Learning Functions on Multiple Sets using Multi-Set Transformers

Abstract

We propose a general deep architecture for learning functions on multiple permutation-invariant sets. We also show how to generalize this architecture to sets of elements of any dimension by dimension equivariance. We demonstrate that our architecture is a universal approximator of these functions, and show superior results to existing methods on a variety of tasks including counting tasks, alignment tasks, distinguishability tasks and statistical distance measurements. This last task is quite important in Machine Learning. Although our approach is quite general, we demonstrate that it can generate approximate estimates of KL divergence and mutual information that are more accurate than previous techniques that are specifically designed to approximate those statistical distances.

1 INTRODUCTION

Typical deep learning algorithms are constrained to operate on either fixed-dimensional vectors or ordered sequences of such vectors. This is a limiting assumption which prevents the application of neural methods to many problems, particularly those involving sets. While some investigation has now been done into the problem of applying deep learning to functions on sets [Lee et al., 2018, Zaheer et al., 2017], these works all focus on the problem of learning a function on a single input set to either a corresponding set of outputs or a single fixed-dimensional output. Very little work has been done on functions of *multiple* sets.

This work seeks to fill that gap. We propose a general architecture to learn functions on multiple permutation invariant sets based on previous works by Zaheer et al. [2017] and Lee et al. [2018]. We demonstrate that this architecture is a universal approximator of *partially-permutation-invariant* functions, based on the work of Yun et al. [2020]. We demonstrate how this architecture vastly outperforms existing approaches and several baselines on a variety of tasks, as well as discuss how this can be applied to the special case of learning distance functions between two sets. This latter application allows this model to be trained as an effective estimator of quantities such as KL divergence and Mutual Information, which are highly relevant for many applications within machine learning. Furthermore, we demonstrate how to obtain an architecture that generalizes with the dimensionality of the data by making the computation equivariant with respect to the input dimensions.

2 RELATED WORK

Our method is based on the work of Zaheer et al. [2017], who originally drew attention to the problem of using neural networks to approximate functions on permutation-invariant sets. We draw particularly from the work of Lee et al. [2018], who extended this work to use transformer-based models on sets.

Gui et al. [2019] also address the idea of designing neural networks to learn functions between multiple permutation-invariant sets. Their work, however, focuses on graph embeddings as a primary application, and does not consider estimating distances or divergences. Their method also relies on a more simplistic architecture that has been criticized in Wagstaff et al. [2019a], whereas our proposed architecture has a number of theoretical and empirical advantages.

3 METHOD

3.1 BACKGROUND

Consider first the problem of learning a function upon a single set $X = \{x_1, ..., x_n\}$ in $\mathbb{R}^d$. As discussed in Zaheer et al. [2017], this general problem takes on one of two forms: the *permutation-equivariant* and *-invariant* cases. In the permutation-equivariant case, the function takes the form f :

Submitted to the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022). **To be used for reviewing only**.

$2^{\mathbb{R}^d} \rightarrow 2^{\mathbb{R}^{d'}}$, and must obey the restriction that permutations of the inputs correspond to identical permutations of the outputs - i.e. for a permutation π,

$$f(\pi(X)) = \pi(f(X))$$

In the permutation-invariant case, the function takes the form $f : 2^{\mathbb{R}^d} \rightarrow \mathbb{R}^{d'}$, and must obey the restriction that permutations of the inputs correspond to no change in the output - i.e.

$$f(\pi(X)) = f(X)$$

Zaheer et al. [2017] proposed an architecture to learn such functions that we will refer to as the *sum-decomposition* architecture (following Wagstaff et al. [2019a]). This architecture proposes to learn permutation-invariant functions using the model:

$$f(X) = \rho\left(\sum_{x \in X} \phi(x)\right) \tag{1}$$

Each member of the set x is encoded into a latent representation $\phi(x)$, which are then summed and decoded to produce an output. While this architecture is adequate for some purposes, it is also very simple, and has difficulty modeling interactions between multiple elements in the set. In particular, Wagstaff et al. [2019b] proved in their paper that *sum-decomposable* architectures such as this require each element to be mapped to a latent vector with latent size at least as large as the maximum number of elements in the input set in order to be universal approximators of functions on sets. Practically speaking, this is a major restriction when working with large sets, because this introduces a hard maximum on the size of the input set for a given latent size.

Lee et al. [2018] improved on this architecture in their paper, proposing an architecture they referred to as 'Set Transformers'. Based on their architecture, we can present a general model for attention-based models on sets as follows. Let $\text{MHA}(X, Y)$ be the standard multiheaded attention operation defined in Vaswani et al. [2017] with queries X and keys/values Y. We define a single *transformer block* by:

$$\begin{aligned} \text{ATTN}(X) &= \text{LN}(X + \text{MHA}(X, X)) \\ T(X) &= \text{LN}(\text{ATTN}(X) + \text{FF}(\text{ATTN}(X))) \end{aligned} \tag{2}$$

wherein LN is the Layer Norm operation [Ba et al., 2016], and FF is a 2-layer feedforward network with ReLU activation applied independently to each element in the set. Note that this transformer block lacks positional encodings, and is thus a permutation-equivariant operation.

In order to create a general set transformer model, we can stack multiple of these blocks followed by a pooling operation Γ and a decoder ρ to obtain:

$$f(X) = \rho\left(\prod_X T_l(T_{l-1}(...T_1(X))\right) \tag{3}$$

The Set Transformer architecture bears some similarities to another model proposed by Santoro et al. [2017]. They propose a simpler permutation-invariant architecture called "relation networks", which follow the model:

$$f(X) = \rho\left(\sum_{x_i \in X} \sum_{x_j \in X} \theta(x_i, x_j)\right) \tag{4}$$

wherein ρ is again a decoder, and θ is a feedforward *pairwise* encoder which encodes the relationship between each pair of elements in X.

In general, we can consider all three of these architectures to consist of an equivariant encoder ϕ on the set X, followed by a pooling operation Γ and decoder ρ:

$$f(X) = \rho\left(\prod_X \phi(X, X)\right) \tag{5}$$

In both the Set Transformer and Relation Network case, $\phi(X, X)$ can be written as

$$\phi(X, X) = \bigwedge_X \theta(X, X) \tag{6}$$

where $\theta(X, X)$ is a pairwise encoder that computes an encoding of each pair (x_i, x_j), then Λ is a form of pooling operation that reduces this $N \times N$ encoding matrix into a single vector for each element in X. For the base relation network architecture, θ is simply a feedforward neural net, and both pooling operations take the form of sums. For the set transformer architecture, $\phi(X, X)$ is the multiheaded self-attention operator, with $\theta(X, X)$ being the dot product of the transformed queries and keys and Λ consisting of a softmax and matrix multiplication by the transformed value matrix. The pooling operator Γ is given by the pooling-by-multiheaded-attention (PMA) operator defined in Lee et al. [2018].

3.2 MULTIPLE SETS

We propose to extend these methods to the case of multiple permutation invariant sets - which we refer to as the case of *partial permutation invariance*. We say a function $f : 2^{\mathbb{R}^d} \times 2^{\mathbb{R}^d} \rightarrow \mathbb{R}^{d'}$ is *partially permutation invariant* if it obeys the property:

$$f(\pi_1(X), \pi_2(Y)) = f(X, Y)$$

Similarly, we say a function $f : 2^{\mathbb{R}^d} \times 2^{\mathbb{R}^d} \rightarrow 2^{\mathbb{R}^d} \times 2^{\mathbb{R}^d}$ is *partially permutation equivariant* if it obeys the property:

$$f(\pi_1(X), \pi_2(Y)) = (\pi_1(f_X(X, Y)), \pi_2(f_Y(X, Y)))$$

Gui et al. [2019] propose a similar definition in their work, where they define a partially permutation invariant model:

$$f(X_1, ..., X_m) = \rho\left(\sum_{x \in X_1} \phi_1(x), ..., \sum_{x \in X_m} \phi_m(x)\right) \tag{7}$$

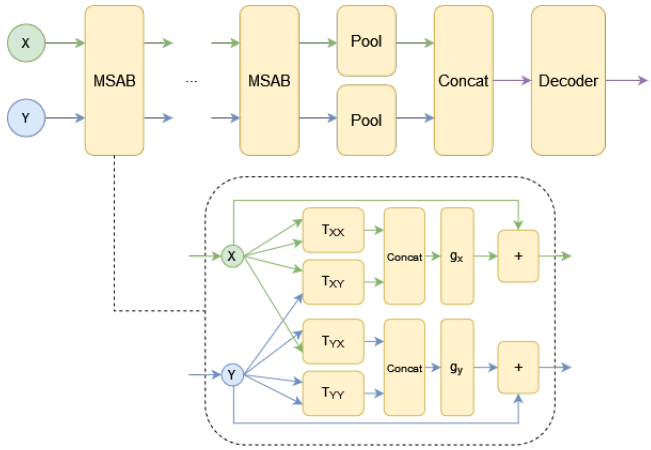


Figure 1: Diagram of the Multi-Set Transformer and Multi-Set Attention Block

As mentioned previously, this sum-decomposition architecture has a number of limitations, and instead we choose to follow the method of Lee et al. [2018]. This architecture is also advantageous for other reasons related specifically to the modelling of distance functions. Models such as transformers which explicitly model the relationship between pairs of elements in a set carry a useful inductive bias for learning distance functions. Consider the case of computing the Wasserstein distance, wherein computing the ground distance (e.g. euclidean distance) between every pair of elements of the sets is a necessary step. Similarly for quantities such as KL divergence, methods such as the algorithm of Wang et al. [2009] often rely on nearest-neighbour distances as a useful proxy for the concentration of the distribution.

3.3 OUR MODEL

We choose to instead begin from the model presented in Equation 5. In order to generalize this, let us now consider applying these architectures to the case where the single input X is now replaced by $X \bigsqcup Y$ - the concatenation of the two inputs X and Y. When the encoder acts upon this input, $\phi(X, X)$ is replaced by:

$$\phi(X \bigsqcup Y, X \bigsqcup Y) = \prod_{X \bigsqcup Y} \begin{pmatrix} \theta(X,X) & \theta(X,Y) \\ \theta(Y,X) & \theta(Y,Y) \end{pmatrix} \quad (8)$$

Instead of having a single encoder learn these four relationships, we can split θ into four separate pair encoders: θ_{XX}, θ_{XY}, θ_{YX}, and θ_{YY}. Rather than pooling over the entirety of the joint set $X \bigsqcup Y$, we pool over only the first input:

$$\phi_{xx}(X, X) = \prod_X \theta_{xx}(X, X)$$
$$\phi_{xy}(X, Y) = \prod_X \theta_{xy}(X, Y)$$
$$\phi_{yx}(Y, X) = \prod_Y \theta_{yx}(Y, X)$$
$$\phi_{yy}(Y, Y) = \prod_Y \theta_{yy}(Y, Y)$$

In this manner, each encoder learns the relationships between one of the four pairs of sets separately. This information can then be recombined to produce output encodings:

$$\phi_x(X, Y) = g_x(\phi_{xx}(X, X), \phi_{xy}(X, Y))$$
$$\phi_y(X, Y) = g_y(\phi_{yx}(Y, X), \phi_{yy}(Y, Y))$$

Equation 5 then becomes

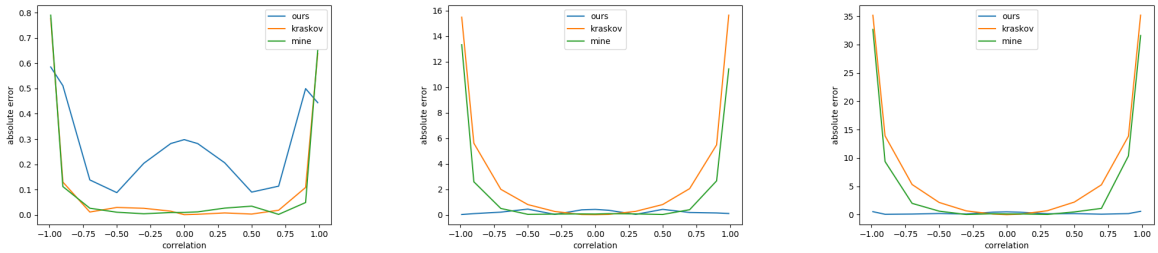
$$f(X, Y) = \rho\left(\prod_X \phi_x(X, Y), \prod_Y \phi_y(X, Y)\right) \quad (9)$$

This structure satisfies the property of partial permutation equivariance, and allows the model to retain the benefits of explicitly representing relationships between each pair of elements and each pair of sets. This general model can now be used to extend any single-set model defined by Equation 5 - including both Set Transformers and Relation Networks.

3.4 MULTI-SET TRANSFORMER

The primary architecture we consider is the *multi-set transformer* architecture, which follows from constructing the model defined in Equation 12 with transformer encoders as ϕ. We define the *multi-set attention block* $\text{MSAB}(X, Y) = (Z_X, Z_Y)$ where

$$Z_X = X + g_x(T_{xx}(X, X), T_{xy}(X, Y)) \quad (10)$$
$$Z_Y = Y + g_y(T_{yx}(Y, X), T_{yy}(Y, Y)) \quad (11)$$



(a) $d = 2$ (b) $d = 10$ (c) $d = 20$

Figure 2: Plot of absolute error in predicted mutual information for correlated Gaussian data with 2d, 10d and 20d marginals for our model and baselines.

and where $T_{ab}(A, B)$ is a transformer block as defined in Eq. 2, and the functions g are 1-layer feedforward networks with ReLU activations which are applied to the element-wise concatenation of the outputs of the two transformer blocks. These MSABs can now be treated like regular transformer blocks and stacked to form a deep encoder. Figure 1 illustrates our model with MSABs. We can then define a multi-set analogue of Eq. 3:

$$f(X, Y) = \rho\left(\prod_X \phi(X, Y)_X, \prod_Y \phi(X, Y)_Y\right) \tag{12}$$

ϕ is an encoder formed of stacked MSABs, which produces outputs of Z_X and Z_Y. These outputs are then pooled over X and Y independently, concatenated, then passed into a feedforward decoder to produce the final output. See section A for a detailed discussion of how our multi-set attention model is derived from a single-set attention block.

3.5 VARIABLE-DIMENSION ENCODERS

Another application of permutation-invariance that will be particularly useful when discussing distance functions is invariance to the input dimension itself. Applying the principles of permutation invariance to the input dimension itself, we can arrive at a formulation where the model is in fact invariant to the input dimension, and can accept inputs of any size. A traditional machine learning pipeline in - for example - NLP might learn an embedding scheme in which different dimensions encode different semantic representations of the input sequence, and should thus be treated differently from one another. In order to compute a function such as the Wasserstein distance or KL divergence on a diverse array of distributions, however, each dimension must be treated symmetrically. In the setting of statistical distances, this representation of the input dimension carries an inductive bias that is useful for the model, and generally leads to improved performance.

Zaheer et al. [2017] propose a simplistic form of this in their original paper. They demonstrate that a linear layer with a weight matrix that takes the form $\Theta = \lambda I + \gamma 1 1^T$ is equivariant with respect to the input dimension. This essentially corresponds to a neural network that computes:

$$y_i = \lambda x_i + \gamma \sum_i x_i$$

Each output is thus computed from a constant multiple of the corresponding input added to a multiple of the sum of all inputs. This has some unfortunate properties, however, since it is constrained to an output size that is exactly equal to the input size at every layer. The solution to this is to introduce multiple channels. Instead of mapping each input dimension to a single output dimension, each input dimension can be mapped to a multichannel output. Multiple encoder layers can thus be stacked, each acting only on this multichannel representation of the input dimensions, and treating the input dimension itself as a batch dimension. Then, after these encoder layers are applied, a pooling operation can be introduced over the input dimension to obtain a fixed dimensional output. This procedure allows inputs of any size to be mapped to a fixed dimension encoding.

In accordance with this, we propose an analogous *multi-channel transformer block*, wherein the weight matrices are applied similarly as multichannel transformations which treat the input dimension as a batch dimension. A standard multiheaded attention block receives inputs $X, Y \in \mathbb{R}^{n \times d}$ and computes:

$$\text{MHA}(X, Y) = \sigma\left((XW_Q)(YW_K)^T\right) YW_V W_O \tag{13}$$

Our multichannel attention block instead is a function from $\mathbb{R}^{n \times d \times n_c} \times \mathbb{R}^{m \times d \times n_c}$ to $\mathbb{R}^{n \times d \times n_c}$ which computes:

$$A = \sigma\left(\sum_{i=1}^{d} (X_{:,i} W_Q)(Y_{:,i} W_K)^T\right)$$
$$\text{MHA}(X, Y)_{:,i} = AY_{:,i} W_V W_O$$

Our multichannel transformer architecture consists of an initial projection from a single input channel up to k input

channels, followed by n_b multichannel transformer blocks, followed by a max pooling over the input dimension d. This is also compatible with the Multi-Set Attention Block architecture, in which case the multichannel transformer blocks are replaced by multichannel MSABs, with a max pooling at the end as before.

4 THEORETICAL ANALYSIS

We demonstrate that our proposed multi-set transformer architecture is a universal approximator on *partially permutation equivariant* functions, and that combined with a pooling layer it is also a universal approximator of *partially permutation invariant* functions.

First, some preliminaries. We will follow the notation in Yun et al. [2020], for their Theorem 2 forms a foundation for the theorem we will state shortly. Let $\mathcal{F}_{PE}$ be the class of all continuous permutation-equivariant functions with compact support from $\mathbb{R}^{d\times n}$ to $\mathbb{R}^{d\times n}$. Given $f, g : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ and $1 \leq p \leq \infty$, let

$$d_p(f, g) = \left(\int \|f(\mathbf{X}) - g(\mathbf{X})\|_p^p d\mathbf{X} \right)^{1/p} \tag{14}$$

Let $t^{h,m,r} : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ denote a transformer block with an attention layer with h heads of size m and a feedforward layer with r hidden nodes. Then, let $\mathcal{T}^{h,m,r}$ define the class of functions $g : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ such that g consists of a composition of transformer blocks of the form $t^{h,m,r}$. We can now restate Theorem 2 from Yun et al. [2020], which is given by:

Theorem 4.1. *Let $1 \leq p \leq \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{PE}$ there exists a transformer network $g \in \mathcal{T}^{2,1,4}$ such that $d_p(f, g) \leq \epsilon$. [Yun et al., 2020]*

To extend this to the case of partial permutation equivariance, we must now modify these definitions slightly. Let $\mathcal{F}_{PPE}$ be the class of all continuous partially-permutation-equivariant functions on two sets with compact support from $\mathbb{R}^{d\times n} \times \mathbb{R}^{d\times m}$ to $\mathbb{R}^{d\times n} \times \mathbb{R}^{d\times m}$. Let $c^{h,m,r} : \mathbb{R}^{d\times n} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{d\times n} \times \mathbb{R}^{d\times m}$ denote a multi-set attention block with attention layers with h heads of size m and feedforward layers with r hidden nodes, and let $\mathcal{T}_C^{h,m,r}$ define the class of functions $g : \mathbb{R}^{d\times n} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{d\times n} \times \mathbb{R}^{d\times m}$ such that g consists of a composition of MSABs of the form $c^{h,m,r}$. Our theorem now states:

Theorem 4.2. *Let $1 \leq p \leq \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{PPE}$ there exists a multi-set transformer network $g \in \mathcal{T}_C^{2,2,4}$ such that $d_p(f, g) \leq \epsilon$.*

A proof of Theorem 4.2 is given in the appendices. If we define $\mathcal{F}_{PPI}$ to be the class of all continuous partially-permutation-invariant functions on two sets with compact support from $\mathbb{R}^{d\times n} \times \mathbb{R}^{d\times m}$ to $\mathbb{R}^d$, this now directly leads to the corollary:

Corollary 4.3. *Let $1 \leq p \leq \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{PPI}$ there exists a function $f(X, Y) = \max g(X, Y)$ such that $d_p(f, g) \leq \epsilon$, wherein $g \in \mathcal{T}_C^{2,2,4}$ is a multi-set transformer network.*

Observe that this corollary follows directly from Theorem 4.2, since for any function $f \in \mathcal{F}_{PPI}$ we can construct $g \in \mathcal{F}_{PPE}$ such that $g(X, Y)_{:,j} = f(X, Y)\ \forall j$ (i.e., a set of outputs where each entry simply contains $f(X, Y)$). f thus obeys the equation $f(X, Y) = \max_j g(X, Y)_{:,j}$. Therefore, any function in $f \in \mathcal{F}_{PPI}$ can be expressed as a pooling function applied to a function $g \in \mathcal{F}_{PPE}$ and f can thus be approximated by a multi-set transformer network by simply approximating g as per Theorem 4.2.

5 EXPERIMENTS

In order to evaluate the model, we consider several tasks, including a number of simple image-based set tasks as well as the aforementioned distance functions. We compare our model against the PINE model proposed in [Gui et al., 2019], as well as a number of single-set models. For those baselines, we take a single-set architecture such as Deep Sets (Single-Set RFF), Relation Networks (Single-Set RN) or Set Transformers (Single-Set Transformer), compute pooled representations for each of X and Y, then concatenate these representations and pass them into a feedforward decoder. We also consider several variants and ablations of our model. The two variants of our model include Multi-Set Transformer and Multi-Set RN (Relation Networks). In the latter, transformer blocks are replaced by relation network blocks for the four encoders, with max pooling operations for both Λ and Γ. Then, we also consider several ablations of our model. First, we consider a variant where $g_x(T_{xx}(X, X), T_{xy}(X, Y)) = T_{xx}(X, X) + T_{xy}(X, Y)$ (referred to as Sum-Merge). Second, we consider modifications to the four-block encoder structure itself by removing T_{XX} and T_{YY} - leaving only the cross terms T_{XY} and T_{YX} (referred as Cross-Only). Finally, the single set transformer baseline (Single-Set Transformer) can itself be considered an ablation of our model with the cross-set blocks removed instead of the same-set blocks. For all experiments we perform three trials and report the average and standard deviation.

Hyperparameter settings for all experiments and other details can be found in section B of the appendix.

5.1 STATISTICAL DISTANCES

One particular application of partially-permutation-invariant models that is worth highlighting is their ability to learn to

	d=2	d=4	d=8	d=16
Baselines				
KNN	0.2047	0.5662	4.0584	28.0382
PINE	0.1737 ± 0.0003	0.4958 ± 0.0011	2.0804 ± 0.0004	10.534 ± 0.0109
Single-Set RFF	0.1219 ± 0.0114	0.4400 ± 0.0159	1.7770 ± 0.0119	8.0078 ± 0.1636
Single-Set RN	0.1555 ± 0.0007	0.5264 ± 0.0019	2.1425 ± 0.0032	9.5963 ± 1.6214
Single-Set ST	0.0732 ± 0.0032	0.2601 ± 0.0043	1.6885 ± 0.0337	7.5911 ± 0.2125
Our Model				
Multi-Set Transformer	0.0731 ± 0.0011	**0.1903** ± 0.0082	0.9212 ± 0.0186	11.105 ± 0.0717
Multi-Set RN	0.1061 ± 0.0020	0.3926 ± 0.0144	1.5170 ± 0.0430	7.4002 ± 0.1263
Cross-Only	0.0792 ± 0.0019	0.1968 ± 0.0014	0.9926 ± 0.0351	5.4214 ± 0.1694
Sum-Merge	**0.0699** ± 0.0008	0.1953 ± 0.0047	0.9320 ± 0.0138	10.5080 ± 0.0662
Multi-Set-Transformer-Equi	0.0726 ± 0.0017	0.1917 ± 0.0020	**0.8002** ± 0.0222	**4.7000** ± 0.2966

Table 1: Mean absolute error of models trained on Gaussian mixture data for estimating KL divergence.

approximate statistical distances between distributions such as the KL divergence or mutual information. Both Mutual Information and the KL divergence are useful metrics that are widely used in a variety of settings within machine learning, and both are very difficult to estimate for any but the simplest distributions. We consider both our proposed model, as well as the dimension-equivariant model discussed in Sect. 3.5.

5.1.1 KL Divergence

Training the estimator to learn the KL divergence has unique challenges, as calculating the ground truth requires the log likelihood for both the source and target distributions. In order to train our model to learn the KL divergence between a general class of distributions, we need to find a class of models that are effective universal approximators and also admit a tractable log likelihood. The most obvious class of models fitting this criteria is that of Gaussian mixture models. We generate Gaussian mixtures with a uniformly random number of components (between 1 and 10) and mixture weights sampled from a uniform Dirichlet distribution. The means of each Gaussian are generated from a uniform distribution, and the covariance matrices are generated by multiplying a correlation matrix sampled from a Lewandowski-Kurowicka-Joe (LKJ) distribution (with $\nu = 5$) by a vector of covariances distributed according to a log-normal distribution (with $\mu_0 = 0$, $\sigma_0 = 0.3$). Each training example consists of a random number of points $X \sim p_X$ and a random number of points $Y \sim p_Y$ (with $N_X, N_Y \in [100, 150]$). The ground truth is estimated by a Monte Carlo estimate of the true KL divergence using the closed-form log likelihoods, with the generated points X as the samples. We normalize the generated data by computing the mean and covariance across both X and Y, then applying a whitening transformation

$$[X'; Y'] = \Sigma_{XY}^{-1/2} \left([X; Y] - \mu_{XY}\right) \tag{15}$$

under which the KL divergence is invariant.

We compare our model (with and without dimension-equivariance) against the aforementioned baselines, as well as the k-nearest-neighbours estimator of Wang et al. [2009]. Table 1 shows the mean average error of each model on Gaussian mixture data, averaged over 3 runs. Our model has the lowest error on all dimensions considered. The dimension-equivariant model performed approximately equal to the standard transformer model in low dimension, but performed significantly better in high dimension.

Convergence was a significant issue with the GMM data in higher dimensions, since as the dimensionality increased the true KL divergence of the generated distributions would often explode. This effect was especially notable when the concentration parameter of the LKJ distribution was small, but always occurs once the dimensionality gets large enough.

5.1.2 Mutual Information

We also show the effectiveness of our method for estimating mutual information. Following previous work [Belghazi et al., 2018, Kraskov et al., 2004], we use Gaussians with componentwise correlations $\rho \in (-1, 1)$, with standardized Gaussian marginals. Training examples are generated in a similar fashion as in the KL case, with a ρ sampled uniformly from the interval $(-1, 1)$, then a random number of samples between 100 and 150 drawn from the resulting distribution for each of X and Y. We plot the performance of our model for varying values of ρ compared to both Kraskov et al. [2004] and MINE [Belghazi et al., 2018] in 2, 10 and 20 dimensions (see Fig. 2). Our model performs somewhat worse than the other methods shown in the 2-dimensional case, but is almost indistinguishable from the ground truth in the 10 and 20-dimensional cases. Note also that while methods such as MINE must be trained on a particular dataset in order to predict its mutual information, our method need

Model	MNIST	Omniglot
Baselines		
PINE	0.4607 ± 0.0074	0.6627 ± 0.0152
Single-Set RFF	0.4233 ± 0.0171	0.6271 ± 0.0034
Single-Set RN	0.5288 ± 0.0718	0.6712 ± 0.0030
Single-Set Transformer	0.9200 ± 0.0276	0.7154 ± 0.0071
Our Models		
Multi-Set Transformer	0.9742 ± 0.0018	0.8374 ± 0.0159
Multi-Set RN	0.9775 ± 0.0080	$\mathbf{0.8523} \pm 0.0045$
Cross-Only	0.9743 ± 0.0007	0.8031 ± 0.0412
Sum-Merge	$\mathbf{0.9776} \pm 0.0008$	0.8366 ± 0.0107

Table 2: Accuracy and standard deviation of each model across the MNIST and Omniglot counting tasks.

Model	CoCo	FastText
Baselines		
PINE	0.5017 ± 0.0022	0.4977 ± 0.0024
Single-Set RFF	0.4992 ± 0.0051	0.4974 ± 0.0028
Single-Set RN	0.5019 ± 0.0039	0.7858 ± 0.0134
Single-Set Transformer	0.8928 ± 0.0022	0.7698 ± 0.0114
Our Model		
Multi-Set Transformer	0.9171 ± 0.0060	$\mathbf{0.8221} \pm 0.0018$
Multi-Set RN	$\mathbf{0.9222} \pm 0.0019$	0.7625 ± 0.0090
Cross-Only	0.9143 ± 0.0024	0.8097 ± 0.0070
Sum-Merge	0.9016 ± 0.0048	0.8160 ± 0.0092

Table 3: Accuracy and standard deviation of each model on the alignment task.

only be trained once, and can then be used for inference on any similar dataset wthout retraining.

5.2 IMAGE TASKS

We begin by looking at a selection of tasks similar to those considered by Lee et al. [2018] and Zaheer et al. [2017]. We study the model's ability to perform a number of simple set-based operations between sets of images. When working with image or text data, each example was first individually encoded as a fixed-size vector using an appropriate image or text encoder, then passed through the set based model.

5.2.1 Counting Unique Images

For the first task, the models were given input sets consisting of images of characters. The task was to identify the number of unique characters that were shared between the two input sets of a variable number of images (6-10 for Omniglot, 10-30 for MNIST). For this task, we used simple CNN encoders that were pretrained on the input datasets as classifiers for a short number of steps, then trained end to end with the set-based model. We evaluated the models on the MNIST and Omniglot datasets, and found that our models convincing outperformed the alternatives - achieving almost 98% accuracy on the MNIST task and 83-85% accuracy on the Omniglot task (see Table 2). In each case, our models outperformed the baselines by considerable margins. The RN-based model outperformed the transformer model by a margin of about 1.5% on the Omniglot task, and they performed equivalently on the MNIST task. The effects of the ablations were variable, but in both cases they were relatively close to the performance of the base model.

5.2.2 Alignment

While the first task was purely synthetic, this second task is representative of a general class of applications for this model - predicting alignment between two sets. The first example of this we chose was image captioning on the MSCOCO dataset. The models were given a set of 8-15 images and a set of captions of the same size, and tasked to predict the probability that the two sets were *aligned* - i.e. that the given set of captions consisted of captions for the given set of images. For this task, we used the pretrained models BERT and ResNet-101 as encoders for the text and images respectively. For the second example, we chose to use cross-lingual embeddings. Conneau et al. [2017] show that there is a geometric relationship between learned FastText embeddings across languages. As such, the model should be able to predict the alignment between sets of embeddings in one language and sets of embeddings in another. We show the model a set of 10-30 embeddings in English and another set of the same size of embeddings in French. The model is tasked to predict whether or not the embeddings in the two sets correspond to the same words.

Results on these tasks are shown in Table 3. Our model performed the highest across both tasks. Interestingly, the single-set transformer model performed quite well on the CoCo task (though not as well as our model), but was significantly worse on the FastText task. No other baseline aside from the single-set RN performed notably better than chance. Given that the unaligned sets consisted of entirely disjoint images and captions (i.e., no images and captions overlapped), it's possible that learning whether the net sum of all embedded vectors in each set were aligned might be sufficient, and the model might not need to directly compare individual elements across sets. This might explain the high performance of the single-set transformer model - though interestingly, this did not hold true for the FastText task (perhaps due to the fact that the FastText task appeared to be more difficult).

Model	Synthetic	Meta-Dataset
Baselines		
PINE	0.5012 ± 0.0020	0.5048 ± 0.0028
Single-Set RFF	0.5005 ± 0.0018	0.7831 ± 0.0069
Single-Set RN	0.4997 ± 0.0013	0.7981 ± 0.0642
Single-Set Transformer	0.6039 ± 0.0178	0.8811 ± 0.0092
Our Models		
Multi-Set Transformer	0.7289 ± 0.0353	0.8922 ± 0.0142
Multi-Set RN	$\mathbf{0.7350} \pm 0.0094$	0.8679 ± 0.0111
Cross-Only	0.6353 ± 0.0191	$\mathbf{0.9043} \pm 0.0093$
Sum-Merge	0.6292 ± 0.0101	0.8683 ± 0.0073

Table 4: Accuracy and standard deviation of each model on the distinguishability task.

5.2.3 Distinguishability

The last task in this category was distinguishability. Given two sets of samples, the models would be asked to predict whether the two input sets were drawn from the same underlying distribution. We again considered two examples for this task: a synthetic dataset, and a dataset of real-world images. For the synthetic data, we sampled sets from randomly generated 8-dimensional Gaussian mixtures (Gaussian mixture parameters were generated in the same fashion as the data in Section 5.1.1). For the second example, we used Meta-Dataset [Triantafillou et al., 2019], a dataset consisting of 12 other image datasets, each with many subclasses. In each case, each training example consisted of a batch of two sets of 10-30 data points. The data points would be drawn from the same distribution (the same Gaussian mixture for the synthetic data, or the same class from the same parent dataset for Meta-Dataset) with probability $1/2$, and generated from different distributions with probability $1/2$. The model was tasked to predict the probability of the sets being drawn from the same distribution. For the meta-dataset task, images were first encoded using a CNN trained along with the network.

Results are shown in Table 4. The multi-set transformer and RN models performed by far the best on the synthetic task. On the image task, the single-set transformer again performed very well, though not as well as the multi-set transformer model. We hypothesize this might be because the distinguishability task relies on recognizing the distribution from which the set is drawn, which is a task that might be possible to do by simply reducing each input set to a single vector and then comparing the resulting vectors.

5.3 ANALYSIS

Overall, we considered two different variants of our model (RN-based and transformer-based), as well as several ablations of the transformer model. The base multi-set transformer model performed consistently well across every task, with either the highest accuracy or close to the highest accuracy. The relation-network variant of the model performed slightly better on a number of tasks, but significantly worse on many others. This variant of the model also had significant issues with memory usage, and often required very small batch sizes in order to fit on GPUs. Each of the single-set transformer, cross-only, and sum-merge models can be considered ablations of the multi-set transformer architecture. The cross-only model performed competitively or slightly better on some tasks, but similar to the RN model, it performed worse by notable margins on others. It's possible that the T_{XX} and T_{YY} blocks - which computed the internal relationships between elements in X and Y - were simply not needed for certain tasks, but very helpful for others. The sum-merge model generally performed comparably to the base model, and degraded performance by notable margins only in the case of the distinguishability tasks. This is not entirely unexpected, given that it is the most minor of the ablations, and represents only a small change to the structure of the model.

6 DISCUSSION AND CONCLUSION

The Multi-Set Transformer model we define performs well at estimating a variety of distance/divergence measures between sets of samples, even for quantities that are notoriously difficult to estimate. It clearly outperforms existing multi-set and single-set architectures, and beats existing knn-based estimators in the settings we analyzed. In an ideal case, the model could be pretrained once and then applied as an estimator for, e.g., KL divergence in a diverse array of settings. This is one of our primary areas of focus going forward.

Since this model is a universal approximator for partially permutation equivariant functions, its applications are far broader than simply that of training estimators for divergences between distributions. We showcase a number of simple applications with image data, but these are merely meant to be representative of larger classes of applications. The applications in terms of distinguishability, for example, are highly reminiscent of GANs [Goodfellow et al., 2014], and the FastText task from the alignment section bears some similarities to existing work in which GANs are used [Conneau et al., 2017], where our model might be able to be lead to improvements. Our model could also be used to train bespoke distance functions that could be trained end to end as part of a particular task. We hope to show these more diverse applications in greater detail in future work, as well as exploring better ways to train estimators that will generalize broadly, and looking at other quantities of interest such as Wasserstein Distance or f-divergences.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Ishmael Belghazi, Sai Rajeswar, Aristide Baratin, R. Devon Hjelm, and Aaron C. Courville. MINE: mutual information neural estimation. *CoRR*, abs/1801.04062, 2018. URL `http://arxiv.org/abs/1801.04062`.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Shupeng Gui, Xiangliang Zhang, Pan Zhong, Shuang Qiu, Mingrui Wu, Jieping Ye, Zhengdao Wang, and Ji Liu. PINE: universal deep embedding for graph nodes via partial permutation invariant set functions. *CoRR*, abs/1909.12903, 2019. URL `http://arxiv.org/abs/1909.12903`.

Alexander Kraskov, Harald Stoegbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E*, 69(6):066138, Jun 2004. ISSN 1539-3755, 1550-2376. doi: 10.1103/PhysRevE.69.066138. arXiv: cond-mat/0305641.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer. *CoRR*, abs/1810.00825, 2018. URL `http://arxiv.org/abs/1810.00825`.

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter W. Battaglia, and Timothy P. Lillicrap. A simple neural network module for relational reasoning. *CoRR*, abs/1706.01427, 2017. URL `http://arxiv.org/abs/1706.01427`.

Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *CoRR*, abs/1903.03096, 2019. URL `http://arxiv.org/abs/1903.03096`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A. Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, page 6487–6494. PMLR, May 2019a. URL `http://proceedings.mlr.press/v97/wagstaff19a.html`.

Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A. Osborne. On the limitations of representing functions on sets. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6487–6494. PMLR, 09–15 Jun 2019b. URL `https://proceedings.mlr.press/v97/wagstaff19a.html`.

Qing Wang, Sanjeev R. Kulkarni, and Sergio Verdu. Divergence estimation for multidimensional densities via k-nearest-neighbor distances. *IEEE Transactions on Information Theory*, 55(5):2392–2405, 2009. doi: 10.1109/TIT.2009.2016060.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions?, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. *CoRR*, abs/1703.06114, 2017. URL `http://arxiv.org/abs/1703.06114`.

A ATTENTION DERIVATION

A typical self-attention block with the set $X \in \mathbb{R}^{n \times d}$ as the input queries and keys/values obeys the following equation:

$$\begin{aligned} Z &= \text{MHA}(X, X) \\ &= \left[\sigma\left((XW_Q)(XW_K)^T\right)(XW_V)\right] W_O \\ &= \sigma\left(X(W_Q W_K^T)X^T\right) XW_V W_O \end{aligned}$$

If we now consider the joint set $X \bigsqcup Y \in \mathbb{R}^{n+m \times d}$ and perform self-attention on that, we find the following:

$$\begin{aligned} \begin{pmatrix} Z_X \\ Z_Y \end{pmatrix} &= \text{ATTN}\left(\begin{pmatrix} X \\ Y \end{pmatrix}, \begin{pmatrix} X \\ Y \end{pmatrix}\right) \\ &= \left[\sigma\left(\begin{pmatrix} XW_Q \\ YW_Q \end{pmatrix}\begin{pmatrix} XW_K \\ YW_K \end{pmatrix}^T\right)\begin{pmatrix} X \\ Y \end{pmatrix} W_V\right] W_O \\ &= \begin{pmatrix} A_{xx} & A_{xy} \\ A_{yx} & A_{yy} \end{pmatrix}\begin{pmatrix} X \\ Y \end{pmatrix} W_V W_O \end{aligned}$$

wherein

$$\begin{pmatrix} A_{xx} & A_{xy} \\ A_{yx} & A_{yy} \end{pmatrix} = \sigma\left(\begin{pmatrix} X(W_Q W_K^T)X^T & X(W_Q W_K^T)Y^T \\ Y(W_Q W_K^T)X^T & Y(W_Q W_K^T)Y^T \end{pmatrix}\right)$$

Then, we find that

$$\begin{aligned} Z_X &= A_{xx} XW_V W_O + A_{xy} YW_V W_O \\ Z_Y &= A_{yx} XW_V W_O + A_{yy} YW_V W_O \end{aligned}$$

This function remains entirely equivariant with respect to the order of the elements in the joint set $X \bigsqcup Y$ - there is no distinction between elements in X and elements in Y.

Suppose, however, that we were to let the softmax for $A_{\alpha\beta}$ be computed only over the elements of the set α, and let the parameter matrices be different for each of the 4 terms. Then, we find

$$\begin{aligned} Z_X &= \sigma(X(W_{Q,xx} W_{K,xx}^T)X^T) XW_{V,xx} W_{O,xx} \\ &\quad + \sigma(X(W_{Q,xy} W_{K,xy}^T)Y^T) YW_{V,xy} W_{O,xy} \\ Z_Y &= \sigma(Y(W_{Q,yx} W_{K,yx}^T)X^T) XW_{V,yx} W_{O,yx} \\ &\quad + \sigma(Y(W_{Q,yx} W_{K,yy}^T)Y^T) YW_{V,yy} W_{O,yy} \end{aligned}$$

These are just four separate attention blocks! Now we can simply write

$$\begin{aligned} Z_X &= \text{MHA}_{xx}(X, X) + \text{MHA}_{xy}(X, Y) \\ Z_Y &= \text{MHA}_{yx}(Y, X) + \text{MHA}_{yy}(Y, Y) \end{aligned}$$

Since the attention function is equivariant with respect to the first input and invariant with respect to the second, this meets the conditions for partial equivariance. To make this slightly more general, we can now write

$$\begin{aligned} Z_X &= g_x\left(\text{MHA}_{xx}(X, X), \text{MHA}_{xy}(X, Y)\right) \\ Z_Y &= g_y\left(\text{MHA}_{yx}(Y, X), \text{MHA}_{yy}(Y, Y)\right) \end{aligned}$$

where the function g acts on each vector in the output set independently.

B EXPERIMENT DETAILS

The base architecture used in all experiments was the architecture shown in Figure 1, with the MSAB blocks replaced as appropriate for each baseline. The only exception was the PINE model, which followed the architecture described in their paper.

In all cases (except where noted otherwise), we used architectures with 4 blocks, 4 attention heads (for the transformer models), and 1-layer feedforward decoders. We used Pooling by Multiheaded Attention (PMA) (see Lee et al. [2018]) as the pooling layer for the overall network, and max pooling within each relation network block. We used layer norm around each encoder block, as well as within the transformer blocks as per usual. Each block used the same latent and hidden size, and linear projection layers were added at the beginning of the network to project the inputs to the correct dimension if needed.

For the KL and MI experiments, we trained for 100,000 batches of size 64 with a learning rate of 1e-4. The models used a latent size of 16 per input dimension and feedforward size of 32 per input dimension. The dimension-equivariant model was trained across data of multiple dimensions (1-3 for $d = 2$, 3-5 for $d = 4$, 7-9 for $d = 8$ and 14-18 for $d = 16$). Sets were generated as described in sections 5.1.1 and 5.1.2.

For the Counting experiments, we trained with a batch size of 64 using a latent size of 128 and hidden size of 256. We used a single projection layer as a decoder, with no hidden layers. For MNIST, we used a convolutional encoder with 3x3 convolutional layers of 32 and 64 filters, each followed by a max pool, with a linear projection to the latent size of 128 at the end. This encoder was pretrained for 1000 batches, then the network and encoder were trained end to end for 10,000 batches with a learning rate of 3e-4. Sets were randomly sampled with set size randomly selected in $[10, 30]$. For Omniglot, the convolutional encoder used one 7x7 conv with stride 2 and 32 filters, followed by three blocks of two 3x3 convs each with 32, 64 and 128 filters respectively. Each block was followed by a max pool, and a final linear projection to size 128 was again added at the end. This was pretrained for 300 batches, then the network itself was trained end to end for 10,000 batches with a learning rate of 1e-4. Loss was calculated by mean-squared error. Sets were randomly sampled with set size randomly selected in $[6, 10]$.

The CoCo experiments again used convolutional encoders to obtain fixed size representations of each image, and used transformer encoders to do the same for the captions. This time, the pretrained ResNet-101 model was used as the image encoder, with BERT used as the text encoder. The model was trained for 2500 batches of batch size 48 with learning rate 1e-5, using a latent size of 512 and hidden size of 1024. The set size was increased gradually according to a schedule during training, beginning at sets with size randomly selected in $[1, 5]$ for 1250 batches, $[3, 10]$ for 625 batches and $[8, 15]$ for 625 batches. Standard image preprocessing techniques were applied, with each image rescaled to 256x256, center cropped to size 224, then normalized according to the method expected by PyTorch's pretrained ResNet models (see https://pytorch.org/vision/stable/models.html). The FastText experiments used common crawl FastText vectors for English and French [1], with ground truth translations taken from MUSE [2]. The model was trained for 3125 batches of batch size 128 with learning rate 1e-5, using a latent size of 512 and hidden size of 1024. The set size was increased gradually according to a schedule during training, beginning at sets with size randomly selected in $[1, 5]$ for 1250 batches, $[3, 10]$ for 625 batches, $[8, 15]$ for 625 batches and $[10, 30]$ for 625 batches.

Meta-Dataset experiments used the same convolutional encoder architecture as the Omniglot experiments, though without pretraining. Images were preprocessed in the standard fashion performed by the Pytorch Meta-Dataset library. The model was trained for 7500 batches of batch size 64 with set size randomly selected in $[10, 30]$ and learning rate 1e-5, using a latent size of 512 and hidden size of 1024. The synthetic experiments for distinguishability were trained for 7500 batches of batch size 256 with set size randomly selected in $[10, 30]$ and learning rate 1e-5, using a latent size of 8 and hidden size of 16.

C PROOF OF THEOREM 4.2

Our proof will closely follow the work of Yun et al. [2020]. In their work, they define the following theorem:

Theorem 2 [Yun et al., 2020]. *Let $1 \leq p \leq \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{PE}$ there exists a transformer network $g \in \mathcal{T}^{2,1,4}$ such that $d_p(f, g) \leq \epsilon$.*

The proof of this theorem follows several stages, enumerated here:

1. Approximate $\mathcal{F}_{\text{PE}}$ by piecewise constant functions on the grid of resolution δ, denoted $\mathbb{G}_\delta = 0, \delta, ..., 1 - \delta^{d \times n}$.
2. Approximate $\overline{\mathcal{F}}_{\text{PE}}$ by modified transformers, with hardmax replacing softmax.

Proposition 4 [Yun et al., 2020]. *For each $\overline{f} \in \overline{\mathcal{F}}_{PE}$ and $1 \leq p < \infty$, $\exists \overline{g} \in \overline{\mathcal{T}}^{2,1,1}$ such that $d_p(\overline{f}, \overline{g}) = O(\delta^{d/p})$*

[1] https://fasttext.cc/docs/en/crawl-vectors.html

[2] https://github.com/facebookresearch/MUSE#ground-truth-bilingual-dictionaries

3. Approximate the class of modified transformers $\overline{\mathcal{T}}^{2,1,1}$ with $\mathcal{T}^{2,1,4}$

Each of these steps leads to a certain error under d_p, with step 1 and 3 contributing an error of order $\epsilon/3$ and step 2 contributing an error of order $\mathcal{O}(\delta^{d/p})$. This leads to a total error of less than order ϵ, so long as δ is chosen to be sufficiently small. Of these steps, we will focus our attention on step 2, as the others remain unchanged. The key is to prove our own version of proposition 4 [Yun et al., 2020]. The proof of this proposition itself follows the following steps:

1. Given $X \in \mathbb{R}^{d\times n}$, quantize X to $L^{(x)} \in G_\delta^+$, where G_δ is the $[0,1]^{d\times n}$ grid with resolution δ and $G_\delta^+ = G_\delta \cup \{-\delta^{-nd}\}$.
2. Implement a *contextual mapping* $q(X)$ such that all elements of $q(L), q(L')$ are distinct if L, L' are not permutations of each other. This essentially maps each (x_i, X) to a unique representation.
3. Since each (x_i, X) is mapped to a unique representation, we can use feedforward networks to approximate any desired decoder to approximate any equivariant function on X.

The critical part of this proof comes in Lemma 6 from Yun et al:

Lemma 6 [Yun et al., 2020]. *Let $\widetilde{G}_\delta = \{L \in G_\delta |\ L_{:,i} \neq L_{:,j}\ \forall i \neq j\}$. Let $n \geq 2$ and $\delta^{-1} \geq 2$. Then, there exists a function $q(L)$ of the form $q(L) = u^T g_c(L)$ where $u \in \mathbb{R}^d$, and $g_c(L) : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ is a function composed of $\delta^{-d} + 1$ self-attention layers using the σ_H operator, such that $q(L)$ has the following properties:*

1. *For any $L \in \widetilde{G}_\delta$, all entries of $q(L)$ are distinct*
2. *For any $L, L' \in \widetilde{G}_\delta$, if L is not a permutation of L' then all entries of $q(L)$ and $q(L')$ are distinct*
3. *For any $L \in \widetilde{G}_\delta$, all entries of q(L) are in $[t_l, t_r]$*
4. *For any $L \in G_\delta^+ \setminus \widetilde{G}_\delta$, all entries of q(L) are not in $[t_l, t_r]$*

For our purposes, we will keep the structure of this proof, and change it by substituting our own version of this lemma:

Lemma 6'. *Let $\widetilde{G}_\delta = \{L \in G_\delta |\ L_{:,i} \neq L_{:,j}\ \forall i \neq j\}$. Let $n \geq 2$ and $\delta^{-1} \geq 2$. Then, there exists a function $q(L^X, L^Y)$ on $L^X, L^Y \in G_\delta^+; L^X \neq L^Y$ consisting of $2\delta^{-d} + 4$ MSAB layers and constants $t_l^X, t_r^X, t_l^Y, t_r^Y$ such that*

1. *For any $L^X, L^Y \in G_\delta$, all entries of $q(L^X, L^Y)$ are distinct*
2. *For any $L^X, L^Y, L'^X, L'^Y \in G_\delta$, if L'^X is not a permutation of L^X and L'^Y is not a permutation of L^Y then all entries of $q(L^X, L^Y)$ and $q(L'^X, L'^Y)$ are distinct*
3. *For any $L^X, L^Y \in \widetilde{G}_\delta$, all entries of $q^X(L^X, L^Y)$ are in $[t_l^X, t_r^X]$*
4. *For any $L^X, L^Y \in \widetilde{G}_\delta$, all entries of $q^Y(L^X, L^Y)$ are in $[t_l^Y, t_r^Y]$*
5. *For any $L^X \in G_\delta^+ \setminus \widetilde{G}_\delta$, $L^Y \in G_\delta^+$, all entries of $q(L^X, L^Y)$ are not in $[t_l, t_r]$*
6. *For any $L^Y \in G_\delta^+ \setminus \widetilde{G}_\delta$, $L^X \in G_\delta^+$, all entries of $q(L^X, L^Y)$ are not in $[t_l, t_r]$*

Once our version of Lemma 6 is established, we then make a slight modification of Yun et al's Lemma 7. This lemma states:

Lemma 7 [Yun et al., 2020]. *Let $g_c : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ be the function defined in Lemma 6 [Yun et al., 2020]. Then, there exists a function $g_v : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ composed of $O(n(\frac{1}{\delta})^{dn}/n!)$ column-wise feedforward layers ($r = 1$) such that g_v is defined by a function $g_{col} : \mathbb{R}^d \to \mathbb{R}^d$,*

$$g_v(Z) = [g_{col}(Z_{:,1}), ..., g_{col}(Z_{:,n})]$$

where $\forall j = 1, ..., n$

$$g_{col}(g_c(L)_{:,j}) = \begin{cases} (A_L)_{:,j} & L \in \widetilde{G_\delta} \\ 0_d & L \in G_\delta^+ \setminus \widetilde{G_\delta} \end{cases}$$

Our modified version states

Lemma 7'. *Let $g_c : \mathbb{R}^{d\times n} \times \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n} \times \mathbb{R}^{d\times n}$ be the function defined in Lemma 6'. Then, there exists a function $g_v : \mathbb{R}^{d\times n} \times \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n} \times \mathbb{R}^{d\times n}$ composed of $O(n(\frac{1}{\delta})^{dn}/n!)$ column-wise feedforward layers ($r = 1$) such that g_v is defined by a function $g_{col} : \mathbb{R}^d \to \mathbb{R}^d$,*

$$g_v(Z) = [g_{col}(Z_{:,1}), ..., g_{col}(Z_{:,n})]$$

where $\forall i = 1, ..., n, j = 1, ...m$

$$g_{col}(g_c(L)_{:,j}) = \begin{cases} (A_L)_{:,j} & L \in \widetilde{G_\delta} \\ 0_d & L \in G_\delta^+ \setminus \widetilde{G_\delta} \end{cases}$$

The construction of $q(L^X, L^Y)$, and the proofs of Lemma 6' and 7' will follow in the subsequent sections.

C.1 CONSTRUCTION OF $q(L^X, L^Y)$

The construction of this function proceeds as follows. First, note that a multi-set attention block can implement functions consisting of multiple feedforward or attention blocks successively by using the skip connections to ignore the other component when needed. It can also implement a function consisting of multiple blocks successively applied to a single one of the input sets by simply letting the action of the block on the other set be the identity. The multi-set attention block can also implement a layer which simply performs the attention computation e.g. $ATTN_{XX}(X, X)$ by letting $g_X(f_{XX}, f_{XY}) = f_{XX}$. As such, multi-set attention blocks can reproduce any function on either X or Y implemented by regular transformer blocks - such as the constructions defined in Yun et al. [2020].

Let $g_c(L)$ be the iterated selective shift network defined in Yun et al. [2020], consisting of n selective shift operations followed by a final global shift layer. This results in the mapping

$$u^T g_c(L) = \widetilde{\ell}_j + \delta^{-(n+1)d}\widetilde{\ell}_n$$

where $\widetilde{\ell}_j$ is the j-th output of the selective shift layers, sorted in ascending order. We will now construct our own analogous network, $g_c(L^X, L^Y)$. Let $g_s(L)$ be the selective shift portion of $g_c(L)$. Then, we let the first δ^{-d} blocks implement $g_s(L^X)$ on L^X alone while performing the identity operation on L^Y ($T_{XY}, T_{YX} = 0$ for this component). The next δ^{-d} blocks do the same thing on L^Y, while performing the identity on L^X. The next block then applies a modified global shift to each set with attention component $\delta^{(n+1)d}\psi(\cdot; 0)$ - the same global shift as in Yun et al. [2020]. This shift is applied with attention over X, and a scaled version is also applied with attention over Y - i.e. shifting L^X by $\delta^{-(n+1)d} \max_k u^T L^X_{:,k}$ *and* $\delta^d \max_k u^T L^Y_{:,k}$ (and the same in reverse for L^Y). This can be implemented by a single MSAB block with $T_{XX} = T_{YY} = \delta^{-(n+1)d}\psi(\cdot; 0)$ and $T_{XY} = T_{YX} = \delta^d\psi(\cdot; 0)$. This comprises our $g_c(L^X, L^Y)$ block, and results in an output of

$$q^X(L^X, L^Y)_j = u^T g_c^X(L^X, L^Y)_{:,j} = \widetilde{\ell}_j^X + \delta^{-(n+1)d}\,\widetilde{\ell}_n^X + \delta^{(m+1)d}\,\widetilde{\ell}_m^Y$$
$$q^Y(L^X, L^Y)_j = u^T g_c^Y(L^X, L^Y)_{:,j} = \widetilde{\ell}_j^Y + \delta^{-(n+1)d}\,\widetilde{\ell}_m^Y + \delta^{(n+1)d}\,\widetilde{\ell}_n^X$$

C.2 PROOF OF LEMMA 6'

The proof of Lemma 6' proceeds much as the proof of Lemma 6 [Yun et al., 2020]. We must now check that all conditions are satisfied.

C.2.1 Property 2

For the second property, let us begin by considering the case where L^X, L'^X are not permutations of each other. Then, analogous to Yun et al., we have that

$$u^T g_c^X(L^X, L^Y)_{:,j} \in [\delta^{-(n+1)d}\widetilde{\ell}_n^X + \delta^{(m+1)d}\widetilde{\ell}_m^Y, \delta^{-(n+1)d}\widetilde{\ell}_n^X + \delta^{(m+1)d}\widetilde{\ell}_m^Y + \delta^{-(n+1)d+1} - \delta^{-nd+1})$$

As in Yun et al., $L^X, L'^X \in \widetilde{G_\delta}$ which are not permutations of each other must result in $\widetilde{\ell}_n^X, \widetilde{\ell}_n'^X$ differing by at least δ. By Lemma 10, distinct L^Y, L'^Y can lead to $\widetilde{\ell}_m^Y, \widetilde{\ell}_m'^Y$ differing by a value strictly less than $\delta^{-(m+1)d+1}$. The smallest net change this can result in is $\delta^{-(n+1)d} \cdot \delta - \delta^{(m+1)d} \cdot \delta^{-(m+1)d+1} = \delta^{-(n+1)d+1} - \delta$. Since this is larger than the width of the original interval and the intervals are open on at least one end, the intervals must thus be disjoint, and thus if L^X and L'^X are distinct, Q^X and Q'^X must be distinct. Now, consider the case where $L^X, L'^X \in \widetilde{G_\delta}$ are permutations of each other, but $L^Y, L'^Y \in \widetilde{G_\delta}$ are not. In this case, since $\widetilde{\ell}_m^Y, \widetilde{\ell}_m'^Y$ must differ by at least δ, Q^X and Q'^X must again be distinct. Since $|\widetilde{\ell}_m^Y - \widetilde{\ell}_m'^Y| < \delta^{-(m+1)d+1}$, the resulting change in Q^X must be strictly less than δ. Since $\widetilde{\ell}_j^X, \widetilde{\ell}_k^X$ must be separated by at least δ for $j \neq k$, $Q_j^X \neq Q_k'^X$ for any $j \neq k$. Thus, all entries of Q^X and Q'^X must be distinct in this case as well. These results apply symmetrically for Q^Y and Q'^Y, and thus this proves Property 2.

Note that if L^X, L'^X are not permutations of each other then $u^T g_c^X(L^X, L^Y), u^T g_c^X(L'^X, L'^Y)$ must be separated from each other by at least δ, whereas if L^X, L'^X are permutations of each other, but L^Y, L'^Y are not, $u^T g_c^X(L^X, L^Y), u^T g_c^X(L'^X, L'^Y)$ must be separated by at least δ^{m+2}. In general then, all entries of $u^T g_c^X(L^X, L^Y), u^T g_c^X(L'^X, L'^Y)$ must be separated from each other by at least δ^{m+2} (and conversely with δ^{n+2} for Y).

C.2.2 Properties 3-4

By the same procedure as Yun et al (in B.5.1), we can see that $q^X(L^X, L^Y)$ obeys

$$(\delta^{-2nd+1} + \delta^{2d+1})(\delta^{-d} - 1) \leq u^T g_c^X(L^X, L^Y) < (\delta^{-(2n+1)d+1} + \delta^{d+1})(\delta^{-d} - 1)$$

if $L^X, L^Y \in \widetilde{G}_\delta$. Thus, $\forall L^X, L^Y \in \widetilde{G}_\delta$ $q^X(L^X, L^Y) \in [t_l, t_r]$ where

$$\begin{aligned} t_l^X &= (\delta^{-2nd+1} + \delta^{2d+1})(\delta^{-d} - 1) \\ t_r^X &= (\delta^{-(2n+1)d+1} + \delta^{d+1})(\delta^{-d} - 1) \end{aligned}$$

The same holds for $q^Y(L^X, L^Y)$ with

$$\begin{aligned} t_l^Y &= (\delta^{-2md+1} + \delta^{2d+1})(\delta^{-d} - 1) \\ t_r^Y &= (\delta^{-(2m+1)d+1} + \delta^{d+1})(\delta^{-d} - 1) \end{aligned}$$

C.2.3 Property 1

Within Q^X and Q^Y, all entries must be distinct since $\widetilde{\ell}_1 < ... < \widetilde{\ell}_n$. Suppose $n \neq m$. Consider the bounds t_l^X, t_r^X from the previous section. Without loss of generality, suppose $m < n$. If $m = n - k$ then we have $t_r^Y = (\delta^{-(2n+1-2k)d+1} + \delta^{d+1})(\delta^{-d} - 1) < t_l^X$ and thus Q^X and Q^Y belong to disjoint intervals.

If $n = m$, then we can apply a similar argument as we did in proving Property 2, and argue that $\widetilde{\ell}_n^X, \widetilde{\ell}_n^Y$ which are not permutations must differ by at least δ. This results in intervals shifted from each other by at least $\delta^{-(n+1)d+1} - \delta^{d+1}$, which will always be larger than $\delta^{-(n+1)d+1} - \delta^{-nd+1}$, which is the width of the intervals. Thus, in this case too Q^X and Q^Y must be distinct, and Property 1 is proven.

C.2.4 Properties 5-6 - Case 1

Take $L^Y \in G_\delta^+$ and $L^X \in G_\delta \setminus \widetilde{G}_\delta$. This corresponds to an L^X with duplicate columns but within the region of compact support. In this case,

$$\widetilde{\ell}_n^X \leq \delta^{-(n-1)d+1}(\delta^{-d} - 1) - \delta(\delta^{-d} - 1)^2$$

Then,

$$\begin{aligned} u^T g_c^X(L^X, L^Y)_{:,j} &= \widetilde{\ell}_j^X + \delta^{-(n+1)d}\, \widetilde{\ell}_n^X + \delta^{(m+1)d}\, \widetilde{\ell}_m^Y \\ &\leq (\delta^{-(n+1)d} + 1)(\delta^{-(n-1)d+1}(\delta^{-d} - 1) - \delta(\delta^{-d} - 1)^2) + \delta^{(m+1)d}(\delta^{-md+1}(\delta^{-d} - 1) - \delta(\delta^{-d} - 1)^2)) \\ &< \delta^{-2nd+1}(\delta^{-d} - 1) + \delta^{-(n-1)d+1}(\delta^{-d} - 1) - \delta^{-(n+1)d+1}(\delta^{-d} - 1)^2 + \delta^{d+1}(\delta^{-d} - 1) \\ &< \delta^{-2nd+1}(\delta^{-d} - 1) + \delta^{-(n-1)d+1}(\delta^{-d} - 1)(1 + \delta^{nd} - \delta^{-2d}(\delta^{-d} - 1)) \\ &< \delta^{-2nd+1}(\delta^{-d} - 1) < t_l^X \end{aligned}$$

since $\delta^{-d} \geq 2$. Thus, Q^X falls strictly outside $[t_l^X, t_r^X]$, proving Property 6 for the case when $L^Y \in G_\delta^+$ and $L^X \in G_\delta \setminus \widetilde{G}_\delta$. The same applies in reverse for Property 7.

C.2.5 Properties 5-6 - Case 2

Take $L^Y \in G_\delta^+$ and $L^X \in G_\delta^+ \setminus G_\delta$. This corresponds to an L^X that contains at least one element outside the region of compact support. This leads to columns of L^X containing negative values. Note first that for a column $L^X_{:,j}$ containing $-\delta^{-nd}$, $\ell_j^X = \widetilde{\ell}_j^X$ as the selective shift operation does not alter it. From Yun et al, we have that $\widetilde{\ell}_j^X \leq -\delta^{-nd} + \delta^{-d+1} - 1 < 0$, and that the last layer shifts negative values by $\delta^{-(n+1)d} \min_k \widetilde{\ell}_k^X$. After this shift is applied and our additional attention-based shift is applied,

$$\begin{aligned} u^T g_c^X(L^X, L^Y)_{:,j} &\leq (-\delta^{-nd} + \delta^{-d+1} - 1)(1 + \delta^{-(n+1)d}) + \delta^{(m+1)d}\widetilde{\ell}_m^Y \\ &\leq (-\delta^{-nd} + \delta^{-d+1} - 1)(1 + \delta^{-(n+1)d}) + \delta^{(m+1)d}\delta^{-(m+1)d+1} \\ &\leq -\delta^{-(2n+1)d} + \delta^{-(n+2)d+1} + \delta < 0 < t_l^X \end{aligned}$$

where we use that $\widetilde{\ell}_m^Y \leq \delta^{-(m+1)d+1}$ and $\delta^{-1} \geq 2$. Thus, any negative column is mapped to a value outside of $[t_l^X, t_r^X]$. Note that this holds for any L^Y, including an $L^Y \in G_\delta^+ \setminus G_\delta$.

In the case where *all* columns are negative, the argument proceeds exactly as in Yun et al, and all elements are straightforwardly less than zero as shown above. In the case where only some columns are negative, the negative columns are mapped to negative values as before, and the positive columns satisfy $u^T g_c^X(L^X, L^Y)_{:,i} \geq \delta^{-(2n+2)d+}1$ before we apply our attention shift layer. If $\max_k \widetilde{\ell}_k^Y > 0$, then $u^T g_c^X(L^X, L^Y)_{:,i} \geq \delta^{(} - (2n+2)d + 1$ still holds. If not, all entries of $\widetilde{\ell}_k^Y$ must be negative, which means $\max_k \widetilde{\ell}_k^Y = -\delta^{-md}(\delta^{-d} - 1)$. We then have

$$u^T g_c^X(L^X, L^Y)_{:,i} \geq \delta^{-(2n+2)d+1} - \delta^d(\delta^{-d} - 1) > t_r^X \tag{16}$$

and thus Property 5 is proved for all cases (and symmetrically for Property 6).

C.3 PROOF OF LEMMA 7'

This proof very closely follows the proof shown in Yun et al, with a few small changes. The first layer used to map all invalid entries to strictly negative numbers becomes

$$Z \mapsto Z - (M-1)\mathbf{1}_{n+m}(\phi^X(u^T Z^X) + \phi^Y(u^T Z^Y)) \tag{17}$$

where M is the maximum value of the image of $g_c(G_\delta^+, G_\delta^+)$ and

$$\phi^X(t) = \begin{cases} 0 & t \in [t_l^X, t_r^X] \\ 1 & t \notin [t_l^X, t_r^X] \end{cases}$$
$$\phi^Y(t) = \begin{cases} 0 & t \in [t_l^Y, t_r^Y] \\ 1 & t \notin [t_l^Y, t_r^Y] \end{cases}$$

This layer is applied to both X and Y, and ensures that if either X or Y contain invalid elements, the entirety of both sets are mapped to negative values. The next d layers, which map all negative entries to the zero matrix, remain unchanged and are applied again to both X and Y.

The remaining layers must now map $g_c^X(L^X, L^Y)$ to A_L^X and $g_c^Y(L^X, L^Y)$ to A_L^Y. In a similar fashion to Yun et al, we add $\mathcal{O}\left((n+m)(1/\delta)^{d(n+m)}/(n!m!)\right)$ feedforward layers, each of which maps one value of $u^T g_c^X(L^X, L^Y)$ or $u^T g_c^Y(L^X, L^Y)$ to the correct output while leaving the others unaffected. For a given value of $u^T g_c^X(\bar{L}^X, \bar{L}^Y)_{:,j}$ these layers take the form:

$$\begin{aligned} Z^X &\mapsto Z^X + \left((A_{\bar{L}}^X)_{:,j} - g_c^X(\bar{L}^X, \bar{L}^Y)_{:,j}\right)\phi^X\left(u^T Z^X - u^T g_c^X(\bar{L}^X, \bar{L}^Y)_{:,j}\mathbf{1}_n^T\right) \\ Z^Y &\mapsto Z^Y + \left((A_{\bar{L}}^Y)_{:,j} - g_c^Y(\bar{L}^X, \bar{L}^Y)_{:,j}\right)\phi^Y\left(u^T Z^Y - u^T g_c^Y(\bar{L}^X, \bar{L}^Y)_{:,j}\mathbf{1}_m^T\right) \end{aligned}$$

wherein

$$\phi^X(t) = \begin{cases} 0 & t < -\delta^{m+2}/2 \text{ or } t \geq \delta^{m+2}/2 \\ 1 & -\delta^{m+2}/2 \leq t < \delta^{m+2}/2 \end{cases} \qquad \phi^Y(t) = \begin{cases} 0 & t < -\delta^{n+2}/2 \text{ or } t \geq \delta^{n+2}/2 \\ 1 & -\delta^{n+2}/2 \leq t < \delta^{n+2}/2 \end{cases}$$

If $Z = g_c(L^X, L^Y)$ where L^X is not a permutation of $\bar{L}^X$ and L^Y is not a permutation of $\bar{L}^Y$, then $\phi^X\left(u^T Z - u^T g_c^X(\bar{L}^X, \bar{L}^Y)_{:,j}\mathbf{1}_n^T\right) = 0$ and Z is unchanged. If on the other hand L^X *is* a permutation of $\bar{L}^X$ and L^Y *is* a permutation of $\bar{L}^Y$, with $\bar{L}^X_{:,j} = L^X_{:,i}$, then $\phi^X\left(u^T Z^X - u^T g_c^X(\bar{L}^X, \bar{L}^Y)_{:,j}\mathbf{1}_n^T\right) = (e^{(i)})^T$ and $g_c^X(L^X, L^Y)$ is mapped to $(A_L^X)_{:,i}$. Thus, this layer maps $g_c^X(L^X, L^Y)_{:,j}$ to the correct output for the specific inputs $\bar{L}^X, \bar{L}^Y$ (or permutations thereof), and does not affect any other inputs. By stacking $\mathcal{O}\left((n+m)(1/\delta)^{d(n+m)}/(n!m!)\right)$ of these layers together, we achieve the correct output for any possible inputs L^X, L^Y.