(x_i) \log f(x_i) \ dx_i \\ &= \sum_i H(X_i) \end{aligned} \tag{B.1}$$

and that

$$\begin{aligned} H(X_i) &= \int_{x_i} f(x_i) \log \frac{1}{f(x_i)} \ dx_i \\ &= \int_{x_i} (x_i - \mu_i)^2 f(x_i) \log \left(e^{\frac{1}{(x_i-\mu_i)^2}} + \frac{1}{f(x_i)} \right) \ dx_i. \end{aligned}$$

Since by assumption each X_i is bounded in a compact interval, we have $\log \left(e^{1/(x_i-\mu_i)^2} + 1/f(x_i) \right) > 1$. It also achieves its maximum and minimum values on this compact interval which we denote as k and h, respectively. We then have

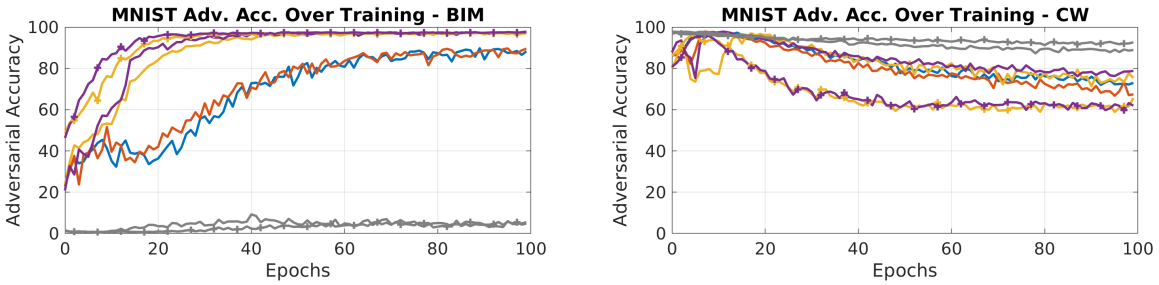
$$h \cdot \mathrm{Var}[X_i] \leq H(X_i) \leq k \cdot \mathrm{Var}[X_i]$$

indicating that maximizing the entropy of $\mathbf{X}$ is equivalent to maximizing $\sum_i \mathrm{Var}(X_i)$. □

B.2 Additional Results

Method	BIM	CW	FGM	PGD
CNN	7.1	30.5	36.8	2.6
BNN	5.5	88.9	55.2	0.9
BNN-Off	4.8	-	40.0	2.1
M	88.2	73.0	93.0	55.2
M-V	89.5	67.4	94.0	70.0
M-S	97.1	75.4	96.3	94.0
M-V-S	97.8	78.8	97.2	95.8
M-S-Off	97.2	65.6	97.6	95.8
M-V-S-Off	97.5	61.9	97.4	94.5

Table B.1: Adversarial accuracy on MNIST with various combinations of diversity promotion against L_∞ attacks.



(a) art attack

(b) art attack

Figure B.1: White box attack accuracy after each epoch.

APPENDIX C: SUPPORTING INFORMATION FOR PRACTICAL INTEGRATION

C.1 Proofs

C.1.1 Additively Separable Functions

Theorem C.1.1 (Additive Independent Integration)**.** *Given an additively separable function, $f(\mathbf{v})$, an independent likelihood, $p(\mathbf{v}) = \prod_{m=1}^{M} p_m(v_m)$, and domain, $\mathcal{D}_v = \mathcal{D}_{v_1} \times ... \times \mathcal{D}_{v_M}$:*

$$\mathbb{E}_{\mathbf{v}\sim p(\mathbf{v})}\left[f\left(\mathbf{v}\right)\right] = \sum_{m=1}^{M} \int_{\mathcal{D}_{v_m}} f_m\left(v_m\right) p_m\left(v_m\right) dv_m \tag{C.11}$$

Proof.

$$\int f_n\left(v_n\right) p_m\left(v_m\right) dv_m =$$

$$\begin{cases} f_n\left(v_n\right), & \text{if } n \neq m \\ \int f_n\left(v_n\right) p_n\left(v_n\right) dv_n, & \text{else} \end{cases} \tag{C.12}$$

$$\mathbb{E}_{\mathbf{v}\sim p(\mathbf{v})}\left[f\left(\mathbf{v}\right)\right] = \int_{\mathcal{D}} \left(\sum_{n=1}^{M} f_n\left(v_n\right)\right) \left(\prod_{m=1}^{M} p_m(v_m)\right) d\mathbf{v} \tag{C.13}$$

$$= \sum_{n} \int_{\mathcal{D}} f_n\left(v_n\right) \prod_{m=1}^{M} p_m(v_m)\, d\mathbf{v} \tag{C.14}$$

Substituting Eq. C.11 into Eq. C.13 reduces the M-dimensional integral over independent likelihoods into a 1-dimensional integral and completes the proof. □

C.1.2 Multiplicatively Separable Functions

Theorem C.1.2 (Multiplicative Independent Integration)**.** *Given a multiplicitively separable function, $g(\mathbf{v})$, an independent likelihood, $p(\mathbf{v}) = \prod_{m=1}^{M} p_m(v_m)$, and domain, $\mathcal{D}_v = \mathcal{D}_{v_1} \times ... \times$*

$\mathcal{D}_{v_M}$:

$$\mathbb{E}_{\mathbf{v}\sim p(\mathbf{v})}\left[g\left(\mathbf{v}\right)\right] = \prod_{m=1}^{M} \int_{\mathcal{D}_{v_m}} g_m\left(v_m\right) p_m\left(v_m\right) dv_m \tag{C.15}$$

Proof. Since each component of g_n is matched to a component in p_m, the two products can be recombined to isolate like-dimensions. Each dimension can then be integrated independently.

$$\begin{aligned}
\mathbb{E}_{\mathbf{v}\sim p(\mathbf{v})}\left[g\left(\mathbf{v}\right)\right] &= \int d\mathbf{v} \left(\prod_{n=1}^{M} g_n(v_n)\right)\left(\prod_{m=1}^{M} p_m(v_m)\right) \\
&= \int \cdots \int \left(\prod_{n=1}^{M} g_n(v_n) p_n(v_n)\right) dv_1 \cdots dv_M \\
&= \int g_1(v_1)p_1(v_1)dv_1 \cdots \int g_M(v_M)p_M(v_M)dv_M
\end{aligned}$$

□

C.2 Additional Experiments

All image datasets were trained using Glow as the bijector with the number of levels adjusted for the size of the image. MNIST and Fashion MNIST were trained with Adam for 50 epochs with an exponentially decaying learning rate of $\gamma = 0.75$. SVHN and CIFAR10 utilized a similar training procedure but with 75 epochs and a slower exponential decay of $\gamma = 0.99$. We find that utilizing an average instead of a sum over separable features and soft-thresholding at -1 improves training stability.

C.2.1 Adversarial Robustness

We test the efficacy of the local cross-entropy integral by utilizing a synthetic dataset from [163]. The data is constructed via

$$y \sim \{-1,\ +1\}, \qquad x_1 = \begin{cases} +y, & \text{w.p. } p \\ -y, & \text{w.p. } 1-p \end{cases}, \qquad x_2, ..., x_{M+1} \sim \mathcal{N}(\eta y, 1) \tag{C.21}$$

and the adversarial examples are constructed analytically by, essentially, flipping the sign of η. We utilize this synthetic dataset to test adversarial robustness because it provides an upper bound on the adversarial accuracy relative to the standard accuracy which allows us to quantify our performance with respect to an absolute. Additionally, because adversarial examples can be constructed exactly without relying on an optimization procedures, there can be no concerns that this defense relies on obfuscated gradients [9]. We refer the interested reader to the original paper for a thorough discussion.

We choose $M = 10, p = 0.95$, and $\eta = 2/\sqrt{M}$. For these choices, a strong classifier will obtain near perfect standard accuracy with zero adversarial accuracy while a robust classifier will obtain 95% standard and adversarial accuracy. We use a similar architecture to that found in Sec. C.3.1 except we utilize ten blocks instead of five. We train one model with only the standard cross-entropy and negative log-likelihood losses and a second model with the additional global and local cross-entropy integration loss. We find that the model with only the standard losses achieves reasonably good standard performance and fairly poor adversarial accuracy. The model with the additional integration losses contains considerably better adversarial performance, nearing the upper and, necessarily lower, standard accuracy bound. Table C.1 summarizes our results and the ideal performances.

Table C.1: Std. & Adv. Accuracy

	Std.	Adv.
Accurate	100.0	0.0
Robust	95.0	95.0
Baseline	97.9	27.5
Integrated	94.5	90.8

C.2.2 Toy Semi-supervised Regression

To illustrate the utility of global regularizations, we construct a one-dimensional, semi-supervised problem with $x \sim \mathcal{N}(0, 4)$ and $y = \tanh(x)$.

Table C.2: Semi-supervised integration regularization

$\mathbb{E}[y] = 0$	Standard	Integrated
Sup. MSE	1.195e-5 $\pm$ 1.882e-5	2.487e-5 $\pm$ 1.769e-5
Unsup. MSE	1.399 $\pm$ 1.198	0.06187 $\pm$ 0.04538
$\mathbb{E}[\hat{y}]$	0.1041 $\pm$ 0.08582	8.570e-4 $\pm$ 4.150e-4

We keep all values of x but re-

move y values corresponding to

negative x during training. Unlike the standard semi-supervised problem, the missing labels are not random but are the result of limitations or bias in the data collection process. We train two models that are identical except that the integrated model includes a penalty over $\mathbb{E}\left[\Omega\left(\hat{y}\left(\mathbf{x}\right)\right)\right]$ based on foreknowledge that the average value is zero. Table C.2 shows how the models perform over the supervised and unsupervised regions. The inclusion of the integration regularizer allows the model to reason about regions that it has not directly observed and has decreased the error in that region by two orders of magnitude.

C.2.3 Standard Performance

We test the impact of integrable models on OOD performance against several standard image classification datasets: MNIST [40], Fashion MNIST (FMNIST) [178], SVHN [128], and CIFAR10 [101]. See Appendix B for architures, distributions, and training parameters. Table C.3 contains the validation standard accuracy and bits per dimension for all datasets with all integration regularizers. The upper portion of the table is averaged over 10 runs and contains a reject option in the separable model. The lower portion is a single run without a reject option or any integrated regularizers. We find that the model performs reasonably well for MNIST and FMNIST but the integrated losses cause a large degredation in performance for SVHN and CIFAR10. The removal of these losses produces similar accuracies but much-improved BPD, consistent with the hybrid network results reported by ResidualFlows [30] when Glow is used.

Table C.3: Standard accuracy and bits-per-dimension for different datasets

	Acc.	BPD
MNIST	98.3 ± 0.2	2.54 ± 0.13
FMNIST	88.1 ± 2.3	4.76 ± 0.64
CIFAR10	73.4 ± 1.8	5.62 ± 0.76
SVHN	89.9 ± 0.7	4.14 ± 0.08
CIFAR10*	76.5	3.78
SVHN*	90.0	2.24

C.2.4 Out of Distribution Detection AUROC Comparisons

Table C.4: Area under the ROC curve (percentage)

		GAN	ODIN	MSMA	OE	CCU	Hybrid	Int. Reg.
MNIST	FMNIST	99.4	98.7	-	99.9	99.9	93.6 ± 6.5	99.8 ± 0.12
	EMNIST	92.8	88.9	-	95.8	92.0	77.7 ± 12.6	98.2 ± 0.28
FMNIST	MNIST	99.9	99.0	82.6	96.3	97.8	69.8 ± 9.2	95.4 ± 3.1
	EMNIST	99.9	99.3	-	99.3	99.5	63.4 ± 18.2	87.8 ± 4.4
SVHN	CIFAR10	96.8	95.9	97.6	100.	100.	87.0 ± 0.7	91.3 ± 0.6
CIFAR10	SVHN	83.9	96.7	99.1	98.8	98.2	73.5 ± 1.8	78.85 ± 2.7

C.2.5 MNIST Leave-one-out

Following [4], we test the robustness of our method against semantic OOD examples by training models on all but one class from MNIST [40] and assessing OOD performance on the held out class. We repeat the process for each class and report the AUROC and AUPR. In all cases, we train using ten different random seeds and report the average performance and standard deviation across seeds that do not degenerate against the in-distribution validation set. Table C.6 and Table C.5 contain the results of our method using both integrated losses compared to several relevant baselines. Results from Efficient GAN-Based Anomaly Detection (EGBAD) [187] and GANomaly [6] are taken by estimating the results from figures in both works. The Semantic Anomalies [4] results are obtained by executing the official version of the code using the rotation auxiliary task with two different batch sizes (128 and 256). The Semantic Anomalies results in both tables are the best across both batch sizes and detection methods (ODIN [111] and MSP [79]) based on the AUPR.

We see that the Semantic Anomalies generally achieves the best AUROC across all digits. The integration often achieves the best AUPR but struggles with certain held out digits. In particular, the integration performs significantly worse on the "4" and "9" digits. This is a reasonable confuser due to the similarities and variability of the two classes. These results indicate that the integration penalties are helpful for inducing the model to detect semantic anomalies.

Table C.5: MNIST leave one out: AUPR (%)

Method	0	1	2	3	4
EGBAD	78	29	67	52	46
SA: Rotation	89.24	83.57	78.20	66.33	90.45
Global & Local	95.26 $\pm$ 1.98	89.67 $\pm$ 4.33	86.02 $\pm$ 3.19	93.41 $\pm$ 1.88	69.89 $\pm$ 7.02
	5	6	7	8	9
EGBAD	43	57	35	54	35
SA: Rotation	83.38	75.57	95.19	68.84	84.88
Global & Local	87.76 $\pm$ 2.1	91.7 $\pm$ 2.84	83.43 $\pm$ 3.72	87.5 $\pm$ 2.91	72.9 $\pm$ 7.23

Table C.6: MNIST leave one out: AUROC (%)

Method	0	1	2	3	4
EGBAD	78	29	67	52	45
GANomaloy	88	65	95	79	80
SA: Rotation	98.85	98.17	94.60	94.22	98.23
Global & Local	95.93 $\pm$ 1.91	89.32 $\pm$ 4.76	84.83 $\pm$ 3.94	94.25 $\pm$ 1.25	74.49 $\pm$ 6.98
	5	6	7	8	9
EGBAD	43	57	39	55	36
GANomaloy	85	85	68	85	55
SA: Rotation	97.21	92.52	99.16	92.69	97.02
Global & Local	89.47 $\pm$ 2.32	93.11 $\pm$ 2.98	83.45 $\pm$ 4.01	88.42 $\pm$ 2.97	75.66 $\pm$ 5.65

C.3 Training and Architectures

C.3.1 Spirals

The bijective layer is composed of five blocks where each block contains an affine-coupling layer [47] and an invertible fully connected layer. The data distribution is bimodal over both latent dimensions with means at ± 1 and standard deviations of 0.4. The separable network is composed of quadratic functions. We train the model as discussed, using the standard cross-entropy loss for each observed point, the negative log-likelihood over the input space, and the global cross-entropy integration with respect to the contrastive prior. The learning rate is set to 0.001 with standard weight decay of 1e-4 over 20 epochs using Adam [93].

C.3.2 Fashion MNIST

The bijective layers are composed of the Glow [96] architecture with two levels composed of 32 steps and 512 channels. We utilize an open-source Glow implementation[1] wrapped in Pyro's [18] bijective operators. The data distribution is bimodal over all latent dimensions with means at ± 1 and standard deviations of 0.5. The noise constrastive distribution is a standard Gaussian. The separable network is composed of hinge functions (see Sec. 4.3). We utilize a batch size of 256,

[1] https://github.com/chrischute/glow

a learning rate of 1.5e-4 over the bijective layers, and a learning rate of 1.0e-3 over the separable layers using Adam [93]. Both learning rates are exponentially decayed at a rate of 0.75 per epoch. Standard weight decay is applied with a weight of 1e-4. The network is trained for 50 epochs. The adversarial attacks are performed against the model at the epoch with the lowest validation cross-entropy loss.

C.4 Approximate Expected Cross-Entropy over a Domain

We desire the expected cross-entropy, $\text{CE}(\hat{y}(\mathbf{x}), y_c)$, over a domain $\mathcal{D}_c$ where $\mathbf{y}_c$ is the desired probability vector over classes, $\hat{y}(\mathbf{x})$ is the model's prediction of $y \in \mathbb{R}^K$ from $\mathbf{x} \in \mathbb{R}^M$ and is composed of a bijective function, $\mathbf{h}$, a separable function, $f_k(\mathbf{x}) = \sum_m f_{k,m}(x_m)$, and the soft-max function, σ, such that $\hat{\mathbf{y}}(\mathbf{x}) = \sigma(\mathbf{f}(\mathbf{h}(\mathbf{x})))$. If we are attempting to regularize the model to produce a single class label over $\mathcal{D}_c$, then $\mathbf{y}$ will correspond to a one-hot vector. In general, however, $\mathbf{y}_c$ may be a dense vector with elements y_k. The expected cross-entropy is expressed as

$$\mathbb{E}_{\mathcal{D}_c}\left[\text{CE}(\hat{y}(\mathbf{x}), y_c)\right] = -\sum_{k=1}^{K} y_k \mathbb{E}_{\mathcal{D}_c}\left[\log(\hat{y}_k(\mathbf{x}))\right] = -\sum_{k=1}^{K} y_k \int_{\mathcal{D}_c} \log(\hat{y}_k(\mathbf{x}))\, p_{\mathcal{D}_c}(\mathbf{x}|\mathbf{y}) d\mathbf{x} \quad \text{(C.41)}$$

where $p_{\mathcal{D}_c}(\mathbf{x}|\mathbf{y})$ is the probability density function over $\mathcal{D}_c$ conditioned on $\mathbf{y}$. We can take advantage of the bijective transform to convert this expression into an expectation over the latent space, $\mathbf{z}$, and latent domain, $\mathcal{Z}_c$, which allows us to write

$$\sum_{k=1}^{K} y_k \mathbb{E}_{\mathcal{D}_c}\left[\log(\hat{y}_k(\mathbf{x}))\right] = \sum_{k=1}^{K} y_k \mathbb{E}_{\mathcal{Z}_c}\left[\log(\sigma(\mathbf{f}(\mathbf{z}))\right] = \sum_{k=1}^{K} y_k \int_{\mathcal{Z}_c} \log(\sigma_k(\mathbf{f}(\mathbf{z})))\, p_{\mathcal{Z}_c}(\mathbf{z}|\mathbf{y}) d\mathbf{z} \quad \text{(C.42)}$$

where σ_k is the kth element after the soft-max normalization and $p_{\mathcal{Z}_c}(\mathbf{z}|\mathbf{y})$ is the independent probability density function in the latent space, e.g., $p_{\mathcal{Z}_c}(\mathbf{z}|\mathbf{y}) = \prod_m p_m(z_m)$. We can then

expand the soft-max operator within the expectation to get

$$\sum_{k=1}^{K} y_k \mathbb{E}_{\mathcal{Z}_c} \left[\log(\sigma(\mathbf{f}(\mathbf{z})) \right] = -\mathbb{E}_{\mathcal{Z}_c} \left[\log \left(\sum_{j=1}^{K} \exp(f_j(\mathbf{z})) \right) \right] + \sum_{k=1}^{K} y_k \mathbb{E}_{\mathcal{Z}_c} \left[f_k(\mathbf{z}) \right] \tag{C.43}$$

The combination of the separable function and independent densities allows for an exact simplification of the second term into

$$\begin{aligned} \sum_{k=1}^{K} y_k \mathbb{E}_{\mathcal{Z}_c} \left[f_k(\mathbf{z}) \right] &= \sum_{k=1}^{K} y_k \sum_{m=1}^{M} \int_{\mathcal{Z}_m} f_{k,m}(z_m) p_m(z_m) dz_m \\ &\approx \frac{1}{G} \sum_{k=1}^{K} y_k \sum_{m=1}^{M} \sum_{g=1}^{G} f_{k,m}(\tilde{z}_{m,g}) \end{aligned} \tag{C.44}$$

where we have overloaded $\mathcal{Z}_m$ to correspond to the domain of the mth latent dimension and have approximated the integral via Monte Carlo integration with $\tilde{z}_{m,g} \tilde{p}_m(z_m)$ and G draws. The first term on the right-hand side of Eq. C.43 does not enjoy the same exact simplification. However, it is possible to bound the term via Jensen's Inequality by moving the logarithm into the sum over j or outside the expectation operator. We consider the latter case where

$$\mathbb{E}_{\mathcal{Z}_c} \left[\log \left(\sum_{j=1}^{K} \exp(f_j(\mathbf{z})) \right) \right] \leq \log \left(\mathbb{E}_{\mathcal{Z}_c} \left[\sum_{j=1}^{K} \exp(f_j(\mathbf{z})) \right] \right). \tag{C.45}$$

Then, we expand the separable function, exchange the order of the sum over dimensions and the exponent, take advantage of the multiplicative integral simplification (Eq. C.15), approximate via

Monte Carlo integration, and utilize the $\log \sum \exp$ operator for stability:

$$\begin{aligned}
\log\left(\mathbb{E}_{\mathcal{Z}_c}\left[\sum_{j=1}^{K}\exp(f_j(\mathbf{z}))\right]\right) &= \log\left(\mathbb{E}_{\mathcal{Z}_c}\left[\sum_{j=1}^{K}\exp(\sum_{m=1}^{M}f_{j,m}(z_m))\right]\right) \\
&= \log\left(\sum_{j=1}^{K}\int_{\mathcal{Z}_c}d\mathbf{z}\prod_{m=1}^{M}\exp(f_{j,m}(z_m))\prod_{n=1}^{M}p_n(z_n)\right) \\
&= \log\left(\sum_{j=1}^{K}\prod_{m=1}^{M}\int_{\mathcal{Z}_m}\exp(f_{j,m}(z_m))p_m(z_m)dz_m\right) \\
&\approx \log\left(\sum_{j=1}^{K}\prod_{m=1}^{M}\sum_{g=1}^{G}\exp(f_{j,m}(\tilde{z}_{m,g}))\right) - M\log(G) \\
&= \log\sum_{j=1}^{K}\exp\sum_{m=1}^{M}\log\sum_{g=1}^{G}\exp(f_{j,m}(\tilde{z}_{m,g})) - M\log(G).
\end{aligned} \tag{C.46}$$

Finally, we can substitute Eq. C.44 and Eq. C.46 into Eq. C.41 to get

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}_c}\left[\text{CE}(\hat{y}(\mathbf{x}), y_c)\right] &\leq -\sum_{k=1}^{K}y_k\sum_{m=1}^{M}\int_{\mathcal{Z}_m}f_{k,m}(z_m)p_m(z_m)dz_m \\
&\quad + \log\left(\sum_{j=1}^{K}\prod_{m=1}^{M}\int_{\mathcal{Z}_m}\exp(f_{j,m}(z_m))p_m(z_m)dz_m\right) \\
&\approx -\frac{1}{G}\sum_{k=1}^{K}y_k\sum_{m=1}^{M}\sum_{g=1}^{G}f_{k,m}(\tilde{z}_{m,g}) \\
&\quad + \log\sum_{j=1}^{K}\exp\sum_{m=1}^{M}\log\sum_{g=1}^{G}\exp(f_{j,m}(\tilde{z}_{m,g})) - M\log(G)
\end{aligned} \tag{C.47}$$

In the event that y_c is a one-hot vector (at the cth element) this simplifies to

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}_c}\left[\text{CE}(\hat{y}(\mathbf{x}), y_c)\right] &\leq -\sum_{m=1}^{M}\int_{\mathcal{Z}_m}f_{c,m}(z_m)p_m(z_m|c)dz_m \\
&\quad + \log\left(\sum_{j=1}^{K}\prod_{m=1}^{M}\int_{\mathcal{Z}_m}\exp(f_{j,m}(z_m))p_m(z_m|c)dz_m\right)
\end{aligned} \tag{C.48}$$

where we reintroduced the condition over the class within the probability densities $p_m(z_m|c)$ for emphasis.

APPENDIX D: SUPPORTING INFORMATION FOR CONTINUOUSLY PARAMETERIZED MIXTURES

D.1 Numerical Integration

The typical strategy to train a likelihood model is to optimize based on the predicted *log-likelihood* and not on the likelihood directly. In fact, when training a Gaussian mixture model, we typically calculate the log-likelihood of each component separately prior to utilizing the numerically-stable $\log \sum \exp$ operator instead of performing the $\exp$, followed by the $\sum$, followed by the $\log$. This joint version stably handles difficulties resulting from over- or under-flow when the log-likelihood of any single component is very (large) small.

When attempting to evaluate the log-likelihood for a continuum of components we should utilize a similar $\log \int \exp$:

$$\log p(\mathbf{x}) = \frac{-M}{2} \log(2\pi) + \log \int_0^1 \exp\left(-\log |\mathfrak{Sigma}(s)| /2 - (\mathbf{x} - \mu(s))^T \mathfrak{Sigma}^{-1}(s) (\mathbf{x} - \mu(s)) /2 + \log \pi(s) \right) ds.$$

Unfortunately, none of the available black-box integrators match this form. This requires that we utilize a naive implementation of the $\log \int \exp$ operator. In practice, we find this sufficient to train the model for several epochs without issue but the ODE becomes increasingly stiff before ultimately underflowing and "`NaN`ing" out. A better strategy to estimate a continuous mixture model will require a new black-box integrator which we consider an excellent avenue for future work.

D.2 Augmentations

For all image datasets we added uniform random noise between zero and one *prior to rescaling*, e.g., a pixel with an 8bit value of 34 would then be distributed uniformly between 34 and 35. For color images, we additionally perform standard random translations and horizontal flips. For

tabular data, we add Gaussian noise to each feature such that the standard deviation of the noise is 1-10% of the spread of the data itself, e.g., we normalize each feature to be zero mean and unit standard deviation, add Gaussian noise scaled by 0.01 to 0.1 and unnormalize that data.

When training with the curriculum (see Sec. 6.3.3), we utilize similar augmentations prior to transforming into the various spaces such that the augmentation is shared across spaces. When training the CPMM student, we then add a small mount of Gaussian noise (standard deviation of 0.01) regardless of the space, including when fine-tuning on the input space.

D.3 MNIST Ablations

We experiment with different types of initializations and modifications of the CPMM on MNIST. In particular, we test random initializations and kmeans-based initializations. We also explore restricting the diagonal portion of the covariance within each MFA to be constant (though still learnable and still with the low rank updates, A) and attempt to utilize a separate ODE for each cluster.

Table D.1: MNIST Ablation

Init.	Diagonal	BPD	Acc.
Random	Constant	7.67	48.83
kmeans	Constant	7.56	64.93
kmeans	Variable	4.49	46.42
kmeans*	Constant	7.55	71.22

Table D.1 contains some representative runs attempting to train the CPMM without guidance from the teacher. We find that training the models with a variable diagonal often results in the model degenerating and failing to train. In fact, training with a variable diagonal and a random initialization tends to fail after a few epochs and is therefore excluded from the table. Utilizing a kmeans initialization produces moderately good BPD (better than standard GMM training methods) but slightly worsened accuracies. Restricting the model to use a constant diagonal greatly stabilizes training and can produce reasonably good clustering accuracies. However,

this strong assumption on the covariance results in very poor BPD. This is not surprising, as the model must raise the variance of pixels that would normally be very small (background pixels) to sufficiently capture the spread elsewhere. Finally, the bottom row in Table D.1 contains results using a kmeans initialization with a constant diagonal but utilizes a separate ODE for each trajectory. This added flexibility results in virtually no improvement to the BPD but does provide a noticeable increase in accuracy. We find that training separate ODEs per cluster results in a significant increase to run time.

The results in Table D.1 were chosen as representative values across many different runs. In general, it is possible to trade accuracy for BPD in any given row by adjust other hyperparameters or seeds. Utilizing the curriculum learning process helps to stabilize training and allows us to simultaneously improve the generative and discriminative portions of the model.

D.4 Fashion MNIST Results

Figure D.1 contains the trajectories from a curriculum-based CPMM trained on Fashion-MNIST. Similar to MNIST, the trajectories show smooth variations between means. Many of the trajectories contain visible evolutions from start to finish, i.e., `bags` begin without handles but end with handles or shoes evolve from boots to stilletos. However, some of the fine detail is missing from the trajectories, in particular for the `shirts`, `pullovers`, and `coats` classes.

As in MNIST, we can clearly see why the model achieves 65.74% clustering accuracy. The `shirts`, `pullovers`, and `coats` classes all produce very similar trajectories. Similarly, the `sandals`, `sneakers`, and `ankle boots` classes see considerable overlap. Abstractly, this information can be extracted from any clustering model based on the confusion matrix. But, thanks to the interpretability of the mixture models, the decision process and resulting confusion is clear.

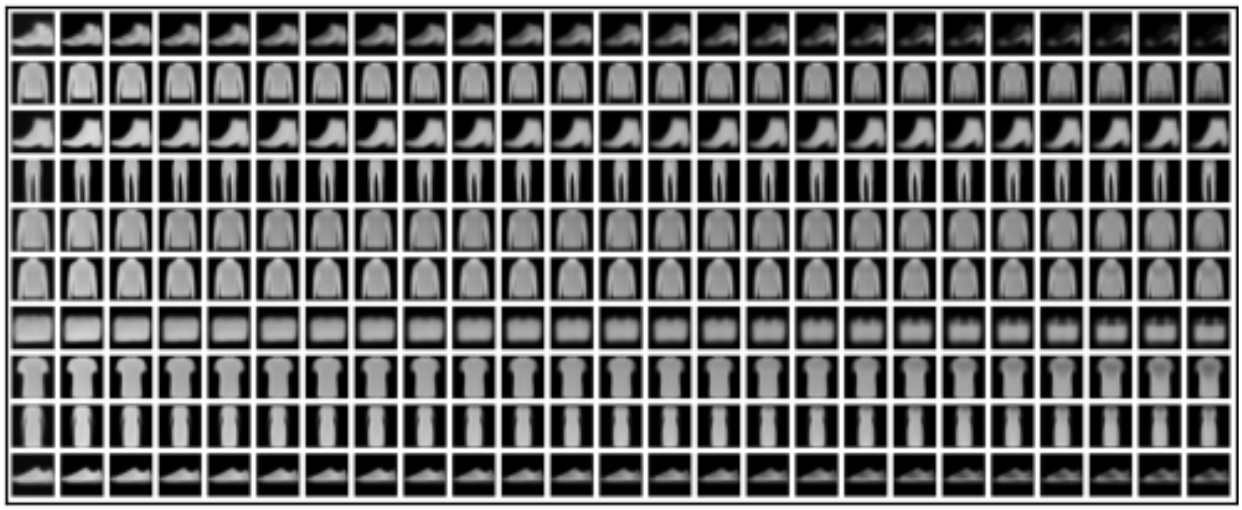


Figure D.1: Means from a hierarchical CPMM trained on Fashion-MNIST with a space-based curriculum. Each trajectory is evaluated at evenly spaced pseudotimes between zero and one and displayed from left to right. The initial component (far left cases) are never the most likely component and could be discarded.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[3] Abien Fred Agarap and Arnulfo P. Azcarraga. Improving k-means clustering performance with disentangled internal representations. *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[4] Faruk Ahmed and Aaron Courville. Detecting semantic anomalies. In *Proceedings of 34th AAAI Conference on Artificial Intelligence*, 2020.

[5] Abhimanyu S. Ahuja. The impact of artificial intelligence in medicine on the future role of the physician. *PeerJ*, 2019. doi: https://doi.org/10.7717/peerj.7702.

[6] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*, pages 622–637. Springer, 2018.

[7] Michael A. Alcorn and Anh Nguyen. The DEformer: An order-agnostic distribution estimating transformer. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021. URL `https://openreview.net/forum?id=H1qfnDmta6`.

[8] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[9] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283. PMLR, 10–15 Jul 2018. URL `http://proceedings.mlr.press/v80/athalye18a.html`.

[10] Mrinal R. Bachute and Javed M. Subhedar. Autonomous driving architectures: Insights of machine learning and deep learning algorithms. *Machine Learning with Applications*, 6:100164, 2021. ISSN 2666-8270. doi: https://doi.org/10.1016/j.mlwa.2021.100164. URL https://www.sciencedirect.com/science/article/pii/S2666827021000827.

[11] Pierre Baldi, Kyle Cranmer, Taylor Faucett, Peter Sadowski, and Daniel Whiteson. Parameterized neural networks for high-energy physics. *The European Physical Journal C*, 76(5), Apr 2016. ISSN 1434-6052. doi: 10.1140/epjc/s10052-016-4099-4. URL http://dx.doi.org/10.1140/epjc/s10052-016-4099-4.

[12] Andrew O. Bazarko. Miniboone: Status of the booster neutrino experiment. *Nuclear Physics B - Proceedings Supplements*, 91(1-3):210–215, Jan 2001. ISSN 0920-5632. doi: 10.1016/s0920-5632(00)00943-9. URL http://dx.doi.org/10.1016/S0920-5632(00)00943-9.

[13] Christopher M. Bender, Yang Li, Yifeng Shi, Michael K. Reiter, and Junier Oliva. Defense through diverse directions. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 756–766. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/bender20a.html.

[14] Christopher M. Bender, Kevin O'Connor, Yang Li, Juan Garcia, Manzil Zaheer, and Junier Oliva. Exchangeable generative models with flow scans. 34:10053–10060, Apr. 2020. doi: 10.1609/aaai.v34i06.6562. URL https://ojs.aaai.org/index.php/AAAI/article/view/6562.

[15] Christopher M. Bender, Patrick Emmanuel, Michael M. Reiter, and Junier Oliva. Practical integration via separable bijective networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=NlObxR0rosG.

[16] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

[17] José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

[18] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

[19] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[20] P. Blanchard, R.L. Devaney, and G.R. Hall. *Differential Equations*. Thomson Brooks/Cole, 2006. ISBN 9780495012658. URL https://books.google.com/books?id=mwxX2pv9UvYC.

[21] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[22] Dankmar Böhning, Wilfried Seidel, Marco Alfò, Bernard Garel, Valentin Patilea, and Gunther Walther. Advances in mixture models. *Comput. Stat. Data Anal.*, 51:5205–5210, 2007.

[23] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[24] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. 2018.

[25] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[26] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

[27] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[28] Chao Chen, Guanbin Li, Ruijia Xu, Tianshui Chen, Mei Wang, and L. Lin. Clusternet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4989–4997, 2019.

[29] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[30] Ricky T. Q. Chen, Jens Behrmann, David K Duvenaud, and Joern-Henrik Jacobsen. Residual flows for invertible generative modeling. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/5d0d5594d24f0f955548f0fc0ff83d10-Paper.pdf.

[31] Tian Qi Chen, Yulia Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

[32] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016. URL http://arxiv.org/abs/1604.06174.

[33] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[34] Moustapha Cissé, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *ICML*, 2017.

[35] Heather Cody, Hongbin Gu, Brent Munsell, Sun Kim, Martin Styner, Jason Wolff, Jed Elison, Meghan Swanson, Hongtu Zhu, Kelly Botteron, Louis Collins, John N. Constantino, Stephen R. Dager, Annette M. Estes, Alan Evans, Vladimir Fonov, Guido Gerig, Penelope Kostopoulos, Robert C. McKinstry, and Core H. Gu. Early brain development in infants at high risk for autism spectrum disorder. *Nature*, 542:348–351, 02 2017. doi: 10.1038/nature21369.

[36] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[37] Adnan Darwiche. A differential approach to inference in bayesian networks. 50(3), 2003. ISSN 0004-5411. doi: 10.1145/765568.765570. URL https://doi.org/10.1145/765568.765570.

[38] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Li Chen, Michael E Kounavis, and Duen Horng Chau. Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression. *arXiv preprint arXiv:1705.02900*, 2017.

[39] Ingrid Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, USA, 1992. ISBN 0898712742.

[40] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[42] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*, 2018.

[43] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous. Tensorflow distributions. *CoRR*, abs/1711.10604, 2017. URL http://arxiv.org/abs/1711.10604.

[44] Fei Ding, Feng Luo, and Yin Yang. Double cycle-consistent generative adversarial network for unsupervised conditional generation, 2019. URL https://arxiv.org/abs/1911.05210.

[45] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *CoRR*, abs/1410.8516, 2014.

[46] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.

[47] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HkpbnH9lx.

[48] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980. ISSN 0377-0427. doi: https://doi.org/10.1016/0771-050X(80)90013-3. URL https://www.sciencedirect.com/science/article/pii/0771050X80900133.

[49] DC Dowson and BV666017 Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

[50] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/7ac71d433f282034e088473244df8c02-Paper.pdf.

[51] Harrison Edwards and Amos Storkey. Towards a neural statistician. In *5th International Conference on Learning Representations (ICLR 2017)*, 2 2017.

[52] Hamid Eghbal-zadeh, W. Zellinger, and G. Widmer. Mixture density generative adversarial networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5813–5822, 2019.

[53] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. In *Advances in Neural Information Processing Systems*, pages 3910–3920, 2018.

[54] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 1625–1634. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00175. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Eykholt_Robust_Physical-World_Attacks_CVPR_2018_paper.html.

[55] et. al. Falcon, WA. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[56] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610 (7930):47–53, Oct 2022. ISSN 1476-4687. doi: 10.1038/s41586-022-05172-4. URL https://doi.org/10.1038/s41586-022-05172-4.

[57] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176–190, 2008. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2007.05.018. URL https://www.sciencedirect.com/science/article/pii/S0031320307002580.

[58] Ming Ming Gao, Chan Tai-hua, and Xiang xiang Gao. Application of gaussian mixture model genetic algorithm in data stream clustering analysis. *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, 3:786–790, 2010.

[59] Alan E Gelfand, Athanasios Kottas, and Steven N MacEachern. Bayesian nonparametric spatial modeling with dirichlet process mixing. *Journal of the American Statistical Association*, 100(471):1021–1035, 2005. doi: 10.1198/016214504000002078. URL https://doi.org/10.1198/016214504000002078.

[60] Efstathios D. Gennatas, Jerome H. Friedman, Lyle H. Ungar, Romain Pirracchio, Eric Eaton, Lara G. Reichmann, Yannet Interian, José Marcio Luna, Charles B. Simone, Andrew Auerbach, Elier Delgado, Mark J. van der Laan, Timothy D. Solberg, and Gilmer Valdes. Expert-augmented machine learning. *Proceedings of the National Academy of Sciences*, 117(9):4571–4577, 2020. doi: 10.1073/pnas.1906831117. URL https://www.pnas.org/doi/abs/10.1073/pnas.1906831117.

[61] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked Autoencoder for Distribution Estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR.

[62] Zoubin Ghahramani and Geoffrey E. Hinton. The EM algorithm for mixtures of factor analyzers. 1996.

[63] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[64] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

[65] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. Deep learning. *Nature*, 521: 436–444, 2015.

[66] Dilan Görür and Carl Edward Rasmussen. Dirichlet process gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25:653–664, 2010.

[67] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJxgknCcK7.

[68] Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1242–1250, Bejing, China, 22–24 Jun 2014. PMLR.

[69] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, Apr 2020. ISSN 1556-4967. doi: 10.1002/rob.21918. URL http://dx.doi.org/10.1002/rob.21918.

[70] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[71] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/gutmann10a.html.

[72] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016. URL https://arxiv.org/abs/1609.09106.

[73] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning. 2001.

[74] Tamir Hazan, George Papandreou, and Daniel Tarlow. *Perturbations, Optimization, and Statistics*. MIT Press, 2016.

[75] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[76] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[78] Kilian Hendrickx, Lorenzo Perini, Dries Van der Plas, Wannes Meert, and Jesse Davis. Machine learning with a reject option: A survey, 2021.

[79] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *Proceedings of International Conference on Learning Representations*, 2017.

[80] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyxCxhRcY7.

[81] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

[82] Hajo Holzmann, Axel Munk, and Tilmann Gneiting. Identifiability of finite mixtures of elliptical distributions. *Scandinavian Journal of Statistics*, 33, 2006.

[83] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

[84] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. URL http://dblp.uni-trier.de/db/journals/nn/nn2.html#HornikSW89.

[85] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2): 433–450, 1990. doi: 10.1080/03610919008812866. URL https://doi.org/10.1080/03610919008812866.

[86] Pavel Izmailov, P. Kirichenko, Marc Finzi, and A. Wilson. Semi-supervised learning with normalizing flows. *ArXiv*, abs/1912.13025, 2020.

[87] Priyank Jaini, Pascal Poupart, and Yaoliang Yu. Deep homogeneous mixture models: Representation, separation, and approximation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing*

Systems, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/c5f5c23be1b71adb51ea9dc8e9d444a8-Paper.pdf.

[88] Tony Jebara, Risi Kondor, and Andrew Howard. Probability product kernels. *J. Mach. Learn. Res.*, 5:819–844, December 2004. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1005332.1016786.

[89] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *IJCAI*, 2017.

[90] Lianmeng Jiao, Thierry Denoeux, Zhunga Liu, and Quan Pan. Egmm: an evidential version of the gaussian mixture model for clustering. *ArXiv*, abs/2010.01333, 2020.

[91] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873): 583–589, Aug 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-03819-2. URL https://doi.org/10.1038/s41586-021-03819-2.

[92] Diederik Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations, ICLR 2014*, 12 2014.

[93] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[94] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

[95] Diederik P. Kingma, Danilo Jimenez Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. *CoRR*, abs/1406.5298, 2014. URL http://arxiv.org/abs/1406.5298.

[96] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018.

[97] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018.

[98] Jan J. Koenderink. *Solid Shape*. MIT Press, Cambridge, MA, USA, 1990. ISBN 0-262-11139-X.

[99] Iryna Korshunova, Jonas Degrave, Ferenc Huszar, Yarin Gal, Arthur Gretton, and Joni Dambre. Bruno: A deep recurrent model for exchangeable data. In *Advances in Neural Information Processing Systems*, pages 7190–7198, 2018.

[100] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[101] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, CIFAR-10 (Canadian Institute for Advanced Research), 2009. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[102] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[103] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 29–37, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[104] Marc T. Law, Raquel Urtasun, and Richard S. Zemel. Deep spectral clustering learning. In *ICML*, 2017.

[105] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

[106] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[107] Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

[108] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer. *arXiv preprint arXiv:1810.00825*, 2018.

[109] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *International Conference on Learning Representations*, 2018.

[110] Yang Li, Shoaib Akbar, and Junier Oliva. ACFlow: Flow models for arbitrary conditional likelihoods. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th*

International Conference on Machine Learning, volume 119 of *Proceedings of Machine Learning Research*, pages 5831–5841. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/li20a.html.

[111] Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1VGkIxRZ.

[112] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 369–385, 2018.

[113] Xuanqing Liu, Yao Li, Chongruo Wu, and Cho-Jui Hsieh. Adv-bnn: Improved adversarial defense through robust bayesian neural network. *arXiv preprint arXiv:1810.01279*, 2018.

[114] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[115] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations, 2017. URL https://arxiv.org/abs/1710.10121.

[116] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

[117] Ahsan Mahmood, Junier Oliva, and Martin Andreas Styner. Multiscale score matching for out-of-distribution detection. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=xoHdgbQJohv.

[118] Louis Mahon and Thomas Lukasiewicz. Selective pseudo-label clustering. In Stefan Edelkamp, Ralf Möller, and Elmar Rueckert, editors, *KI 2021: Advances in Artificial Intelligence*, pages 158–178, Cham, 2021. Springer International Publishing. ISBN 978-3-030-87626-5.

[119] Alexander Meinke and Matthias Hein. Towards neural networks that provably know when they don't know. 2020.

[120] Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022. URL https://christophm.github.io/interpretable-ml-book.

[121] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[122] S. Mukherjee, Himanshu Asnani, Eugene Lin, and S. Kannan. Clustergan : Latent space clustering in generative adversarial networks. *ArXiv*, abs/1809.03627, 2019.

[123] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5):145:1–145:19, October 2019. ISSN 0730-0301. doi: 10.1145/3341156. URL http://doi.acm.org/10.1145/3341156.

[124] Peter Müller and Fernando A. Quintana. Nonparametric Bayesian Data Analysis. *Statistical Science*, 19(1):95 – 110, 2004. doi: 10.1214/088342304000000017. URL https://doi.org/10.1214/088342304000000017.

[125] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using typicality. *arXiv preprint arXiv:1906.02994*, 2019.

[126] Eric T. Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Görür, and Balaji Lakshminarayanan. Hybrid models with deep and invertible features. *Proceedings of the 36th International Conference on Machine Learning*, 2019. URL http://arxiv.org/abs/1902.02767.

[127] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE Access*, 7: 19143–19165, 2019. doi: 10.1109/ACCESS.2019.2896880.

[128] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[129] Jiquan Ngiam, Zhenghao Chen, Pang Wei Koh, and Andrew Y. Ng. Learning deep energy models. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 1105–1112, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

[130] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.1.0. *CoRR*, 1807.01069, 2018. URL https://arxiv.org/pdf/1807.01069.

[131] Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3898–3907, 2018.

[132] Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3898–3907, 2018.

[133] Lindsay C. Page and Hunter Gehlbach. How an artificially intelligent virtual assistant helps students navigate the road to college. *AERA Open*, 3(4):2332858417749220, Oct 2017. ISSN 2332-8584. doi: 10.1177/2332858417749220. URL https://doi.org/10.1177/2332858417749220.

[134] Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. *CoRR*, abs/1901.08846, 2019. URL http://arxiv.org/abs/1901.08846.

[135] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/6c1da886822c67822bcf3679d04369fa-Paper.pdf.

[136] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[137] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[138] Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van Den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7563–7574. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/peharz20a.html.

[139] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 334–344. PMLR, 22–25 Jul 2020. URL https://proceedings.mlr.press/v115/peharz20a.html.

[140] G Peter Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27(2):192–203, 1978. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(78)90004-9. URL https://www.sciencedirect.com/science/article/pii/0021999178900049.

[141] Guilherme G. P. Freitas Pires and Mário A. T. Figueiredo. Variational mixture of normalizing flows. *ArXiv*, abs/2009.00585, 2020.

[142] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690, 2011. doi: 10.1109/ICCVW.2011.6130310.

[143] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[144] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[145] S Hamid Rezatofighi, Anton Milan, Ehsan Abbasnejad, Anthony Dick, Ian Reid, et al. Deepsetnet: Predicting sets with deep neural networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 5257–5266. IEEE, 2017.

[146] Eitan Richardson and Yair Weiss. On gans and gmms. In *NeurIPS*, 2018.

[147] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[148] Matthias Rupp. Machine learning for quantum mechanics in a nutshell. *International Journal of Quantum Chemistry*, 115(16):1058–1073, 2015. doi: https://doi.org/10.1002/qua.24954. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/qua.24954.

[149] J. Rynkiewicz. General bound of overfitting for mlp regression models. *Neurocomputing*, 90:106–110, 2012. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2011.11.028. URL https://www.sciencedirect.com/science/article/pii/S0925231212001865. Advances in artificial neural networks, machine learning, and computational intelligence (ESANN 2011).

[150] Uri Shaham, Kelly Stanton, Haochao Li, Boaz Nadler, Ronen Basri, and Yuval Kluger. Spectralnet: Spectral clustering using deep neural networks. *ArXiv*, abs/1801.01587, 2018.

[151] L.F. Shampine. Solving odes and ddes with residual control. *Applied Numerical Mathematics*, 52(1):113–127, 2005. ISSN 0168-9274. doi: https://doi.org/10.1016/j.apnum.2004.07.003. URL https://www.sciencedirect.com/science/article/pii/S0168927404001187.

[152] Mahmood Sharif, Lujo Bauer, and Michael K. Reiter. n-ML: Mitigating adversarial examples via ensembles of topologically manipulated classifiers. arXiv preprint 1912.09059, December 2019. URL https://arxiv.org/abs/1912.09059.

[153] Yash Sharma and Pin-Yu Chen. Attacking the madry defense model with l_1-based adversarial examples. *arXiv preprint arXiv:1710.10733*, 2017.

[154] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[155] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[156] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[157] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/3001ef257407d5a371a96dcd947c7d93-Paper.pdf.

[158] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

[159] W.A. Strauss. *Partial Differential Equations: An Introduction*. Wiley, 2007. ISBN 9780470054567. URL https://books.google.com/books?id=PihAPwAACAAJ.

[160] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[161] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[162] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

[163] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SyxAb30cY7.

[164] Ch. Tsitouras. Runge–kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Computers & Mathematics with Applications*, 62(2):770–775, 2011. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2011.06.002. URL https://www.sciencedirect.com/science/article/pii/S0898122111004706.

[165] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2175–2183. Curran Associates, Inc., 2013.

[166] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17 (205):1–37, 2016.

[167] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, koray kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/b1301141feffabac455e1f90a7de2054-Paper.pdf.

[168] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[169] Eric Veach. *Robust Monte Carlo methods for light transport simulation*. Stanford University, 1998.

[170] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[171] C. Viroli and G. McLachlan. Deep gaussian mixture models. *Statistics and Computing*, 29: 43–51, 2019.

[172] Peide Wang. Research on comparison of lidar and camera in autonomous driving. *Journal of Physics: Conference Series*, 2093(1):012032, nov 2021. doi: 10.1088/1742-6596/2093/1/012032. URL https://dx.doi.org/10.1088/1742-6596/2093/1/012032.

[173] Siyue Wang, Xiao Wang, Pu Zhao, Wujie Wen, David Kaeli, Peter Chin, and Xue Lin. Defensive dropout for hardening deep neural networks under adversarial attacks. In *Proceedings of the International Conference on Computer-Aided Design*, pages 1–8, 2018.

[174] Xiao Wang, Siyue Wang, Pin-Yu Chen, Yanzhi Wang, Brian Kulis, Xue Lin, and Peter Chin. Protecting neural networks with hierarchical random switching: Towards better robustness-accuracy trade-off for stochastic defenses. *arXiv preprint arXiv:1908.07116*, 2019.

[175] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger B. Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *CoRR*, abs/1803.04386, 2018. URL http://arxiv.org/abs/1803.04386.

[176] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. pages 1912–1920, 06 2015. doi: 10.1109/CVPR.2015.7298801.

[177] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[178] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

[179] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 501–509, 2019.

[180] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, 2016.

[181] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training, 2020. URL `https://openreview.net/forum?id=ByeL1R4FvS`.

[182] Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela van der Schaar. Vime: Extending the success of self- and semi-supervised learning to tabular domain. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11033–11043. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/7d97667a3e056acab9aaf653807b4a03-Paper.pdf`.

[183] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5694–5703, 2018.

[184] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

[185] Manzil Zaheer, Chun-Liang Li, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov. Point cloud gan. *arXiv preprint arXiv:1810.05795*, 2018.

[186] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 39–49, 2017.

[187] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient GAN-based anomaly detection. In *6th International Conference on Learning Representations, ICLR 2018, Workshop Track*, 2018. URL `http://arxiv.org/abs/1802.06222`.

[188] Dejiao Zhang, Y. Sun, B. Eriksson, and L. Balzano. Deep unsupervised clustering using mixture of autoencoders. *ArXiv*, abs/1712.07788, 2017.

[189] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1Ddp1-Rb.

[190] Huan Zhang, Hongge Chen, Zhao Song, Duane Boning, Inderjit S Dhillon, and Cho-Jui Hsieh. The limitations of adversarial training and the blind-spot attack. *arXiv preprint arXiv:1901.04684*, 2019.