Supplementary material: Neural approximation of Wasserstein distance via a universal architecture for symmetric and factorwise group invariant functions

Samantha Chen
Department of Computer Science Engineering
University of California - San Diego
La Jolla, CA 92092
sac003@ucsd.edu

Yusu Wang
Halıcıoğlu Data Science Institute
University of California - San Diego
La Jolla, CA 92092
yusuwang@ucsd.edu

A Approximating Hausdorff distance

Suppose the underlying metric for $\mathcal{X}$ (the space of weighted point sets) is Hausdorff distance, d_H. To clarify, given $S_1 = \{(x_i, w_i) : x_i \in \mathbb{R}^d, \sum_{i=1}^n w_i\}$ and $S_2 = \{(y_i, w_i' : y_i \in \mathbb{R}^d, \sum_{i=1}^m w_i'\}$,

$$\mathrm{d}_H(S_1, S_2) = \mathrm{d}_H(\{x_1, \ldots, x_n\}, \{y_1, \ldots, y_m\}).$$

To get an encoding/decoding for $\mathcal{X}$ equipped with Hausdorff distance using max-pooling, we will use the construction given in the proof of universality in [11], which we will briefly recap here. As in Lemma 3.4, we consider a δ-net for our original space X, $\{y_1, \ldots, y_a\}$. For each y_i, we will again let $h_i(x) = e^{-\mathrm{d}_X(x, B_\delta(y_i))}$ and define $h : X \to \mathbb{R}^a$ as $h(x) = [h_1(x), \ldots, h_a(x)]$. However, instead of using sum-pooling to define $\mathbf{h} : \mathcal{X} \to \mathbb{R}^a$, we will use max-pooling as

$$\mathrm{h}(S) = \max_{x \in S} h(x)$$

where $\max$ is element-wise max. Then the decoding function $g : \mathbb{R}^a \to \mathcal{X}$ is defined as

$$g(v) = \{(y_i, \frac{v_i}{\|v\|_1}) : v_i \geq 1\}$$

Note that $v_i \geq 1$ when there is an $x_i \in S$ such that $x_i \in B_\delta(y_i)$. So for each $x_i \in S$ there is $y_i \in g \circ h(S)$ such that $\mathrm{d}_X(x_i, y_i) < \delta$. Thus, $\mathrm{d}_H(S, g \circ h(S)) < \delta$.

The final $\mathcal{N}_{\mathrm{ProductNet}}$ for approximating uniformly continuous functions for Hausdorff distance with max-pooling is

$$\mathcal{N}^{\max}_{\mathrm{ProductNet}}(A, B) = \rho_{\theta_3}\left(\phi_{\theta_2}\Big(\max_{x \in A}(h_{\theta_1}(x))\Big) + \phi_{\theta_2}\Big(\max_{y \in B}(h_{\theta_1}(y))\Big)\right).$$

Note that $\mathcal{N}^{\max}_{\mathrm{ProductNet}}$ is able to approximate Hausdorff distance to ϵ-accuracy.

B Comparison with other neural network architectures for approximating Wasserstein distance

Through the lens of 1-Wasserstein approximation, we compare the power of $\mathcal{N}_{\mathrm{ProductNet}}$ with other natural architectures, namely standard MLPs and Siamese networks. The standard MLP and Siamese networks are the natural points of comparison for $\mathcal{N}_{\mathrm{ProductNet}}$ as (1) MLPs are the go-to choice in deep learning due to their versatility and efficacy across diverse problem domains and (2) Siamese networks have gained significant prominence in the context of metric learning, frequently employed

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

for tasks aimed at learning effective similarity metrics between pairs of inputs[5, 2, 7] However, we discuss the limitations of both the standard MLP and Siamese networks for the taks of learning Wasserstein distances. For this setting, we will let (X, d_X) be some compact Euclidean space, i.e. $X \subseteq \mathbb{R}^d$ is compact and d_X is some ℓ_p norm. Then $\mathcal{X}$ be the set of weighted point sets of cardinality at most N i.e. $S = \{(x_i, w_i) : w_i \in \mathbb{R}_+, \sum_i w_i = 1, x_i \in X\}$.

Comparison with multilayer perceptrons (MLPs). First, we consider a standard MLP model where the model takes a single tensor as input. Note that for each S, there is an $\tilde{S} \in \mathcal{X}$ such that

$$\tilde{S} = \{(x_i, w_i) : (x_i, w_i) \in S\} \cup \{(x_1, 0)_{\times(N-|S|)}\}$$

where $(x_1, 0)_{\times(N-|S|)}$ means that we repeat the element $(x_1, 0)$ $(N - |S|)$ times. Notice that $\mathrm{W}_p(S, \tilde{S}) = 0$ and for any $S' \in \mathcal{X}$, $\mathrm{W}_p(S, S') = \mathrm{W}_p(\tilde{S}, S')$. If $|S| = N$, $\tilde{S} = S$ To use an MLP model to approximate Wasserstein distance, we want to represent any $(S_1, S_2) \in \mathcal{X} \times \mathcal{X}$ as an element of $\mathbb{R}^{(d+1)\times 2N}$ where $S_1 = \{(x_i, w_i) : w_i \in \mathbb{R}_+, \sum_i w_i = 1, x_i \in X\}$ and $S_2 = \{(x'_i, w'_i) : w'_i \in \mathbb{R}_+, \sum_i w'_i = 1, x'_i \in X\}$. Informally, given any "empty" element in $\mathbb{R}^{(d+1)\times 2N}$, we will map S_1 to the first N columns and map S_2 to the last N columns. Formally, we will define the mapping $\beta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{(d+1)\times 2N}$ given S_1 and S_2 as defined above as

$$\beta(S_1, S_2) = \beta(\tilde{S}_1, \tilde{S}) = \begin{bmatrix} x_1 & \cdots & x_N & x'_1 & \cdots & x'_N \\ w_1 & \cdots & w_N & w'_1 & \cdots & w'_N \end{bmatrix}$$

where we abuse notation slightly and use (x_i, w_i) and (x'_i, w'_i) to describe the elements in $\tilde{S}_1$ and $\tilde{S}_2$ respectively. Given $M \in \mathbb{R}^{(d+1)\times 2N}$, define $\beta^{-1} : \mathrm{Image}(\beta) \to \mathcal{X} \times \mathcal{X}$ as

$$\beta^{-1}\left(\begin{bmatrix} x_1 & \cdots & x_N & x'_1 & \cdots & x'_N \\ w_1 & \cdots & w_N & w'_1 & \cdots & w'_N \end{bmatrix}\right) = (\{(x_i, w_i)\}, \{(x'_i, w'_i)\})$$

Now, in order to use the classical universal approximation theorem to approximate W_p to an ϵ-approximation error, we must show that there is a continuous function $f : \mathrm{Image}(\beta) \to \mathbb{R}$ such that $\mathrm{W}_p(S_1, S_2) = f(\beta(S_1, S_2))$.

Lemma B.1 *Given $\beta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{(d+1)\times 2N}$ as defined previously, there is a continuous function $f : \mathrm{Image}(\beta) \to \mathbb{R}$ such that for any $\epsilon > 0$, and for any $(S_1, S_2) \in \mathcal{X} \times \mathcal{X}$, $|\mathrm{W}_p(S_1, S_2) - f(\beta(S_1, S_2))| < \epsilon$.*

Proof: Let $\epsilon > 0$. From Corollary 3.5, there is a $\delta > 0$ such that are continuous $h : X \to \mathbb{R}^{a(\delta)}$, $\phi : \mathbb{R}^{a(\delta)} \to \mathbb{R}^{a'}$, and $\rho : \mathbb{R}^{a'} \to \mathbb{R}$, such that for any $A, B \in \mathcal{X}$

$$\left|\mathrm{W}_p(A, B) - \rho\Big(\phi\Big(\sum_{x\in A} w_x h(x)\Big) + \phi\Big(\sum_{x\in B} w_x h(x)\Big)\Big)\right| < \epsilon.$$

Let $(S_1, S_2) \in \mathcal{X} \times \mathcal{X}$ and let $M = \beta(S_1, S_2)$. Note that $\beta^{-1}(M) = (S_1, S_2)$. Let $M \in \mathrm{Image}(\beta)$, let $M[:, i]$ denote the ith column vector of M, let $M[: d, i]$ denote vector of the first d elements in the ith column vector of M and let $M[i, j]$ denote the ijth entry of M. We know that $M[:, i] \in \mathbb{R}^{d+1}$. Since $M[:, i] \in \mathrm{Image}(\beta)$, we know that $M[: d, i] \in X$ and $M[d, i]$ corresponds to a weight. Furthermore, since h is a continuous mapping from $X \to \mathbb{R}$, $M[d, i]h(M[: d, i])$ is continuous. Thus, since ϕ is also continuous, the function $f' : \mathrm{Image}(\beta) \to \mathbb{R}^{a'}$

$$\phi(\sum_{i=1}^{N} M[d, i]h(M[: d, i])) + \phi(\sum_{i=N+1}^{2N} M[d, i]h(M[: d, i]))$$

is continuous. Since ρ is also continuous, we get that $\rho \circ f' : \mathrm{Image}(\beta) \to \mathbb{R}$ is continuous. Thus, $f : \mathrm{Image}(\beta) \to \mathbb{R}$ which is $f = \rho \circ f'$ is also continuous and $|f(\beta(S_1, S_2)) - \mathrm{W}_p(S_1, S_2)| < \epsilon$ by the property from Corollary 3.5 ■

Since f in the above Lemma operates on fixed dimensional space, we can now use an MLP to approximate it. Then by the above Lemma and universal approximation of MLPs, there is an MLP

which can approximate $\mathrm{W}_p : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ to an arbitrary ϵ additive error. However, this approach has the following issues: while an MLP can approximate W_p to an arbitrary ϵ additive error, the model complexity of the MLP depends on not only the approximation error, but also on the maximum size N of the input point set. In contrast, our neural network $\mathcal{N}_{\mathrm{ProductNet}}$ introduced in Eqn (2) has model complexity which is independent of N; see Corollary 3.6. Furthermore, the function represented by the MLP (that we use to approximate ϕ) however may not be symmetric to the two point sets, and may not guarantee permutation invariance with respect to $G \times G$. In practice, computationally expensive techniques such as data augmentation are often required in order for an unconstrained MLP to approximate a structured function such as our Wasserstein distance function, which is an SFGI-function.

Comparison with Siamese architectures. As mentioned previously, one common way of learning product functions is to use a Siamese network which embeds $\mathcal{X}$ to Euclidean space and then use some ℓ_p norm to approximate the desired product function. Consider the learning of Wasserstein distances again. To approximate $W_p(A, B)$ for two point sets A and B. The Siamese network will first embed each point set to a fixed dimensional Euclidean space $\mathbb{R}^D$ via a function ϕ_θ modeled by a neural network, and then compute $\|\phi_\theta(A) - \phi_\theta(B)\|_q$ for some $1 \leq q < \infty$. Intuitively, compared to our neural network $\mathcal{N}_{\mathrm{ProductNet}}$ introduced in Eqn (2) (and recall Figure 1), the Siamese network will replace the outer function ρ by simply the L_q-norm of $\phi_\theta(A) - \phi_\theta(B)$. However, this approach intuitively requires that one can find a near-isometric embedding of the Wasserstein distance to the L_q distance in a Euclidean space. However, there exists lower bound results on the distortion incurred when embedding Wasserstein distance to L_p space. In the following section, we will review one such result on the lower bound of metric distortions when embedding the Wasserstein distance to L_1 space; implying that if we chose $q = 1$, then in the worst case the Siamese network will incur at least as much distortion as that lower bound.

C Non-embeddability theorems for Wasserstein distances

Here we summarize results pertaining to the limitations of embedding Wasserstein distance to the L_q distance in a Euclidean space. Consider probability distributions over a grid of points in $\mathbb{R}^2$, $\{0, 1, \ldots, D\}^2$ equipped with the 1-Wasserstein distance, $(\mathcal{P}(\{0, 1, \ldots, D\}^2), \mathrm{W}_1)$. Let L_1 denote the space of Lebesgue measureable functions $f : [0, 1] \to \mathbb{R}$ such that

$$\|f\|_1 = \int_0^1 |f(t)|dt.$$

Given a mapping $F : (\mathcal{P}(\{0, 1, \ldots, n\}^2), \mathrm{W}_p) \to L_1$ such that for any $\mu, \rho \in (\mathcal{P}(\{0, 1, \ldots, D\}^2), \mathrm{W}_p)$,

$$\mathrm{W}_p(\mu, \rho) \leq \|F(\mu) - F(\rho)\| \leq L \cdot \mathrm{W}_p(\mu, \rho) \tag{1}$$

the *distortion* is the value of L.

Theorem C.1 ([9]) *Any embedding $(\mathcal{P}(\{0, 1, \ldots, n\}^2, \mathrm{W}_1) \to L_1$ must incur distortion at least $\Omega(\sqrt{\log D})$*

For $(\mathcal{P}(\{0, 1, \ldots, D\}^d), \mathrm{W}_1)$ where $d \geq 2$, $\mathcal{P}(\{0, 1, \ldots, D\}^d$ contains $\mathcal{P}(\{0, 1, \ldots, D\}^2)$ so the lower bound $O(\sqrt{D})$ still applies for L_1 embeddings from $\mathcal{P}(\{0, 1, \ldots, D\}^d)$, $d > 2$.

From the above results, we can see that for any Siamese architecture (even in the simple case of finite point sets in $\mathbb{R}^2$ and $\mathbb{R}^3$), we are unable to approximate the Wassserstein distances via any L_1 embedding.

Corollary C.2 *Given a neural network $\mathcal{N}_{\mathrm{Siamese}} : \mathcal{X} \to \mathbb{R}^d$ where $\mathcal{X}$ are weighted point sets over $\{0, 1, \ldots, D\}^2$, $\|\mathcal{N}_{\mathrm{Siamese}}(\mu) - \mathcal{N}_{\mathrm{Siamese}}(\nu)\|_1$ incurs distortion at least $\Omega(\sqrt{\log D})$.*

Note that if we consider our input for a Siamese architecture to be finite point sets over $\{0, 1, \ldots, D\}^2$, we allow multisets so the input set size is not bounded by D.

Table 1: Size of the output point set from the WPCE decoder for each dataset.

	noisy-sphere-2	noisy-sphere-6	uniform	ModelNet-small	ModelNet-large	RNAseq
Size	200	200	256	100	2048	100

D Experimental details

D.1 Baseline models for comparison with $\mathcal{N}_{\mathrm{ProductNet}}$

Here, we will detail two baseline neural networks in our experiments.

Wasserstein point cloud embedding (WPCE) network First defined by [6], the WPCE network is an Siamese autoencoder architecture. It consists of an encoder network $\mathcal{N}_{\mathrm{encoder}}$ and a decoder network $\mathcal{N}_{\mathrm{decoder}}$. WPCE takes as input two point sets $P, Q \subseteq \mathbb{R}^d$. $\mathcal{N}_{\mathrm{encoder}}$ is a permutation invariant neural network - which we chose to be DeepSets. In other words,

$$\mathcal{N}_{\mathrm{encoder}}(P) = \phi_{\theta_2}\Big(\sum_{x \in P} h_{\theta_1}(x)\Big).$$

Note that one may also choose PointNet to be $\mathcal{N}_{\mathrm{encoder}}$. However, in our experiments, we did not see a large difference in approximation quality between using PointNet and DeepSets (where the difference between the two amounts to using a sum vs. max aggregator). For sake of consistency with $\mathcal{N}_{\mathrm{ProductNet}}$, we chose to use a sum aggregator for $\mathcal{N}_{\mathrm{encoder}}$ (DeepSets). The decoder network, $\mathcal{N}_{\mathrm{decoder}}$, is a fully connected neural network which outputs a fixed-size point set in $\mathbb{R}^d$. WPCE then uses a Sinkhorn reconstruction loss term to regularize the embedding produced by the encoder. Thus, given a set of paired input point sets and their Wasserstein distance $\{(P_i, Q_i, \mathrm{W}_1(P_i, Q_i)) : i \in [N]\}$ the loss which we optimize over for WPCE is

$$\begin{aligned}\mathcal{L}(P, Q) = \frac{1}{N}\sum \Big(\|\mathcal{N}_{\mathrm{encoder}}(P_i) - \mathcal{N}_{\mathrm{decoder}}(Q_i)\|_2 - W_1(P_i, Q_i)\Big)^2 \\ + \lambda\Big(\frac{1}{N}\sum S_\epsilon(\mathcal{N}_{\mathrm{decoder}}(\mathcal{N}_{\mathrm{encoder}}(P_i)) + \frac{1}{N}\sum S_\epsilon(\mathcal{N}_{\mathrm{decoder}}(\mathcal{N}_{\mathrm{encoder}}(Q_i))\Big) \quad (2)\end{aligned}$$

where λ is some constant in $\mathbb{R}$ that controls the balance between the two loss terms and S_ϵ is the Sinkhorn divergence between P_i and Q_i. Note that ϵ in S_ϵ refers to the regularization parameter for the Sinkhorn divergence. For our experiments, we chose $\lambda = 0.1$ and $\epsilon = 0.1$. Furthermore, for each dataset used in our experiments, we set used a range of different sizes for the fixed-size output point set. This parameter is summarized per dataset in Table 1.

Siamese DeepSets. The baseline Siamese DeepSets model, $\mathcal{N}_{\mathrm{SDeepSets}}(\cdot, \cdot)$, consists of a single DeepSets model which maps two input point sets to some Euclidean space $\mathbb{R}^d$. The ℓ_2-norm between the final embeddings is then used as the final estimate for Wasserstein distance. Formally, let $\mathcal{N}_{\mathrm{DeepSets}}(P) = \phi_{\theta_2}\Big(\sum_{x \in P} h_{\theta_1}(x)\Big)$, where ϕ_{θ_2} and h_{θ_1} are both MLPs, be the DeepSets model. Then given two point sets P, Q, the final approximation of Wasserstein distance given by $\mathcal{N}_{\mathrm{SDeepSets}}(P, Q)$ is

$$\mathcal{N}_{\mathrm{SDeepSets}}(P, Q) = \|\mathcal{N}_{\mathrm{DeepSets}}(P) - \mathcal{N}_{\mathrm{DeepSets}}(Q)\|_2.$$

D.2 Training and implementation details

Datasets. We used several synthetic datasets as well as the ModelNet40 point cloud dataset. Furthermore, we used two different types of synthetic datasets. We construct the 'noisy-sphere-d' dataset by sampling pairs of point clouds from four d-dimensional spheres centered at the origin with increasing radiuses of 0.25, 0.5, 0.75, and 1.0. For our experiments, we used 'noisy-sphere-3' and 'noisy-sphere-6'. Finally, the 'uniform' dataset of points sets in $\mathbb{R}^2$ is constructed by sampling points sets from the uniform distribution on $[-4, 4] \times [-4, 4]$. The full details and names of each dataset are summarized in Table 2.

Table 2: Summary of details for each dataset. Note that min and max refer to the minimum and maximum number of points per input point set.

	dim	min	max	training pairs	val pairs
noisy-spheres-2	3	100	300	2000	200
noisy-spheres-6	6	100	300	3600	400
uniform	2	256	256	3000	300
ModelNet-small	3	20	200	3000	300
ModelNet-large	3	2048	2048	4000	400
RNAseq	2000	20	200	3000	300

Implementation. We implement all neural network architectures using the PyTorch[10] and GeomLoss [3] libraries. The ground truth 1-Wasserstein distances and Sinkhorn approximations were computed using Python Optimal Transport (POT) [4]. Note that for large point sets and for higher dimensional datasets, there is often a high degree of numerical instability in the POT implementation of the Sinkhorn algorithm. In these cases (ModelNet-large and RNAseq) we used our own implementation of the Sinkhorn algorithm. For each model, we used the Leaky-ReLU as the non-linearity. To train each model, we set the batch size for each dataset to be 128 and the learning rate to 0.001. All models were trained on an Nvidia RTX A6000 GPU. For both $\mathcal{N}_{\text{ProductNet}}$ and $\mathcal{N}_{\text{SDeepSets}}$, given two input point sets, we minimize on the mean squared error between the approximation produced by the network and the true Wasserstein distance. In other words, given two point sets P, Q, the loss for $\mathcal{N}_{\text{ProductNet}}$ is defined as $\mathcal{L}_{\mathcal{N}_{\text{ProductNet}}}(P,Q) = \Big(\mathcal{N}_{\text{ProductNet}}(P,Q) - \text{W}_1(P,Q)\Big)^2$ and the loss for Siamese DeepSets is $\mathcal{L}_{\mathcal{N}_{\text{SDeepSets}}}(P,Q) = \Big(\mathcal{N}_{\text{SDeepSets}}(P,Q) - \text{W}_1(P,Q)\Big)^2$. For WPCE, we train the network using the loss function defined in Eq. 2. Note that for $\mathcal{N}_{\text{ProductNet}}$, the hyperparameters are the width, depth, and output dimension for the MLPs which represent h_{θ_1} and ϕ_{θ_2} and the width and depth the MLP which represents ρ_{θ_3}. For WPCE, we set the decoder to a three layer neural network with width 100 and adjusted the width, depth, and output dimension for the MLPs which represent ϕ_{θ_2} and h_{θ_1} in $\mathcal{N}_{\text{encoder}}$. To find the best model for each architecture, we randomly sampled hyperparameter configurations and conducted a hyperparameter search over 85 models for $\mathcal{N}_{\text{ProductNet}}$ and 75 models for both WPCE and $\mathcal{N}_{\text{SDeepSets}}$.

D.3 Approximating 2-Wasserstein distance

To further show the use of our model, we additionally approximate the 2-Wasserstein distance; see Table 5 for results. The experimental set-up is the same as for 1-Wasserstein distance and we largely see the same trends as we see for 1-Wasserstein distance; that is, $\mathcal{N}_{\text{ProductNet}}$ outperforms all other neural network implementations. Note that Table 5 shows that the Sinkhorn approximation with $\epsilon = 0.01$ is more accurate than $\mathcal{N}_{\text{ProductNet}}$. However, as the ϵ parameter for the Sinkhorn approximation decreases, the computation time increases. In Table **??**, we show that the Sinkhorn approximation is already much slower than $\mathcal{N}_{\text{ProductNet}}$ at $\epsilon = 0.1$ while also having a less accurate approximation. The Sinkhorn approximation with $\epsilon = 0.01$ (reported in Table 5 is slower the Sinkhorn approximation with $\epsilon = 0.1$ and additionally, is also much slower than $\mathcal{N}_{\text{ProductNet}}$ at inference time.

D.4 Generalization to large point sets

In addition to the results reported in Tables 1 and 5, which record the average relative error for point sets of unseen sizes during training, we have also included several plots in Figures 2 and 1 which demonstrate how the error for each approximation method changes as the input point set size increases for both 1-Wasserstein distance and 2-Wasserstein distance. Observe that for ModelNet-small, noisy-sphere-3, and noisy-sphere-6, the error for $\mathcal{N}_{\text{ProductNet}}$ exhibits a significantly slower increase as compared to WPCE. The rate at which the error increases is not as evident for the uniform and ModelNet-large datasets. It is worth mentioning that we trained the model on fixed-size input for both of these datasets. It is possible that training with fixed-size input leads a rapid deterioration in approximation quality for WPCE and $\mathcal{N}_{\text{SDeepSets}}$ when dealing with point sets of sizes not seen at training time. Furthermore, consider that WPCE may be especially sensitive to differences in input

Table 3: Time for 500 1-Wasserstein distance computations in seconds. Note that we chose input point set size to be the maximum possible point set size that we trained on. Additionally, the Sinkhorn distance reported uses $\epsilon = 0.1$ as the regularization parameter. Note that as ϵ decreases, the error incurred by the Sinkhorn approximation will decrease but the computation time will also increases. Here, when $\epsilon = 0.1$, the Sinkhorn approximation is already much slower than the neural network approximations while being less accurate.

		Models				
Dataset	Input size	$\mathcal{N}_{\mathrm{ProductNet}}$	WPCE	$\mathcal{N}_{\mathrm{SDeepSets}}$	Sinkhorn	Ground truth
noisy-sphere-3	300	1.050	0.676	0.4904	2.591	2.813
noisy-sphere-6	300	0.752	0.491	0.503	1.986	6.6770
uniform	256	0.155	0.184	0.137	15.113	1.018
ModelNet-small	200	0.330	0.330	0.191	2.074	1.615
ModelNet-large	2048	1.174	1.571	0.612	239.448	254.947
RNAseq	200	1.128	0.856	0.792	92.153	105.908

Table 4: Comparison of the mean relative error versus overall computation time for 500 approximations of 1-Wasserstein distance for $\mathcal{N}_{\mathrm{ProductNet}}$ and the Sinkhorn distance. Note that the input point set size is the same as in Table 3 for each dataset. The parameter ϵ controls the accuracy of the Sinkhorn approximation with lower ϵ corresponding to a more accurate approximation once the Sinkhorn algorithm converges. However, notice that in some cases, the Sinkhorn algorithm with $\epsilon = 0.01$ has a higher relative error than the Sinkhorn algorithm with $\epsilon = 0.1$ as the algorithm fails to converge within a reasonable number of iterations (1000).

Dataset		$\mathcal{N}_{\mathrm{ProductNet}}$	Sinkhorn ($\epsilon = 0.10$)	Sinkhorn ($\epsilon = 0.01$)
ModelNet-small	Error	0.084 ± 0.077	0.187 ± 0.232	0.011 ± 0.003
	Time (s)	0.330	2.074	104.712
ModelNet-large	Error	0.140 ± 0.206	0.148 ± 0.048	0.026 ± 0.008
	Time (s)	1.174	239.448	1930.852
Uniform	Error	0.097 ± 0.073	0.073 ± 0.009	0.023 ± 0.098
	Time (s)	0.155	15.113	63.028
noisy-sphere-3	Error	0.046 ± 0.043	0.187 ± 0.232	0.162 ± 0.132
	Time (s)	1.050	2.591	214.185
noisy-sphere-6	Error	0.015 ± 0.014	0.137 ± 0.122	0.326 ± 0.135
	Time (s)	0.752	1.986	101.763
RNAseq	Error	0.012 ± 0.010	0.040 ± 0.009	0.035 ± 0.013
	Time (s)	1.128	92.153	91.573

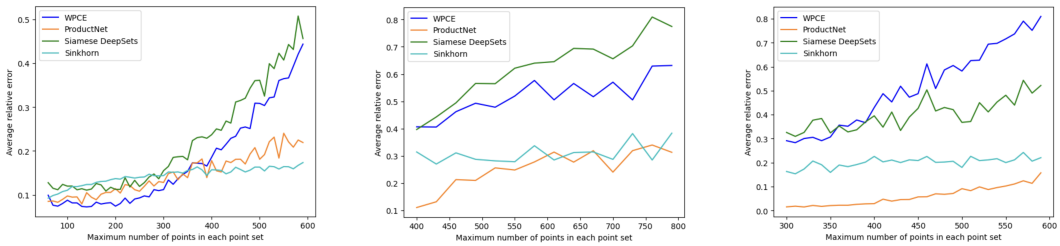
sizes at testing time as training WPCE depends on minimizing the Wasserstein difference between the input point set and a fixed-size decoder point set which may cause the model to be overly specialized to point sets of a fixed input size. This observation could provide an explanation for the observed plots in both the ModelNet-large and uniform cases. Finally, as predicted in our theoretical analysis, the performance of the model degrades for higher dimensional datasets i.e. the RNAseq dataset.

E Extra proofs

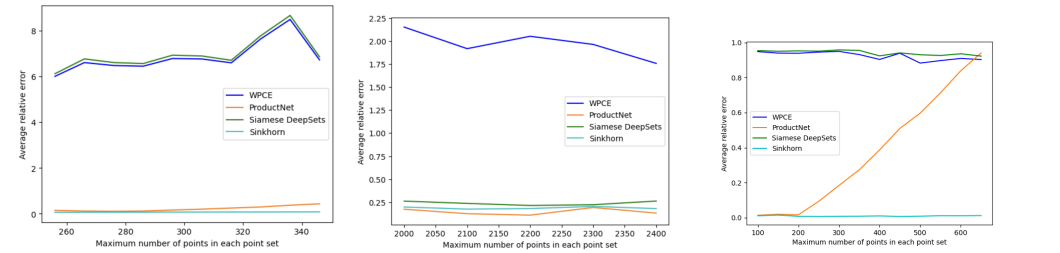
E.1 Proof of DeepSets Universality.

Here we will provide extra details on the proof of Theorem 2.1 using multisymmetric polynomials. Note that multisymmetric polynomials were previously used in [12] and [8] to show universality for equivariant set networks and arbitrary G-invariant neural networks for any permutation group G.

Proof: To begin, we will first define multisymmetric polynomials and power sum multisymmetric polynomials. Let $A[y_1, \ldots, y_n]$ be the ring of polynomials in n variables with coefficients in a ring A.

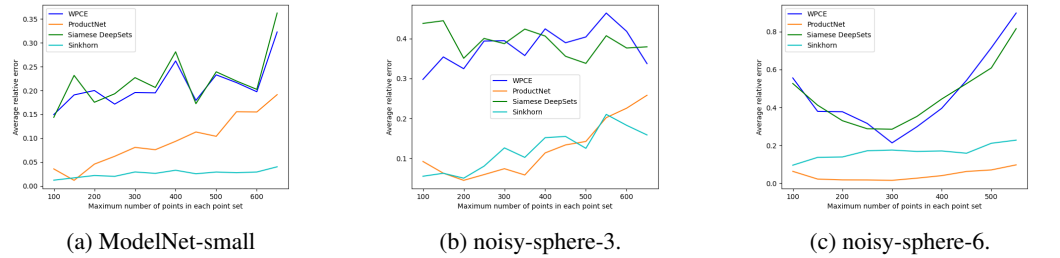


(a) ModelNet-small (b) noisy-sphere-3. (c) noisy-sphere-6.

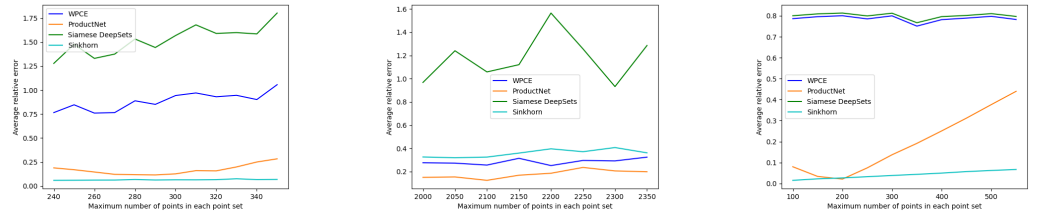


(d) uniform. (e) ModelNet-large. (f) RNAseq.

Figure 1: Average error for 1-Wasserstein approximation for each model as the maximum number of points increases. Note that the Sinkhorn approximation is the $\epsilon = 0.1$. These graphs include point set sizes not seen at training time to display how each approximation performs on unseen examples. Note that especially for ModelNet-small, noisy-sphere-3, and noisy-sphere-6, we can see that the error for $\mathcal{N}_{\text{ProductNet}}$ increases at a slower rate than WPCE.



(a) ModelNet-small (b) noisy-sphere-3. (c) noisy-sphere-6.



(d) uniform. (e) ModelNet-large. (f) RNAseq.

Figure 2: Average 2-Wasserstein error for each model as the maximum number of points increases. Note that for all datasets, the Sinkhorn error is with $\epsilon = 0.10$.

Table 5: Mean relative error between approximations and 2-Wasserstein distance between point sets. The top row for each dataset shows the error for point sets with input sizes that were seen at training time; while the bottom row shows the error for point sets with input sizes that were not seen at training time. Note that the Sinkhorn approximation is computed with the regularization parameter set to $\epsilon = 0.01$.

Dataset	Input size	$\mathcal{N}_{\text{ProductNet}}$	WPCE	$\mathcal{N}_{\text{SDeepSets}}$	Sinkhorn
noisy-sphere-3	[100, 300]	**0.054 ± 0.071**	0.291 ± 0.201	0.400 ± 0.336	0.078 ± 0.186
	[400, 600]	0.188 ± 0.197	0.387 ± 0.386	0.427 ± 0.375	**0.161 ± 0.311**
noisy-sphere-6	[100, 300]	0.024 ± 0.010	0.331 ± 0.237	0.358 ± 0.231	**0.019 ± 0.057**
	[400, 600]	0.092 ± 0.074	0.434 ± 0.598	0.623 ± 0.596	**0.050 ± 0.039**
uniform	256	**0.112 ± 0.082**	0.221 ± 0.162	0.241 ± 0.171	0.182 ± 0.044
	[200, 300]	0.175 ± 0.123	2.431 ± 2.162	4.058 ± 3.324	0.055 ± 0.053
ModelNet-small	[20, 200]	0.078 ± 0.095	0.178 ± 0.148	0.183 ± 0.148	**0.023 ± 0.059**
	[400, 600]	0.163 ± 0.151	0.216 ± 0.252	0.227 ± 0.179	**0.034 ± 0.031**
ModelNet-large	2048	0.187 ± 0.335	0.281 ± 0.203	0.538 ± 0.298	**0.172 ± 0.065**
	[1800, 2000]	0.185 ± 0.302	0.523 ± 0.526	33.086 ± 28.481	**0.046 ± 0.039**
RNAseq	[20, 200]	0.049 ± 0.029	0.508 ± 0.291	0.490 ± 0.271	**0.024 ± 0.009**
	[400, 600]	**0.281 ± 0.057**	0.533 ± 0.300	0.568 ± 0.317	0.987 ± 0.0002

Table 6: Comparison of the mean relative error versus overall computation time for 300 approximations of 2-Wasserstein distance for $\mathcal{N}_{\text{ProductNet}}$ and the Sinkhorn distance. The parameter ϵ controls the accuracy of the Sinkhorn approximation with lower ϵ corresponding to a more accurate approximation.

Dataset		$\mathcal{N}_{\text{ProductNet}}$	Sinkhorn ($\epsilon = 0.10$)	Sinkhorn ($\epsilon = 0.01$)
ModelNet-small	Error	0.078 ± 0.097	0.232 ± 0.132	0.019 ± 0.057
	Time (s)	0.208	1.165	9.857
ModelNet-large	Error	0.187 ± 0.335	0.363 ± 0.255	0.172 ± 0.089
	Time (s)	2.841	6.079	36.265
uniform	Error	0.112 ± 0.082	0.0303 ± 0.022	0.182 ± 0.044
	Time (s)	0.712	16.515	29.312
noisy-sphere-3	Error	0.054 ± 0.071	0.225 ± 0.093	0.078 ± 0.186
	Time (s)	0.677	1.591	12.760
noisy-sphere-6	Error	0.024 ± 0.010	0.324 ± 0.316	0.023 ± 0.059
	Time (s)	0.428	0.877	9.831
RNAseq	Error	0.049 ± 0.029	0.031 ± 0.014	0.024 ± 0.009
	Time (s)	0.716	47.701	81.016

Definition E.1 (Multisymmetric polynomials) *The multisymmetric polynomials on the n families of k variables $\mathbf{x}_1 \ldots, \mathbf{x}_n$ where $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,k})$ are those polynomials that remain unchanged under every permutation of the n families, $\mathbf{x}_1, \ldots, \mathbf{x}_n$.*

Let A be a ring. The *algebra of multisymmetric polynomials* in n families of k variables with coefficients in A is denoted $\mathfrak{J}_n^k(A)$.

Furthermore, we define the multisymmetric power-sum polynomials:

Definition E.2 (Multisymmetric power-sum polynomials) *Let $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{N}^k$. Given $\mathbf{x} = (x_1, \ldots, x_k)$, let*

$$\mathbf{x}^\alpha = x_1^{\alpha_1} \cdots x_k^{\alpha_k}$$

The multisymmetric power sum with multi-degree α is

$$p_\alpha = \sum_i^n \mathbf{x}_i^\alpha$$

Among them, we will consider the set of elementary multisymmetric power sums to be those with $|\alpha| \leq n$.

Notice that there $t = \binom{n+k}{k}$ multisymmetric power sums. Let $\alpha_1, \ldots, \alpha_t$ be the list of all $\alpha \in \mathbb{N}^k$ such that $|\alpha| \leq n$.

Theorem E.3 ([1]) *Let A be a ring in which $n!$ is invertible. Then the multisymmetric power sums generate $\mathfrak{J}_n^r(A)$ as an A-algebra.*

If we take $A = \mathbb{R}$, we get that the multisymmetric power sum polynomials generate $\mathfrak{J}_N^k$. Now given a continuous function $f : \mathbb{R}^{k \times N} \to \mathbb{R}$ which is invariant to permutations of the columns, we know that f can be approximated by a polynomial p which is invariant to permutations of columns (see [8] for a detailed argument). Such a polynomial p is a multisymmetric polynomial in N families of k variables with coefficients in $\mathbb{R}$ i.e., $p \in \mathfrak{J}_N^k$. Given $\mathbf{x} \in \mathbb{R}^k$,

$$\phi(\mathbf{x}) = [x^{\alpha_1}, \ldots, x^{\alpha_t}]$$

Then

$$\sum_{i=1}^{N} \phi(\mathbf{x}_i) = \begin{bmatrix} \sum_i^N x_i^{\alpha_1} \\ \sum_i^N x_i^{\alpha_2} \\ \vdots \\ \sum_i^N x_i^{\alpha_t} \end{bmatrix} = \begin{bmatrix} p_{\alpha_1} \\ p_{\alpha_2} \\ \vdots \\ p_{\alpha_t} \end{bmatrix}$$

By Theorem E.3, we have that $p_{\alpha_1}, \ldots, p_{\alpha_t}$ will generate any polynomial in $\mathfrak{J}_N^k$. Then we have some polynomial $q \in \mathbb{R}[y_1, \ldots, y_t]$ such that $p = q(p_{\alpha_1}, \ldots, p_{\alpha_t})$. ■

E.2 Proofs from Section 3

E.2.1 Proof of Lemma 3.2

Proof: Let $\epsilon > 0$. Since f is uniformly continuous, $\exists \delta$ such that for all $(A_1, \ldots, A_m), (A'_1, \ldots, A'_m) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$ where $\mathrm{d}_{\mathcal{X}_1 \times \cdots \times \mathcal{X}_k}((A_1, \ldots, A_m), (A'_1, \ldots, A'_m)) < \delta$, we have $|f(A_1, \ldots, A_m) - f(A'_1, \ldots, A'_m)| < \epsilon$. Since for any $\delta > 0$ and any $i \in [m]$, $(\mathcal{X}_i, \mathrm{d}_{\mathcal{X}_i})$ has a (δ, a, G)-sketch, we know that there is an $a_i \in \mathbb{N}^+$ where there are continuous $h_i : \mathcal{X}_i \to \mathbb{R}^{a_i}$ and $g_i : \mathbb{R}^{a_i} \to \mathcal{X}_i$ where $\mathrm{d}_{\mathcal{X}_i}(g_i \circ h_i(A), A) < \delta/m$ for each $A \in \mathcal{X}_i$.

Let $g' : \mathbb{R}^{a_1} \times \cdots \times \mathbb{R}^{a_m} \to \mathcal{X}_i \times \cdots \times \mathcal{X}_m$ be defined as $(u_1, \ldots, u_m) \mapsto (g_1(u_1), \ldots, g_m(u_m))$. Since $\mathrm{d}_{\mathcal{X}_i}(g_i \circ h_i(A_i), A_i) < \delta/k$ for $A_i \in \mathcal{X}_i$ and $i \in [m]$,

$$\mathrm{d}_{\mathcal{X}_1 \times \cdots \times \mathcal{X}_m}((g_1 \circ h_1(A_1), \ldots, g_m \circ h_m(A_m)), (A_1, \ldots, A_m)) < \delta.$$

Let $\rho = f \circ g'$ and $\phi_i = h_i$ Then

$$\begin{aligned} |f(A_1, \ldots, A_m) - \rho(\phi_1(A_1), \ldots, \phi_k(A_m))| &= |f(A_1, \ldots, A_m) - f \circ g'(h_1(A_1), \ldots, h_m(A_m))| \\ &= |f(A_1, \ldots, A_m) - f(g \circ h(A_1), \ldots, g \circ h(A_m))| < \epsilon. \end{aligned}$$

Note that if $\mathcal{X}_1 = \mathcal{X}_2 = \cdots = \mathcal{X}_m$ and $G_1 = G_2 = \cdots = G_m$, we can use the same encoding and decoding function - h_i and g_i, respectively - for all $\mathcal{X}_i$. Thus, in this case, $\phi_1 = \phi_2 = \cdots = \phi_m$. ■

E.2.2 Proof of Lemma 3.3

Proof: Using the same argument as Theorem 3.2, we know that for $\epsilon/2$, there is a continuous $h : \mathcal{X} \to \mathbb{R}^a$ and $g : \mathbb{R}^a \to \mathbb{R}$ such that

$$|f(A_1, \ldots, A_m) - f(g \circ h(A_1), \ldots, g \circ h(A_m))| < \frac{\epsilon}{2}.$$

As before, let $g' : \mathbb{R}^{a \times m} \to \mathbb{R}$ be $g'(u_1, \ldots, u_m) = (g(u_1), \ldots, g(u_m))$. Take $F : \mathbb{R}^{a \times m} \to \mathbb{R}$ as $F(u_1, \ldots, u_m) = f \circ g'(u_1, \ldots, u_m) = f(g(u_1), \ldots, g(u_m))$ where u_i represents the ith column of an element in $\mathbb{R}^{a \times m}$. Note that F is continuous and invariant to permutations of the columns. Let $t = \binom{a+m}{m}$. Therefore, by Theorem 2.1, there is a $\gamma : \mathbb{R}^a \to \mathbb{R}^t$ and ρ such that

$$\left| f \circ g'(v_1, \ldots, v_m) - \rho\Big(\sum_{i=1}^{m} \gamma(v_i) \Big) \right| < \frac{\epsilon}{2}$$

Now set $\phi = \gamma \circ h$ which is a function from $\mathbb{R}^a \to \mathbb{R}^t$. We thus have that

$$\Big|f(A_1,\dots,A_m) - \rho\Big(\sum_{i=1}^{m}\phi(A_i)\Big)\Big|$$
$$= \left|f(A_1,\dots,A_m) - f\circ g'(h(A_1),\dots,h(A_m)) + f\circ g'(h(A_1),\dots,h(A_m)) - \rho\Big(\sum_{i=1}^{m}\phi(A_i)\Big)\right|$$
$$< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$$

This completes the proof. ■

References

[1] E. Briand. When is the algebra of multisymmetric polynomials generated by the elementary multisymmetric polynomials? *Beiträge zur Algebra und Geometrie: Contributions to Algebra and Geometry, 45 (2), 353-368.*, 2004.

[2] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

[3] J. Feydy, T. Séjourné, F.-X. Vialard, S.-i. Amari, A. Trouve, and G. Peyré. Interpolating between optimal transport and mmd using sinkhorn divergences. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2681–2690, 2019.

[4] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[5] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[6] K. Kawano, S. Koide, and T. Kutsuna. Learning wasserstein isometric embedding for point clouds. In *2020 International Conference on 3D Vision (3DV)*, pages 473–482. IEEE, 2020.

[7] G. Koch, R. Zemel, R. Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[8] H. Maron, E. Fetaya, N. Segol, and Y. Lipman. On the universality of invariant networks. In *International conference on machine learning*, pages 4363–4371. PMLR, 2019.

[9] A. Naor and G. Schechtman. Planar earthmover is not in l_1. *SIAM Journal on Computing*, 37(3):804–826, 2007.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[11] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[12] N. Segol and Y. Lipman. On universal equivariant set networks. *arXiv preprint arXiv:1910.02421*, 2019.