Exchangeable Generative Models with Flow Scans

Christopher M. Bender
UNC Chapel Hill
bender@cs.unc.edu

Juan Jose Garcia
Escuela Superior Politecnica del Litoral
bambole8@gmail.com

Kevin O'Connor
UNC Chapel Hill
koconn@live.unc.edu

Manzil Zaheer
Google AI
manzilz@google.com

Junier Oliva
UNC Chapel Hill
joliva@cs.unc.edu

Abstract

In this work, we fill a void in machine learning and statistical methodology by developing generative density estimation techniques for exchangeable, non-*i.i.d.* data. The proposed framework, FlowScan, combines invertible flow transformations with a sorted scan to flexibly model the data while preserving exchangeability. Unlike most existing methods, this approach exploits the intradependencies within sets and graphs to learn both global and local structure. Importantly, FlowScan achieves this without resorting to averaging over all possible permutations. We achieve new state of the art performance on point cloud modeling.

1 Introduction

Modeling non-*i.i.d.*, exchangeable data is a fundamental problem in machine learning. Exchangeable data structures (such as sets and graphs) with complicated intrinsic relationships are ubiquitous in our world. For instance, members of a social network behave differently depending on the behavior of other members. Similarly, the forces of galaxies within a cluster yield a rich web of relations depending on the characteristics of each constituent piece of matter. Notwithstanding the abundance of such data, the bulk of existing approaches either ignore the relation between points (*i.i.d.* methods) [17, 24] or model dependencies in a manner that depends on inherent orderings (sequential methods) [18, 23]. In this work we improve upon existing methodology by developing generative models that are inherently exchangeable and flexibly model dependencies among points.

The unorderedness of collections like sets and graphs is captured probabilistically through the notion of *exchangeability*. A set of points $\{x_j\}_{j=1}^n$ with probability density $p(\cdot)$ is called exchangeable if

$$p(x_1, ..., x_n) = p(x_{\pi_1}, ..., x_{\pi_n}) \tag{1}$$

for every permutation π. In this work, we focus primarily on models for sets of points with this property (with a secondary focus on exchangeable graphs).[1] That is, rather than modeling a single data point $x \in \mathbb{R}^d$ as $p(x)$ given a training-set of points $\mathcal{D} = \{x_i \in \mathbb{R}^d\}_{i=1}^N$, we look to model an entire set of points $p(\mathcal{X}) = p(\{x_j\}_{j=1}^n)$ given a training-set of *sets* $\mathcal{D} = \{\mathcal{X}_i\}_{i=1}^N$, where $\mathcal{X}_i = \{x_{i,j} \in \mathbb{R}^d\}_{j=1}^{n_i}$ is a set of n_i points (see Fig. 1). Since a set has no intrinsic ordering, any learned model $p(\cdot)$ should be exchangeable. However, modeling such data in an exchangeable way with traditional methods is not straightforward. In most implementations, a set $\mathcal{X} = \{x_j\}_{j=1}^n$ will be observed in some particular order $(x_1, \ldots, x_n)$, for instance if it is stored as a matrix $\mathbf{x} \in \mathbb{R}^{n \times d}$ (according to the rows of $\mathbf{x}$).

[1] We use the following notational conventions: calligraphic capitals, e.g. $\mathcal{X}$, corresponds to a set of multiple points; bold lowercase, e.g. $\mathbf{x}$, corresponds to a matrix representation of a set; math italics, e.g. x, corresponds to a multivariate point; and $x_i^{(j)}$ corresponds to the jth dimension of the ith point of a set.

Preprint. Under review.

Figure 1: 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 $p(\mathcal{X}_i)$; that is we make a generative model that generates distinct sets.

Traditional models which are not exchangeable will tend to give different results for distinct orderings of the data despite the fact that these inputs correspond to the same unordered set.

Below, we expound upon the motivations, challenges, and details of our methodology for creating non-*i.i.d.* likelihoods that are exchangeable while still modeling intradependencies among points. We introduce a framework – *FlowScan* – which yields a principled relationship between a prescribed scan through elements (possibly in a transformed space) and the underlying exchangeable likelihood. Furthermore, we develop architectures to apply our proposed FlowScan to both set and graph data.

Main Contributions. 1) We introduce a scanning based technique for modeling exchangeable data. We prescribe a way to scan through a set of points and show a direct and analytical relationship between the likelihood of the scanned covariates and the underlying exchangeable likelihood. 2) We show that transforming points with an equivariant change of variables allows for scanning sets in a different space, FlowScan. 3) We develop an autoregressive likelihood and transformations to exploit the structure gained with FlowScan. 4) We extend FlowScan to work over adjacency matrices. 5) We illustrate these methods in synthetic and real-world point cloud data and synthetic graph data.

2 Generating Generative Processes with Exchangeability

There are several key tasks that make use of exchangeable modeling of sets. One direct application is modeling sequential data observed in a randomly permuted, scrambled manner. In this work we focus on another important task: using exchangeability to model a generative process over multiple distributions. In effect, we learn a generative model *over generative models*. To see how modeling set data fits into this task, consider a set-generative process, $p(\mathcal{X} \mid \phi)$, given *latent* "parameters" ϕ. 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). Given a prior generative process, $p_\Phi(\phi)$, over parameters, a set is generated from this model by first drawing a parameter $\phi \sim p_\Phi(\cdot)$ and then drawing a set $\mathcal{X} \sim p(\cdot \mid \phi)$. In the case when the point generative process is conditionally *i.i.d.*, $p(\mathcal{X} \mid \phi) = \prod_{k=1}^n p(x_k \mid \phi)$, the complete likelihood of the set is:

$$p(\mathcal{X}) = \int p_\Phi(\phi) \prod_{j=1}^{n} p(x_j \mid \phi) \, \mathrm{d}\phi, \tag{2}$$

and is in the same vein as De Finetti's theorem [2]. This type of data generating model is especially applicable for point cloud data. In particular, points in each individual set $\mathcal{X}_i$ are drawn *i.i.d.* from 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.

It is important to emphasize that the complete process above (2) does *not* produce *i.i.d.* sets even in the case where $p(\mathcal{X} \mid \phi)$ is conditionally *i.i.d.*. Consider the likelihood of a single point x_k given a disjoint subset $S \subset \mathcal{X} \setminus \{x_k\}$:

$$p(x_k \mid S) = \int p_\Phi(\phi \mid S) \, p(x_k \mid S, \phi) \, \mathrm{d}\phi = \int p_\Phi(\phi \mid S) \, p(x_k \mid \phi) \, \mathrm{d}\phi \neq p(x_k). \tag{3}$$

The likelihood of x_k depends on other points in $\mathcal{X}$ via the posterior $p_\Phi(\phi \mid S)$, which accounts for what point generative processes, $p(\cdot \mid \phi)$, were likely to have generated S. Hence, the complete generative process (2) is not marginally *i.i.d.* since points of the same set are informative (intradependent) for each other, even though the point generative process $p(\cdot \mid \phi)$ may be conditionally *i.i.d.*. For this reason, *i.i.d.* methods are insufficient. One approach to estimating (2) is to estimate the latent parameters ϕ, for instance, in a variational approach [7]. However, this approach can only yield a lower bound for likelihoods, and introduces the added task of encoding sets, which is in itself an active research area. Instead, we effectively marginalize out the point generative processes that produce set elements to directly get a likelihood over observed point sets $\mathbf{x}$. Empirical results

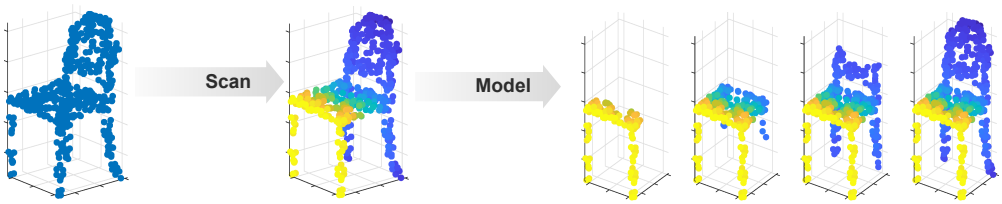


Figure 2: Illustration of our proposed method. First, input sets are scanned (in a possibly transformed space). After, the scanned covariates are modeled (possibly in a autoregressive fashion, as shown).

(detailed in Sec. 6) also indicate that directly estimating the marginal set likelihood yields better performance in terms of likelihoods and samples.

3 Challenges

There are several challenges associated with exchangeable generative modeling. One can achieve exchangeability by treating points in the set as *i.i.d.*, but such models may be severely biased for sets with intradependencies among points. To avoid this simpification, techniques often shoehorn invariances to observed orderings by feeding randomly permuted data into sequential models [18, 23]. Such approaches attempt to average out the likelihood of the model over all permutations:

$$p(\{x_1, \ldots, x_n\}) = \frac{1}{n!} \sum_{\pi} p_{\text{seq}}(x_{\pi_1}, \ldots, x_{\pi_n}), \tag{4}$$

where p_{seq} 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. This is especially true for unseen test data [21].

These shortcomings of existing methods expose two challenges in modeling exchangeable data: identifying what order to analyze data in and determining what operation to use to induce an invariant likelihood. Instead of attempting to wash out the effect of order in an architecture as in (4), we propose to embrace a specific ordering through a scan over elements in this order. However, modeling an exchangeable likelihood (1) through a scanned order also presents challenges, as follows:

- **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 $\mathcal{X} \mapsto (x_{[1]}, \ldots, x_{[n]})$.
- **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 correlations among elements which need to be modeled successfully.

We also note that although recent work [15, 17, 24, 10] has explored neural architectures for constructing permutation invariant set embeddings, this line of work is not a direct solution to our generative modeling task. A permutation invariant embedding, $\varphi(\mathbf{x}) \in \mathbb{R}^s$, will use global-pooling operations so that $\varphi(\Gamma\mathbf{x}) = \varphi(\mathbf{x})$ for all permutation operations Γ. While these embeddings produce features that are useful for (typically supervised) downstream tasks, *the embeddings themselves will not result in valid likelihoods*. This is because $\varphi(\mathbf{x}) \in \mathbb{R}$ will not integrate to 1 for set embeddings [15, 17, 24] and are thus not a direct solution for our task of unsupervised generative modeling.

4 Methods

Now, we develop our FlowScan approach, illustrating how we address the challenges discussed above. We show that there is a direct (and analytical) relationship between the likelihood of elements in this scanned order and underlying exchangeable likelihood.

4.1 Invariance Through a Sorted Scan

In this section, we show that we can model an exchangeable likelihood on a set of points via a sorted scan operation. 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 according to the non-exchangeable (ordered) likelihood of the points in a sorted order $p_s(x_{[1]}, \dots, x_{[n]})$ as:

$$p_e(x_1, \dots x_n) = \frac{1}{n!} p_s(x_{[1]}, \dots, x_{[n]}), \tag{5}$$

where $x_{[j]}$ is the jth point in the sorted order. Note, we take the underlying exchangeable likelihoods p_e to be continuous and non-degenerate (e.g. such that $\forall j \in \{1, \dots, d\}$ $\Pr[x_1^{(j)} \neq x_2^{(j)} \neq \dots \neq x_n^{(j)}] = 1$). Equation (5) may be derived with a variant of the change of variables formula [3] which 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:

$$f(z) = \sum_{j=1}^{M} \left| \det \frac{\mathrm{d} q_j^{-1}}{\mathrm{d} z} \right| p(q_j^{-1}(z)). \tag{6}$$

For the moment, consider sorting points $\{x_j\}_{j=1}^n$ according to the first dimension. That is, $x_{[1]}, \dots, x_{[n]}$ (5) is such that $x_{[1]}^{(1)} < \dots < x_{[n]}^{(1)}$. Essentially, when modeling these sorted points we are applying a transformation of variables $s : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$, $s(x_1, \dots, x_n) = (x_{[1]}, \dots, x_{[n]})$, that sort the points x_j. The transformation s is one-to-one on the partitions of the input space $\mathbb{R}^{n \times d}$ according to the relative order of points. That is, we may partition the input space according to the permutation that would sort the data: $\mathcal{A}_\pi = \{\mathbf{x} \in \mathbb{R}^{n \times d} \mid x_{\pi_1}^{(1)} < x_{\pi_2}^{(1)} < \dots < x_{\pi_n}^{(1)}\}$. We may invert s in $\mathcal{A}_\pi$ via the inverse permutation matrix of π, Γ_π^{-1}. Let Π be the set of all permutations, then:

$$p_s(s(\mathbf{x})) \overset{*}{=} \sum_{\pi \in \Pi} \left| \Gamma_\pi^{-1} \right| p_e(\Gamma_\pi^{-1} s(\mathbf{x})) \overset{**}{=} n! \, p_e(\mathbf{x}), \tag{7}$$

where (*) follows from (6) 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, $p_e(\mathbf{x}) = \frac{1}{n!} p_s(s(\mathbf{x}))$ from (5). Consequently, the exchangeable likelihood may be estimated via an approximation of the scanned covariates: $p_e(\mathbf{x}) \approx \frac{1}{n!} \hat{p}_s(s(\mathbf{x}))$. Given that 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.* Trivially, similar arguments also hold when sorting according to dimension other than the first. In fact, it is possible to sort in an appropriately transformed space of x_j, rather than any native dimension itself. To this end, we show next how exhangeability is preserved when using permutation equivariant transformations, allowing one to transform, scan, and then model points in a set.

4.2 Preserving Exchangeability with Equivariant Flow Transformations

The success of "flow models," have proven the effectiveness of changes of variables when applied to generative models [5, 6, 11]. Using the change of variables formula, one can approximate the likelihood of a d dimensional distribution over real-valued covariates $x = (x^{(1)}, ..., x^{(d)}) \in \mathbb{R}^d$, in terms of an invertible transformation of variables $\hat{q}(x)$ and an estimated base distribution $\hat{f}$ as:

$$\hat{p}(x^{(1)}, ..., x^{(d)}) = \left| \det \frac{\mathrm{d} \hat{q}}{\mathrm{d} x} \right| \hat{f}(\hat{q}(x)). \tag{8}$$

Here $|\det \frac{\mathrm{d}\hat{q}}{\mathrm{d}x}|$ is the Jacobian of the transformation $\hat{q}$. Often, the base distribution is a standard Gaussian. However, [16] recently showed that performance may be improved with a more flexible base distribution on transformed covariates such as an autoreggressive density [8, 9, 14, 19, 20].

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 appropriate, transformed space. However, one must take care to

preserve exchangeability when transforming the data. For instance, naively applying an autoregressive change of variables on points will be sensitive to the order that the elements in $\mathbf{x}$ were observed in and will result in a space that is no longer exchangeable. Below, we show that a simple property – equivariance – is enough to ensure that exchangeability is preserved after the transformation, *allowing one to model set data in a transformed space*.

Let $\hat{q} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$ be a *permutation equivariant*, invertible transformation [24]. That is, for all permutation operators, Γ, we have that $\hat{q}(\Gamma \mathbf{x}) = \Gamma \hat{q}(\mathbf{x})$. Additionally, let the base distribution be permutation invariant: $\forall \Gamma$, $\hat{f}(\Gamma \mathbf{x}) = \hat{f}(\mathbf{x})$. It is straightforward to show that $\hat{p}(\mathbf{x}) = \left|\det \frac{\mathrm{d}\hat{q}}{\mathrm{d}\mathbf{x}}\right| \hat{f}(\hat{q}(\mathbf{x}))$ results in a permutation invariant likelihood. 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.

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 depend on other points in the set while still preserving equivariance. For completeness, we include some examples in App. E.

We combine an equivariant flow transformation with a sorted scan, a *flow scan*, to model set data. We model the resulting covariates as described below.

4.3 FlowScan Architecture

An equivariant flow transformation allows us to model sets in a more expressive space according to an invariant base likelihood. Hence, after an equivariant flow transformation, we scan through points in the new space to model the base distribution. The estimated exchangeable likelihood that results is:

$$\hat{p}_{\text{fs}}(\mathbf{x}) = \frac{1}{n!} \left|\det \frac{\mathrm{d}\hat{q}_{\text{e}}}{\mathrm{d}x}\right| \hat{p}_s(s(\hat{q}_{\text{e}}(\mathbf{x}))), \tag{9}$$

where $\hat{q}$ and $\hat{p}_s$ are the estimated (via maximum likelihood) equivariant flow transformation and sorted flow scan covariate likelihood, respectively. Note that $\hat{p}_s$ is a non-exchangeable likelihood. Furthermore, our FlowScan imparts structure into resulting covariates that we may exploit as follows: use an autoregressive likelihood to directly model the dependencies on points; also, transform the sets by exploiting correspondence in adjacent points. For simplicity, we take the cardinality of sets n to be even, though methods extend in a straighforward manner to the general case.

Autoreggressive Scan Likelihood Let $z = s(\hat{q}(\mathbf{x})) \in \mathbb{R}^{n \times d}$ be the scanned 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 $\hat{p}_s(\text{vec}(z))$ using a flat density estimator. However, such a flattening approach is inflexible to varying cardinalities. Furthermore, the total number of covariates, nd, may be large for sets with large cardinality or dimensionality, and a general flat model loses the context that covariates are from multiple points in some shared set. To address this, we propose using an autoregressive approach: $\hat{p}(z_k) = \prod_{k=1}^{n} \hat{p}(z_k \mid h(z_1, \ldots, z_{k-1})) = \prod_{k=1}^{n} \hat{p}(z_k \mid h_{<k})$, where $\hat{p}(z_k \mid h_{<k})$ is itself a d-dimensional density estimator (such as (8)) conditioned on a recurrent state $h_{<k}$. This proposed approach is capable of sharing parameters across the n d-dimensional likelihood estimation tasks and is more amenable to large, possibly varying, cardinalities.

Correspondence Flow Transformations In much the same way that nearby pixels are correlated in image space, nearby points will be similarly correlated in a scan space. Thus, we propose a coupling [5] invertible transformation to transform adjacent points, exploiting existing correlations among points 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. Let $\hat{q}_{\mathrm{c}} = \hat{q}_{\mathrm{odd}} \circ \hat{q}_{\mathrm{even}} \circ \ldots \circ \hat{q}_{\mathrm{odd}} \circ \hat{q}_{\mathrm{even}}$ be the composition of the correspondence coupling transformations, and let $z_c = \hat{q}_{\mathrm{c}}\left(s(\hat{q}_{\mathrm{e}}(\mathbf{x}))\right)$ be the resulting covariates from corresponding coupling transforming the flow scanned covariates, $s(\hat{q}_{\mathrm{e}}(\mathbf{x}))$. The ultimate FlowScan proposed density estimate is then:

$$\hat{p}_{\mathrm{fs}}(\mathbf{x}) = \frac{1}{n!} \left|\det \frac{\mathrm{d}\hat{q}_{\mathrm{e}}}{\mathrm{d}x}\right| \left|\det \frac{\mathrm{d}\hat{q}_{\mathrm{c}}}{\mathrm{d}x}\right| \prod_{k=1}^{n} \hat{p}(z_{c\,k} \mid h(z_{c\,1}, \ldots, z_{c\,k-1})). \tag{10}$$

After the flow equivariant transformations and scan, we further transform points with a correspondence coupling to model intradependence among points, and model the points autoregressively.

4.4 Adjacency Matrices

Much like sets, invariances are pivotal when modeling adjacency matrices. Shuffling nodes results in different adjacency matrices for the same underlying graph; consequently, we desire a likelihood p such that: $\forall \Gamma \in \Pi,\ p(A) = p(\Gamma A \Gamma^T)$, where Π is the space of all permutation matrices. Likelihoods with this property satisfy an equivalence class over node permutations for the underlying graph. The scanning paradigm easily extends to the task of modeling real-valued, weighted adjacency matrices $A \in \mathbb{R}^{n \times n}$. We take our data to be continuously distributed in a non-degenerate fashion such that $\forall i, j, k, l\ \Pr[A_{ij} \neq A_{kl}] = 1$. A natural choice of scan for nodes is one based on out-degree $o_i = \sum_{j=1}^{n} A_{ij}$. That is, we take the scanned adjacency matrix $s(A) \in \mathbb{R}^{n \times n}$ to be such that $s(A) = \Gamma_{\pi_o} A \Gamma_{\pi_o}^T$ where Γ_{π_o} is the permutation matrix for the permutation that orders out-degrees, $o_{\pi_{o\,1}} < \ldots < o_{\pi_{o\,n}}$. Similarly to the case for sets (Sec. 4.1), we may partition the input space $\mathbb{R}^{n \times n}$ into $n!$ partitions (according to relative outdegree order) such that $s(A)$ would be one-to-one. Thus we have, as before, $p(A) = \frac{1}{n!} p_s(s(A))$. Moreover, similar autoregressive models to (10) may be derived for the adjacency matrix case.

5 Related Work

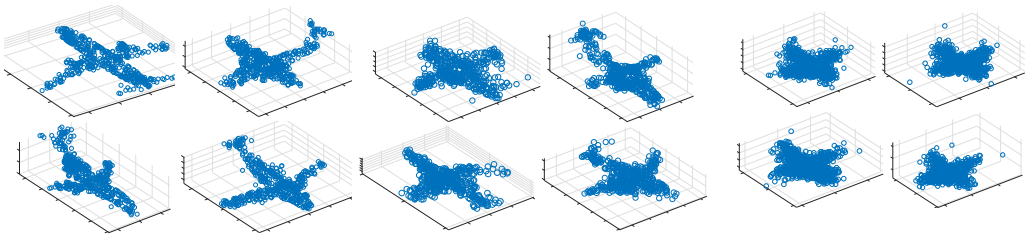
Unlike the recent surge in flexible density estimation for flat vectors with deep architectures [5, 6, 11, 14, 20, 19, 9, 8, 16], exchangeable treatments of data in ML have been limited but there are some notable examples. As mentioned above in Sec. 3, recent work [15, 17, 24] has explored neural architectures for constructing a permutation invariant set embeddings. They featurize the sets in an exchangeable fashion, that are useful for (typically supervised) downstream tasks; *but the embeddings themselves will not result in valid likelihoods*. Recently, Generative Adversarial Networks (GAN) have been explored to sample point clouds [25]. However, these methods do not provide a solution for how to estimate an exchangeable likelihood as is our focus.

A recently proposed model for exchangeable data, BRUNO [13], preserves exchangeability by performing point-wise changes of variables. The resulting points are then modeled with an exchangeable process. The Neural Statistician (NS) [7] estimates a permutation invariant code produced by an exchangeable VAE as in (2). That is, the Neural Statistician uses an encoder, called a statistics network, on the entire exchangeable set to get an approximate posterior on ϕ (2). Given the success of a point cloud autoencoder with a DeepSet network as the statistics network in [16], we consider this architecture for the variational Neural Statistician. We note that this is an especially strong baseline, and represents the state-of-the-art likelihood method for point cloud data.

6 Experiments

Below, we compare methods in exchangeable modeling tasks on both synthetic and real-world datasets. We begin with modeling exhangeable sets and estimating a likelihood $p(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^{n \times d}$. For readability we report estimated per point log likelihoods (PPLL): $\frac{1}{n} \log \hat{p}(\mathbf{x})$.

On point cloud data, we show FlowScan's efficacy both in terms of likelihood and sample quality over two state-of-the-art exchangeable likelihood models: BRUNO [13] and Neural Statistician (NS) [7] (as discussed above). For NS we report the variational lower bound on PPLL. Note, the scatter plots in Fig. 4 are *not reconstructions*, but rather are all completely synthetic generated point clouds from trained models. Implementation details can be found in App. A; code will be made public upon publication.



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

Figure 4: Generated synthetic plane point cloud samples from trained models.

6.1 Shuffled Synthetic Sequential Data

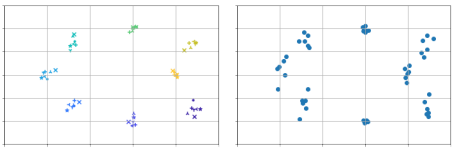


Figure 3: Left: true samples; markers and colors indicate instances and sequential order, respectively. Right: FlowScan samples.

A common scenario that leads to exchangeable data is when time has been marginalized out from a sequential model. That is, when all time-points $x_j \in \mathbb{R}^d$ of a sequence $(x_1, \ldots, x_n)$ are put into an unordered bucket $\{x_1, \ldots, x_n\}$. 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 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 dependencies we consider a sinusoidal sequence (see Fig. 3, App. B for details). We consider $n = 8$, leading to a large number, $8! = 40320$, of summands in the likelihoods.

Table 1: PPLLs on Synthetic Data

BRUNO	NS	FlowScan
-2.28	-1.07	**0.14**

Table 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. We note that we also trained a sequential model on the randomly permuted instances (and marginalizing out the permutation as in (4)); however, randomly permuting the input sequence proved to be ineffective and resulted in low test PPLLs (with severe overfitting).

6.2 ModelNet

Next, we illustrate the efficacy of our model on real world point cloud data. We consider object classes from the ModelNet dataset [22], 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 [22] consisting of *bathtub*, *bed*, *chair*, *desk*, *dresser*, *monitor*, *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*, *chair*, *guitar*, *lamp*, *plant*, *stairs*, *car*, *bed*, *table*, and *laptop* classes.

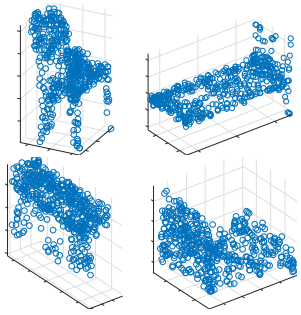


Figure 5: FlowScan samples trained on `ModelNet10`.

Table 2: ModelNet PPLLs

Dataset	BRUNO	NS	FlowScan
`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**

Results can be found in Tab. 2 and four samples from FlowScan are included in Fig. 5. 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. 4 and in App. C. In addition to training on these

ModelNet datasets, we also performed an ablation study (see App. A.1) where we see that our full architecture yields the best performance over alternatives.

6.3 Brain Data

We test FlowScan's performance on a medical imaging task in a higher dimensional setting using samples of the Caudate and Thalamus [4]. Each set contains 512 randomly sampled 7-dimensional points. The first three dimensions contain the Cartesian coordinates of the boundary (as in the ModelNet experiments). 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 [12]). Superior PPLLs and samples suggest that FlowScan seamlessly incorporates the additional geometric information to model point clouds more accurately than baseline methods. For the `Caudate` dataset we find that FlowScan achieves a PPLL of 4.87 compared to 4.49 for NS and 1.29 for BRUNO. Similarly, for the `Thalamus` dataset, FlowScan achieves a PPLL of 3.12 compared to 2.69 for NS and -0.815 for BRUNO. Samples from these models, included in App. C, indicate that FlowScan better captures the geometric features of the data than NS. Overall, these results provide evidence that our model is successful even in higher dimensional settings.

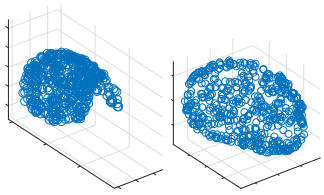


Figure 6: FlowScan samples trained on the `Caudate` and `Thalamus`.

6.4 Graphs

We now demonstrate how our model can be extended to sample exchangeable graphs. Using the approach described earlier (Sec. 4.4), we trained the model on datasets consisting entirely of grids and stars (where all other nodes connect to a center node) with 16 or 36 nodes. We measure what proportion of samples were valid after training. Here we compared to a state-of-the-art graph generator, the GraphRNN [23], which works on binary adjacency matrices. As the FlowScan model operates over real-valued adjacency matrices, we undiscretized binary adjacency matrices by centering $0, 1$ edges around $-1, 1$, respectively and adding *i.i.d.* $\mathcal{N}(0, 0.05^2)$ Gaussian noise to the adjacency matrix elements. FlowScan samples are binarized according to sign. We see competitive results using the straight forward adjacency matrix FlowScan compared to a specialized graph generative model in Tab. 3 illustrating the strength of our framework.

Table 3: Percent of valid sampled graphs.

Type	#Nodes	GraphRNN	Flow Scan
`Grids`	16	94.4%	**98.5%**
`Grids`	36	9.2%	**12.8%**
`Stars`	16	96.1%	**100.0%**
`Stars`	36	85.1%	**100.0%**

7 Conclusions

This work introduces a novel method, FlowScan, for estimating exchangeable densities. We developed a principled approach to scan through covariates and exploit dependencies among points in the same collection, all while maintaining exchangeability (a permutation invariant likelihood). This is a difficult task, where models were previously limited to either simple equivariant correlations [13], or conditionally *i.i.d.* restrictions [7] on elements. We developed an autoregressive architecture that, along with corresponding coupling transformations, exploits the structure gained in a scanned space.

We showed empirically that FlowScan's ability to model intradependencies within sets surpassed that of other state-of-the-art methods in both synthetic and real-world experiments. Quantitatively FlowScan's likelihoods were a substantial improvement (e.g. see Tab. 1, Tab. 2). Furthermore, there was a clear qualitative improvement in samples from FlowScan. For instance, in Fig. 4 we see much more fine-grain detail and variety in the generated `plane` point clouds from FlowScan. Although BRUNO was shown to be very effective at modeling exchangeable sets of similar images [13], the simple exchangeable process was unable to capture much more than the rough shape of planes. The same holds true for the `chair` dataset (App. Fig. 7), where, although NS produces nice chair point clouds, the detail and variety of the NS samples is lower than FlowScan samples. Unlike the other models, FlowScan maintains its sample quality even in diverse datasets like `ModelNet10` and `ModelNet10a` (see Fig. 5, and App. Fig. 8 and Fig. 9). Finally, we demonstrate that FlowScan can be easily extended to work with adjacency matrices, where we saw results that were competitive with those of GraphRNN, a state-of-the-art model for graphs.

References

[1] 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.

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

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

[4] 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.

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

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

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

[8] 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.

[9] 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.

[10] Maximilian Ilse, Jakub M Tomczak, and Max Welling. Attention-based deep multiple instance learning. *arXiv preprint arXiv:1802.04712*, 2018.

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

[12] Jan J. Koenderink. *Solid Shape*. MIT Press, Cambridge, MA, USA, 1990.

[13] 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.

[14] 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.

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

[16] Junier B Oliva, Avinava Dubey, Barnabás Póczos, Jeff Schneider, and Eric P Xing. Transformation autoregressive networks. *arXiv preprint arXiv:1801.09819*, 2018.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] Manzil Zaheer, Chun-Liang Li, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov. Point cloud gan. *arXiv preprint arXiv:1810.05795*, 2018.

A Experiment Details

Experiments were implemented in Tensorflow [1]. We use multiple stacked Real NVP transformations with a Gaussian exchangeable process for BRUNO. 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 [16, 7] 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 [16]; after the scan we implemented the autoregressive likelihood with a 2-layer GRU (256 units), which conditioned a TAN density [16] on points. Models were optimized for 40k iterations on TitanXP GPUs. Modelnet data was gathered as in [24], brain data was gathered as in [4]. All datasets used a random 80/10/10 train/validation/test split.

A.1 ModelNet10 Ablation Study

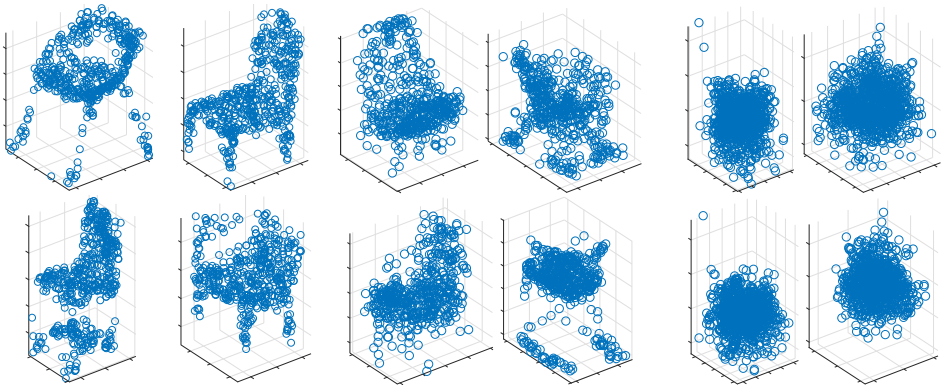
We performed an ablation study using `ModelNet10`. We begin with a full flow scan model, which performs an equivariant flow transformation, scans, does corresponding coupling transformations, and uses an auto-regressive model. Next, we omit the correspondence coupling transformation. After, we also remove 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 to note that even the bare basic scan model is still outperforming the NS likelihood bound.

B Synthetic

The synthetic data in Sec. 6.1 is generated as follows:

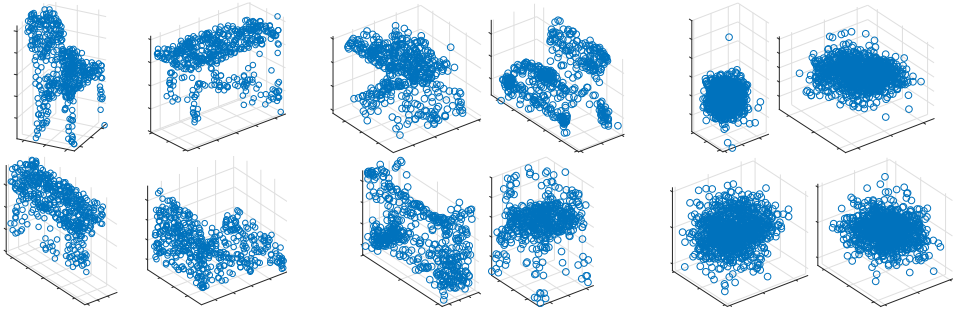
$$x_1^{(1)} \sim \mathcal{N}(2, n^{-2}) \qquad x_1^{(2)} \sim \mathcal{N}(0, n^{-2}(1 + (\pi/3)^2))$$
$$x_k^{(1)} \sim \mathcal{N}(x_1^{(1)} \cos(\pi k/n), n^{-2}) \qquad x_k^{(2)} \sim \mathcal{N}(\cos(\pi k/n + x_1^{(2)}), n^{-2})$$

C Generated Samples for Point Cloud Experiments



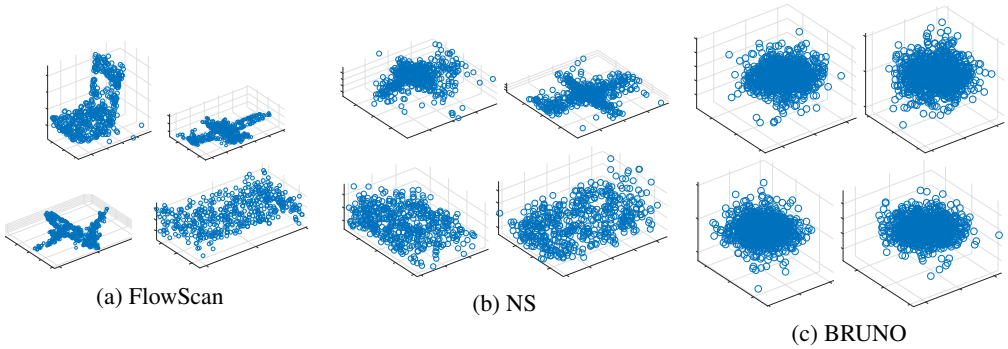
(a) FlowScan (b) NS (c) BRUNO

Figure 7: Chair Samples



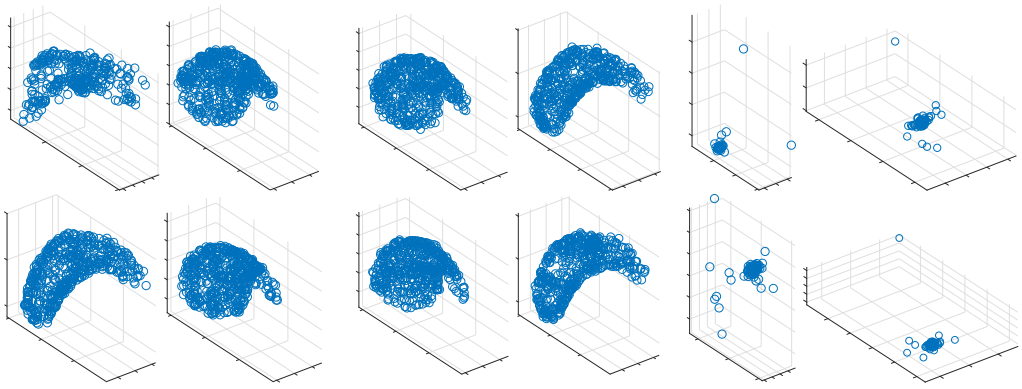
(a) FlowScan (b) NS (c) BRUNO

Figure 8: ModelNet10 Samples



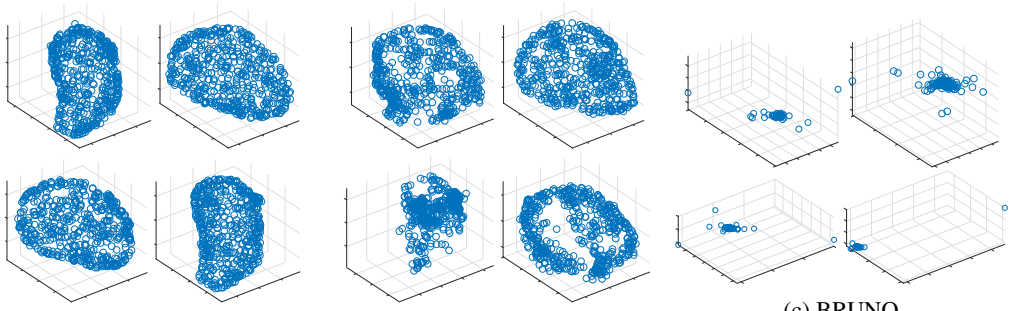
(a) FlowScan (b) NS (c) BRUNO

Figure 9: ModelNet10a Samples



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

Figure 10: Caudate Samples

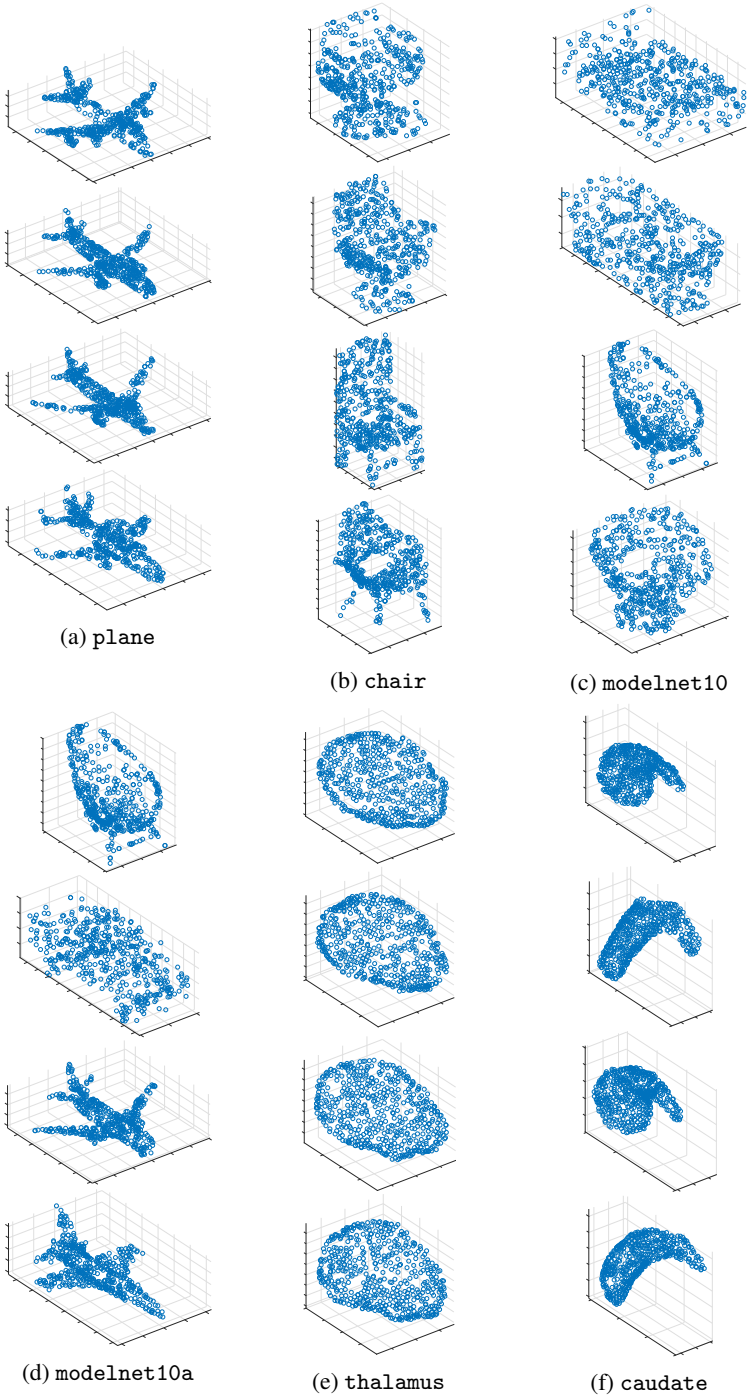


(a) FlowScan (b) NS (c) BRUNO

Figure 11: Thalamus Samples

D 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.



(a) `plane` (b) `chair` (c) `modelnet10`

(d) `modelnet10a` (e) `thalamus` (f) `caudate`

Figure 12: Training examples

E 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:

$$(x_1, ..., x_n) \mapsto (q(x_1), ..., q(x_n)). \tag{11}$$

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

$$(x_1, ..., x_n) \mapsto (q(x_1, \mathbf{x}), ..., q(x_n, \mathbf{x})). \tag{12}$$

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

E.1 Linear Permutation Equivariant (L-PEq)

We start with a linear permutation equivariant transformation. It can be shown [24] that any linear permutation equivariant map of one dimensional points can be written in the form, $\mathbf{x} \mapsto (\lambda\mathbf{I}+\gamma\mathbb{1}\mathbb{1}^T)\mathbf{x}$ 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

$$x_i^{(j)} \mapsto \lambda^{(j)} x_i^{(j)} + \frac{\gamma^{(j)}}{n} \sum_{k=1}^{n} x_k^{(j)}, \tag{13}$$

incorporating the mean of other points in the set. We use the mean rather than the sum as in [24] because it allows for better symmetry with our proposed generalization in Sec. E.2. It is trivial to go between the two formulations by scaling $\gamma^{(j)}$ by n. The log-determinant of the transformation (13) 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:

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

E.2 Nonlinear Weighting (NW-PEq)

We propose a generalization of the linear permutation equivariant transformation (13) 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:

$$\begin{aligned} x_i^{(j)} &\mapsto \lambda^{(j)} x_i^{(j)} + \gamma^{(j)} \frac{\sum_k x_k^{(j)} w\left(x_k^{(j)}\right)}{\sum_m w\left(x_m^{(j)}\right)} \\ &\mapsto \lambda^{(j)} x_i^{(j)} + \gamma^{(j)} \eta^{(j)} \end{aligned} \tag{14}$$

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

$$\log|J| = (n-1)\log|\lambda^{(j)}| + \log\left|\lambda^{(j)} + \gamma^{(j)}\left(1 + \frac{\sum_k \left(x_k^{(j)} - \eta^{(j)}\right) w'\left(x_k^{(j)}\right)}{\sum_m w\left(x_m^{(j)}\right)}\right)\right| \tag{15}$$

where w' is the first derivative of w. It is clear that (15) simplifies to the linear determinant for constant w. Attempting to invert (14) results in an implicit function for $\eta^{(j)}$

$$\eta^{(j)} = \left(1 + \gamma^{(j)}\right)^{-1} \frac{\sum_k x_k^{(j)} w\left(x_k^{(j)} - \gamma^{(j)}\eta^{(j)}\right)}{\sum_m w\left(x_m^{(j)} - \gamma^{(j)}\eta^{(j)}\right)} \tag{16}$$

(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 [5, 6, 11].

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, (16) simplifies to

$$\eta^{(j)} = \left(1+\gamma^{(j)}\right)^{-1} \frac{\sum_k x_k^{(j)} f\left(x_k^{(j)}\right)}{\sum_m f\left(x_m^{(j)}\right)} \tag{17}$$

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,

$$x_i^{(j)} \mapsto \lambda^{(j)} x_i^{(j)} + \gamma^{(j)} \frac{\sum_k x_k^{(j)} \exp\left(\beta^{(j)} x_k^{(j)}\right)}{\sum_m \exp\left(\beta^{(j)} x_m^{(j)}\right)} \tag{18}$$

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 (17):

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

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.

E.3 Set-NVP

Lastly, we propose a set-level scaling and shifting non-volume preserving (NVP) transformation [6]. For d dimensional points, the NVP transformation scales and shifts one subset, $S \subset \{1, \ldots, d\}$ of the d covariates given then rest, S^c as:

$$x^{(S)} \mapsto \exp\left(f\left(x^{(S^c)}\right)\right) x^{(S)} + g\left(x^{(S^c)}\right) \tag{19}$$

$$x^{(S^c)} \mapsto x^{(S^c)}, \tag{20}$$

for learned functions f, and g.

We propose to extend this mapping to sets, as follows:

$$\begin{aligned} x_i^{(S)} &\mapsto \exp\left(f\left(\varphi(\mathbf{x}^{(S^c)}), x_i^{(S^c)}\right)\right) x_i^{(S)} + g\left(\varphi(\mathbf{x}^{(S^c)}), x_i^{(S^c)}\right) \\ x_i^{(S^c)} &\mapsto x_i^{(S^c)}, \end{aligned} \tag{21}$$

where $\varphi(\mathbf{x}^{(S^c)})$ is a permutation invariant DeepSet representation. The embedding φ is responsible for capturing the set-level information from other covariates, which is then combined with each point $x_i^{(S^c)}$ for shifts and scales that are both point and set level dependent. The log-determinant is given by $f\left(\varphi(\mathbf{x}^{(S^c)}), x_i^{(S^c)}\right)$ and the inverse by

$$z_i^{(S)} \mapsto \exp\left(-f\left(\varphi(\mathbf{z}^{(S^c)}), z_i^{(S^c)}\right)\right)\left(z_i^{(S)} - g\left(\varphi(\mathbf{z}^{(S^c)}), z_i^{(S^c)}\right)\right) \tag{22}$$

$$z_i^{(S^c)} \mapsto z_i^{(S^c)}. \tag{23}$$