Efficient Neural Approximations of Geometric Problems

Yusu Wang

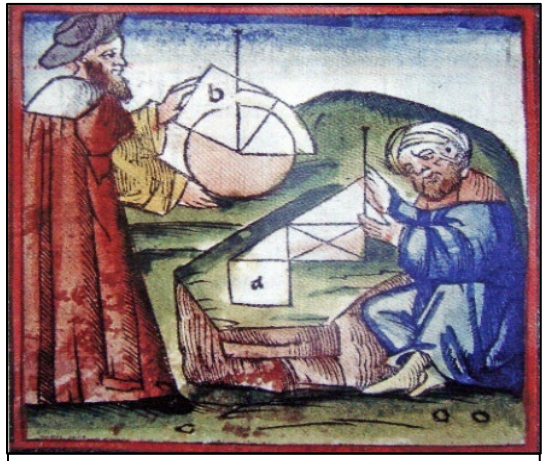
Halıcıoğlu Data Science Institute (HDSI) @ UC San Diego

NSF National AI Institute TILOS

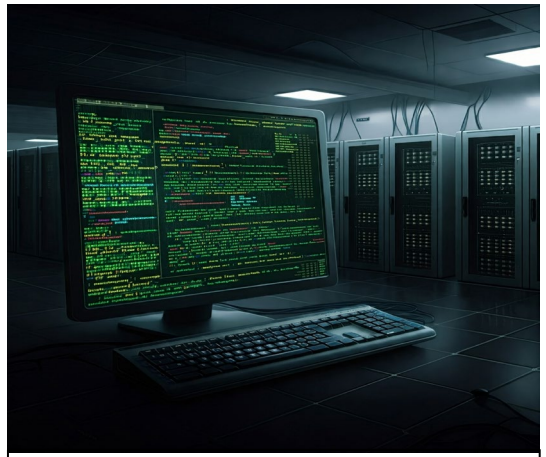
Prelude

- Algorithms, as a systematic procedure for problem solving, have a long history
  - arithmetic or geometric calculation by humans since ancient time
- Modern computations via computers: paradigm shift in algorithms

- Fusing modern AI and Data: another revolution ?



Geometric calculation,
by unknown artist,
15th century



Modern algorithms enabled by
computers,
by Gemini, Oct 2024

New Frontier: Neural Algorithmic Design

- Huge literature in classical algorithms design in TCS
  - Deep insights, elegant algorithmic frameworks, theoretical guarantees …
  - But:
    - Not all theoretically sound algorithms readily transfer to practice
    - Algorithms usually not adapted to data
- Amazing power of AI in learning from data
  - But:
    - Does a learned model generalize (especially OOD)?
    - Does a specific architecture even have the capacity to implement specific algorithms (expressivity) ?

Overarching Goal

How can we combine algorithmic ideas and insights with neural networks to develop more powerful frameworks that can learn from / adapt to data

Focus of This Talk

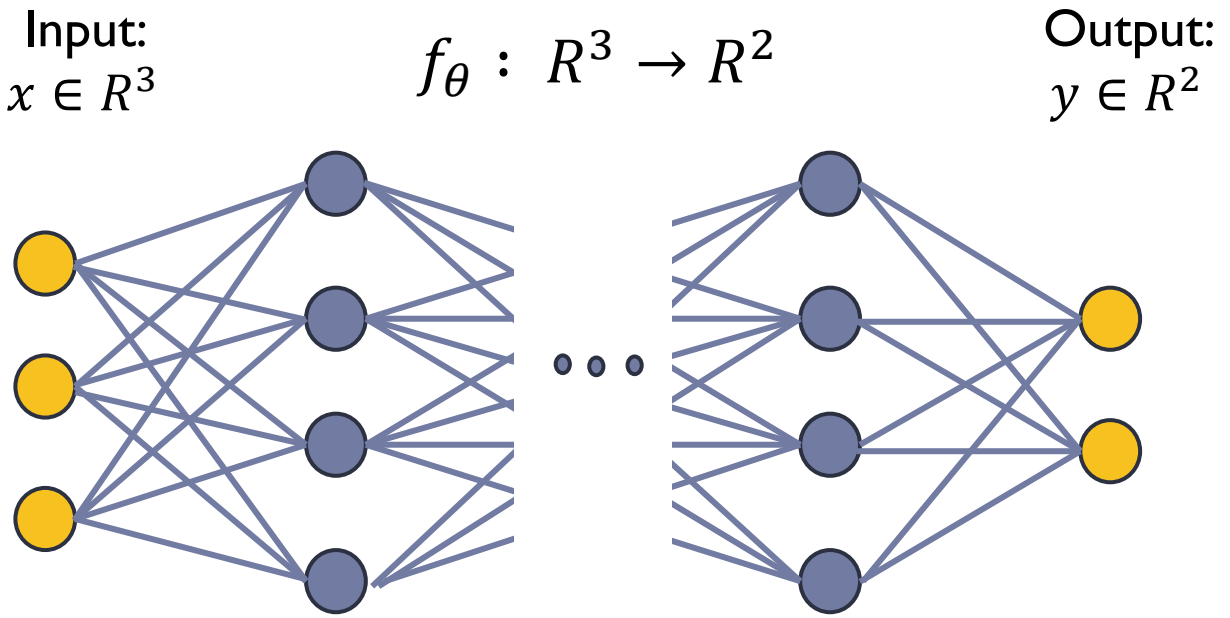
- Efficient and effective neural models to solve geometric / topological problems

- Some challenges:
  - **Expressive power**: What are the suitable neural architecture with the capacity to solve a given problem?
  - **Size (OOD) generalization**: Can a neural model with **bounded complexity** (in terms of # parameters) extrapolate / OOD generalize to input of **arbitrary (unseen) sizes**
  - **Practical performance**: how to facilitate learning ``procedures'' instead of just fitting train data?

Careful consideration of the interplay of type of data, neural architecture and task structures

Introduction

- A neural network models a function $f_\theta: X \rightarrow Y$



Introduction

- A neural network models a function $f_\theta\colon X \to Y$
- Model performance affected by the interplay of neural architecture and type of data

CNN for image data

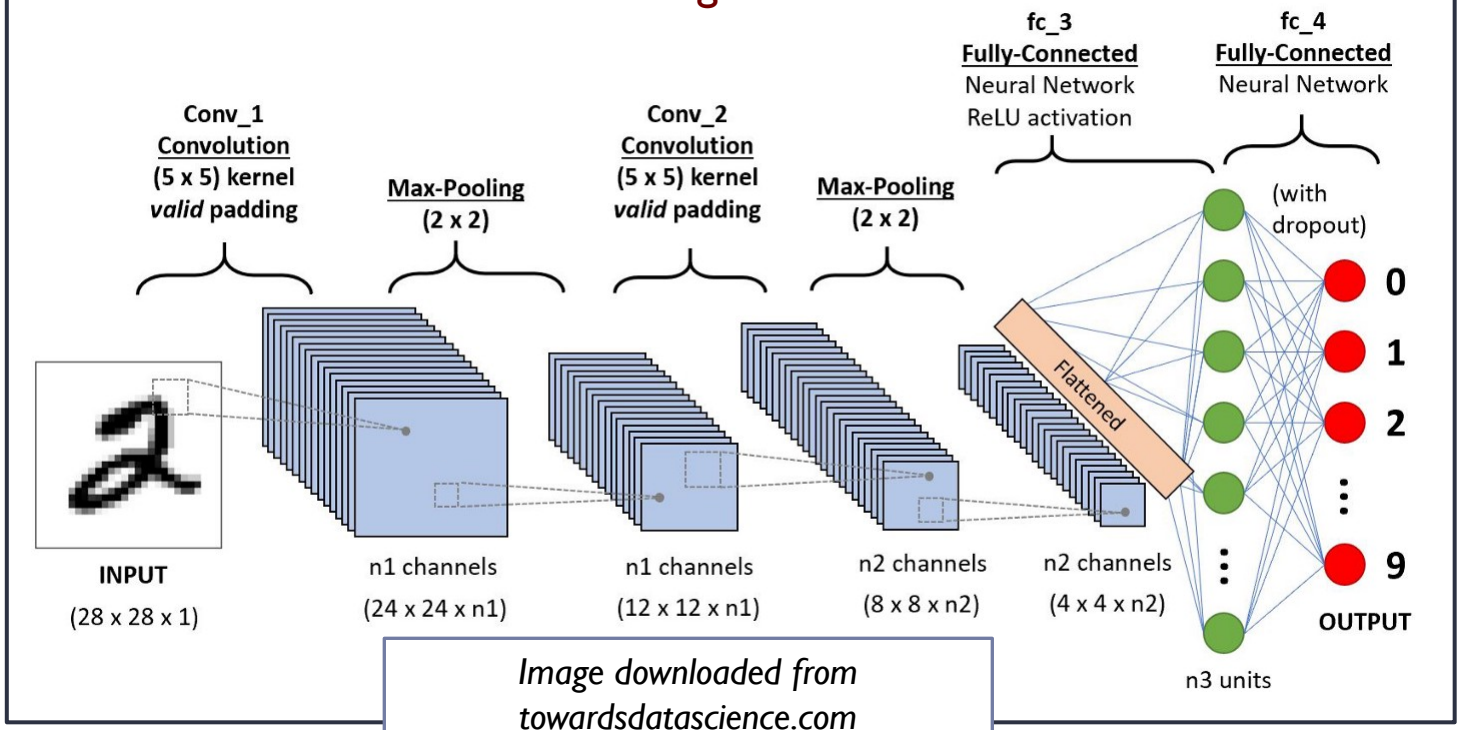
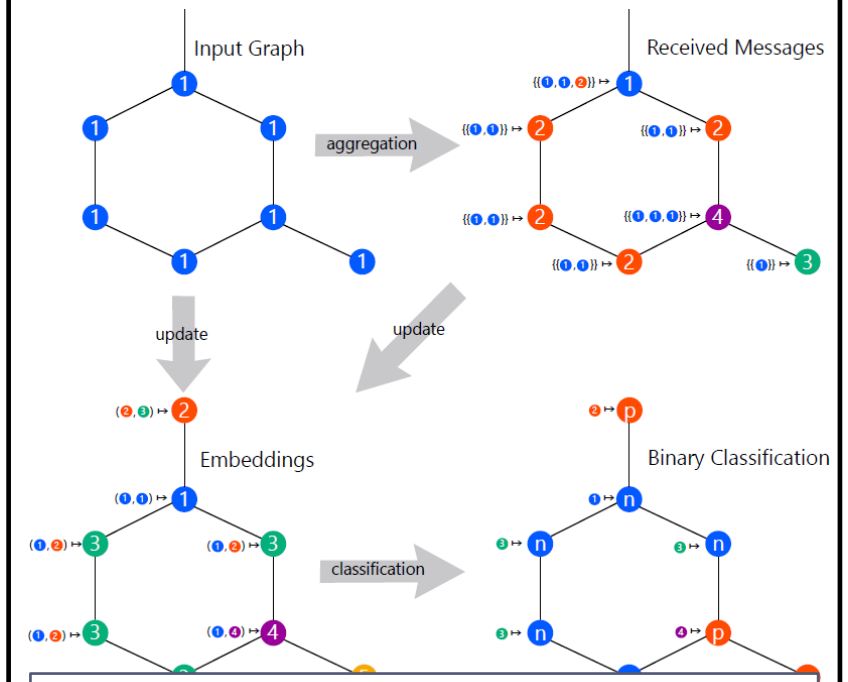


Image downloaded from towardsdatascience.com

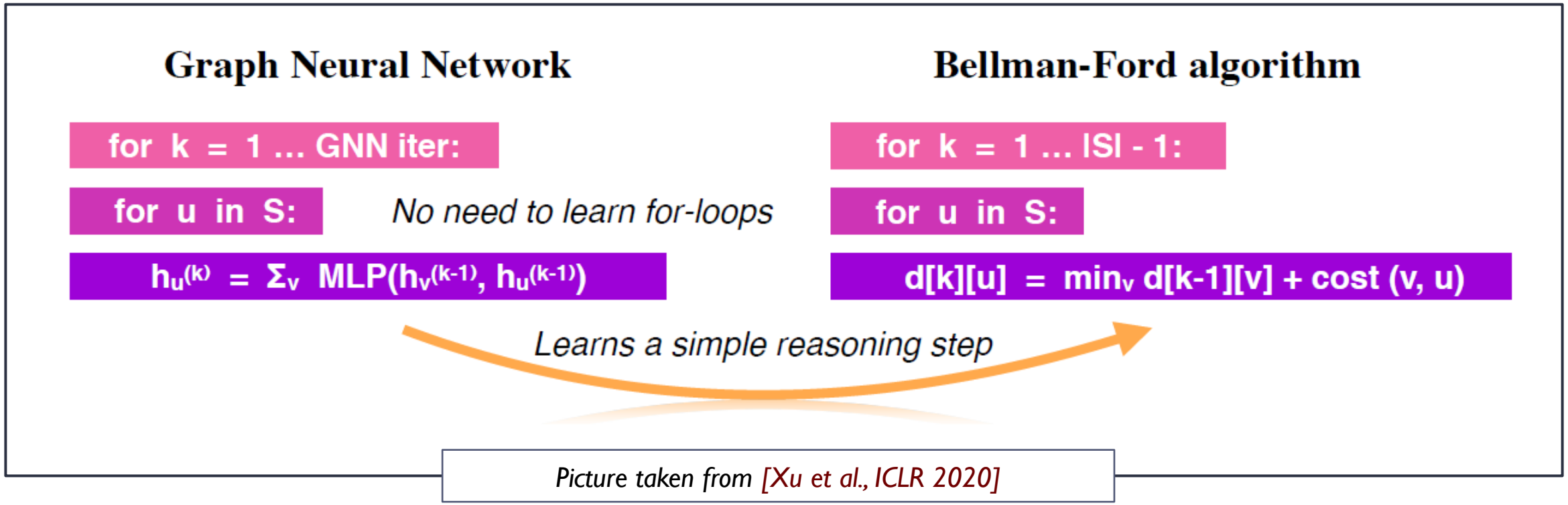
GNN for graph data



Picture taken from the survey paper by [Sato, 2020]

Introduction

- A neural network models a function $f_\theta: X \to Y$
- Model effectiveness affected by the interplay of neural architecture and type of data
- Model effectiveness also affected by the interplay of neural architecture and task structure



Picture taken from [Xu et al., ICLR 2020]

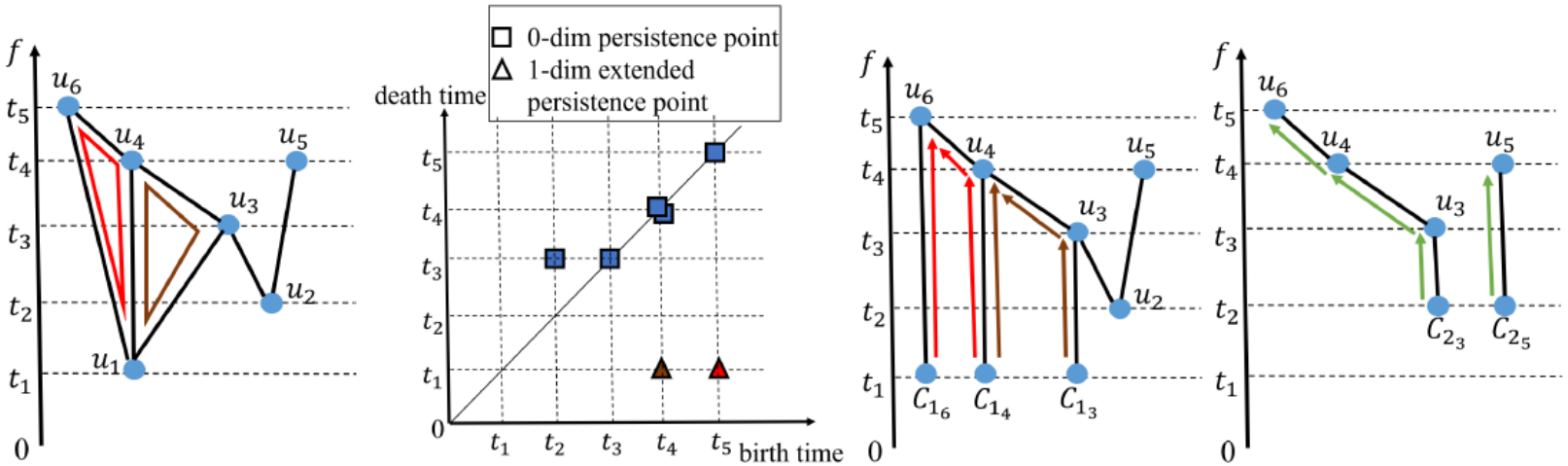
This Talk

- Main theme: "aligning" neural models with task and algorithmic structures
  - Suitable consideration of symmetries required
  - Combine the power of NNs and algorithmic ideas / scaffolds to encourage learning "procedures" instead of fitting data, facilitating effectiveness and size generalization
- Some Remarks
- Two examples of practical neural algorithmic models for geometric problems
  - Neural approximation of Wasserstein distance for Euclidean point sets
    - [Chen, **W.**, NeurIPS'23]
  - Efficient neural models to approximate a certain family of geometric optimization problems
    - [Chen, Ciolli, Sidiropoulos, **W.**, preprint 2025]

It all started in topological data analysis …

- A neural approximation of extended persistence over graphs
  - Efficient surrogate to compute persistent summaries for graph motifs
  - Differentiable: potentially allowing for optimizing for descriptor functions on graphs

- Inspired by a union-find based algorithm to compute extended persistence/levelset zigzag persistence on graphs
  - Modeled the problem as an edge-wise prediction problem over graphs
  - Developed a modified graph neural network to conceptually more aligned with the algorithm
  - [Yan, Ma, Gao, Tang, **W.**, Chen , NeurIPS 2022]

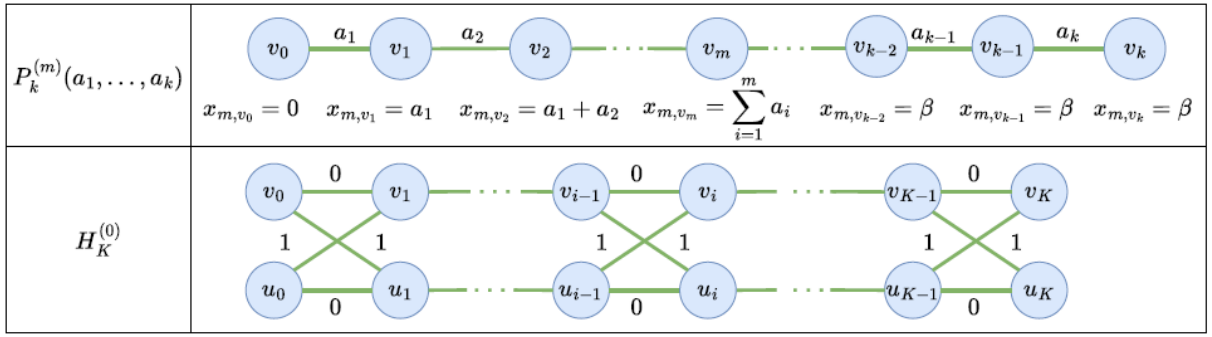


A Theoretical justification of algorithmic alignment

- Graph neural networks with low sparsity-regularized loss on small well-crafted training set provably extrapolate for shortest path (via Bellman-Ford procedure) on arbitrary graphs
  - [Nerem, Chen, Dasgupta, **W.**, arXiv 2025]
  - A first result of this kind!

Theorem [Low loss of overparameterized GNN learns K-step Bellman-Ford]

Let $\mathcal{H}_{small}$ be the specific set of only $O(K)$ graphs. Let $\mathcal{G}_{train}$ be such that $\mathcal{H}_{small} \subseteq \mathcal{G}_{train}$ with M total nodes. Assume $0 < \eta < \frac{1}{M(Lm+m+1)}$. If the regularized training loss L_{reg} of $\mathcal{A}_\theta$ is within ε of the optimal with $0 \leq \varepsilon < \eta$, and then for any positively weighted graph G (of arbitrary size), applying $\mathcal{A}_\theta$ to G approximates the K-step BF for each node with additive error $O(\varepsilon)$.



Practical Neural algorithmic models:

Example 1: Neural approximation of Wasserstein distance for point sets in Euclidean spaces

[Chen, **W.**, NeurIPS'23]

Motivation

- High-level questions:
  - Efficient and differentiable distance for complex objects (e.g, Wasserstein distance between Euclidean point sets)
  - What are the right architectures for them: efficient with theoretical guarantees / justifications, respecting suitable group operations?

- In what follows:
  - Neural approximation for Wasserstein distance between Euclidean point sets
    - Aligned with a sketching-based approximation algorithm to reduce model complexity to be constant
    - The right neural architecture to universally approximate Symmetric and factor-wise group invariant function over product space

Wasserstein Distance

- A distance to compare distributions supported on the same metric space
  - also commonly used in machine learning, e.g, in generative models

- ℓ_1-Wasserstein distance (Earth-mover distance):
  - Given $\alpha, \beta \in \mathcal{P}(Z)$ supported on a metric space (Z, d_Z), $\gamma \in \mathcal{P}(Z \times Z)$ is a coupling between α and β if $\gamma(\cdot, Z) = \alpha(\cdot)$, while $\gamma(Z, \cdot) = \beta(\cdot)$. Let $\mathcal{C}(\alpha, \beta)$ denote the set of all couplings between α and β.
  - The (ℓ_1-) Wasserstein distance between α and β is defined as

$$d_{\mathrm{W}}(\alpha, \beta) := \inf_{\gamma \in \mathcal{C}(\alpha, \beta)} \int_{Z \times Z} d_Z(z, z') \gamma(dz \times dz').$$

- In this talk we focus on distributions induced from weighted point sets in Euclidean space

Computational Complexity

- Input: two weighted point-sets of size $O(n)$
- $O(n^3 \log n)$ in general (for the more general optimal transport distance)
- Sinkhorn distance:
  - Entropic regularization speeds up computation to $O(n^2)$
  - [Cuturi 2013] , [Altschuler et al, 2017]

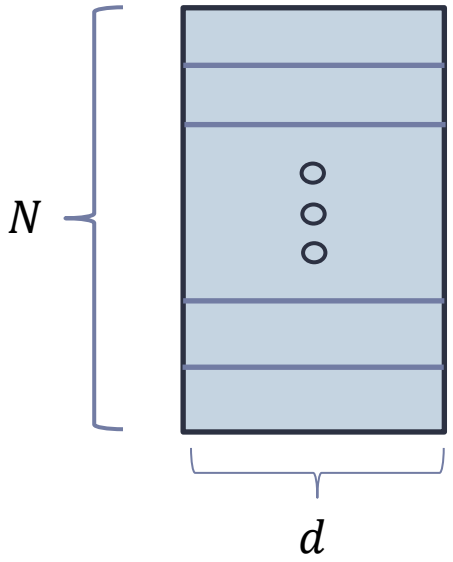
Our Goal:

Efficient bounded-size neural model to approximate Wasserstein distance for point sets of arbitrary sizes, with theoretical justification

Differentiable – can be used in ML pipelines where Wasserstein distance is used (e.g, as part of loss)

Modeling Wasserstein Distance Function

- For simplicity: input are point sets A and B in a hypercube in R^d, of cardinality at most N
  - $d_W(A, B)$ is the Wasserstein distance between the empirical measures induced by A and B
- We can represent A (and B) by a vector in $\left(R^d\right)^N = R^{Nd}$
- Wasserstein distance function:
  - $d_W : R^{Nd} \times R^{Nd} \to R$



- d_W should satisfy the following properties:
  - Symmetric: $d_W\ (A, B) = d_W(B, A)$
  - Each factor is permutation invariant on the left, that is, for any permutations $\Pi_1, \Pi_2 \in S(N)$
    - $d_W(A, B) = d_W(\Pi_1 A, \Pi_2 B)$

SFGI Product Functions

Definition (Symmetric and factor-wise invariant product function)

Let $\mathcal{X}^k = \mathcal{X} \times \cdots \times \mathcal{X}$. We define a **symmetric and factor-wise group invariant** (SFGI) product functions as $f : \mathcal{X}^k \to \mathbb{R}$ where f is symmetric and invariant to the group action $G^k = G \times G \times \cdots \times G$ and G is a group acting on $\mathcal{X}$.

- In the case of Wasserstein distance function
  - $\mathcal{X} = R^{Nd}, k = 2,\ G$ is the permutation group S_N on N elements

- More general:
  - Distance to the mean of k point sets (or other geometric objects) under Wasserstein distance
  - Other than permutations, one may also want to require rotation invariance for each factor

Group Equivariant / Invariant Networks

- Many interesting work in geometric deep learning
  - e.g., the book [Bronstein, Bruna, Cohen and Veličković, 2021]
  - DeepSet [Zaheer et al 2017], [Wagstaff et al, 2022], Invariant Graph Networks (IGN) [Maron et al 2019], [Fereydounian et al, 2022], our work [Tabaghi and **W.**, 2024] …
- Sign and basis invariant networks [Lim et al, 2023] , Stable and Expressive Position Encodings (SPE) [Huang et al, 2024]
  - Elegant ways to handle both basis and permutation invariance

First Observation

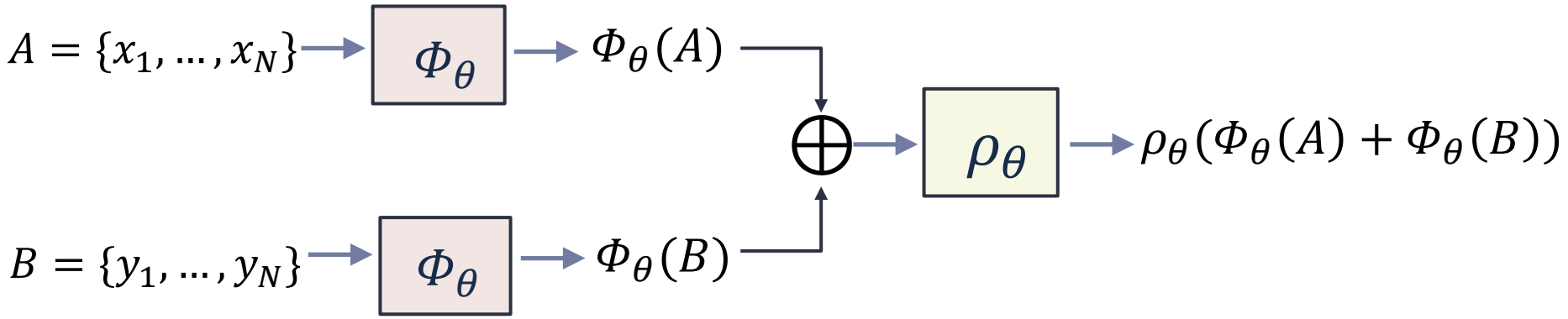
- Lemma (simplified):

  For any $\epsilon > 0$, there is a continuous permutation-invariant function $\Phi: \mathcal{X} \to R^t$ and a continuous function $\rho: R^t \to R$ such that

$$|d_W(A, B) - \rho(\Phi(A) + \Phi(B))| < \epsilon$$

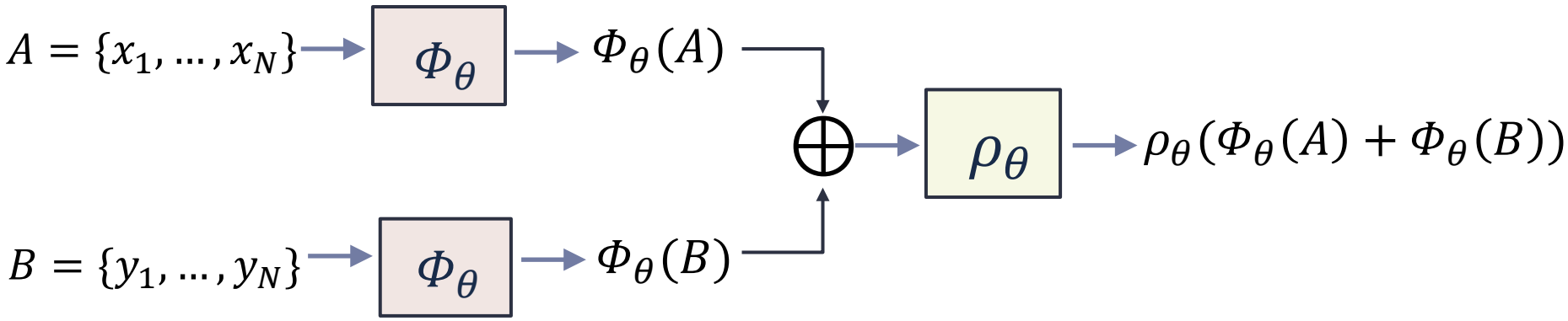
  Here, t is called the latent dimension.

Siamese network
+ MLP
to approximate
$d_W(A, B)$



However

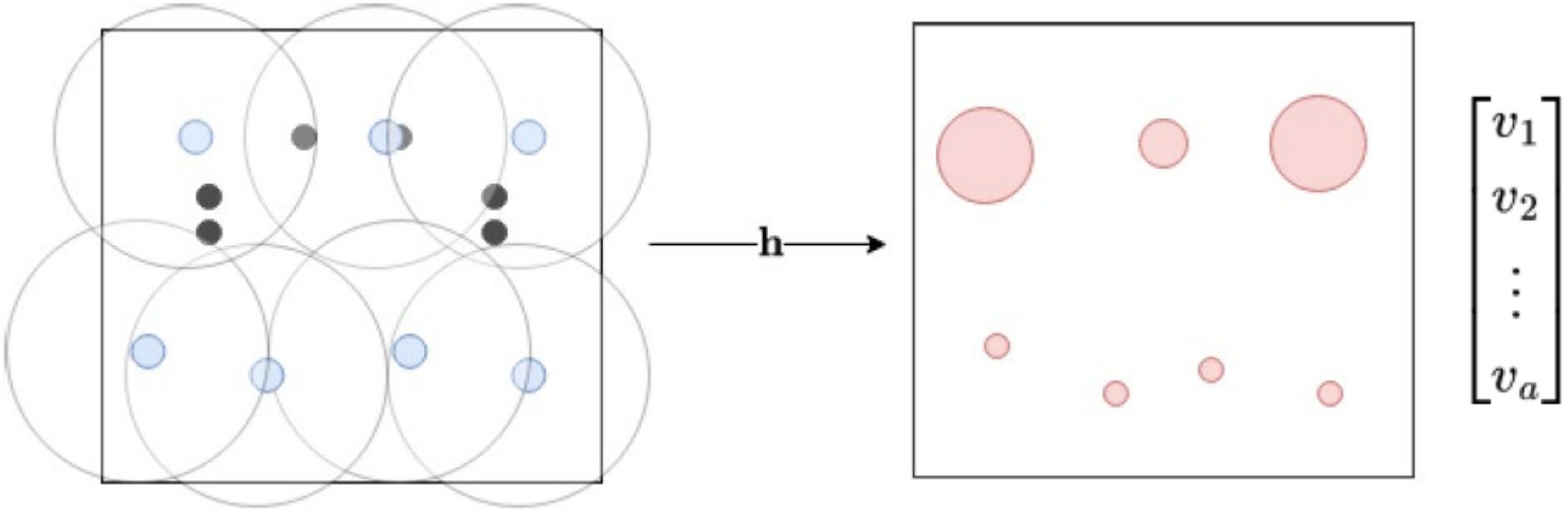
Siamese network
+ MLP
to approximate
$d_W(A, B)$



- The function $\Phi: R^{Nd} \to R^t$ need to be permutation invariant
- Φ can be implemented by DeepSet [Zaheer et al 2017]
- However, in general DeepSet requires model complexity $O(Nd)$ to achieve universality for permutation invariant functions over R^{Nd} [Tabaghi, **W.**, 2024]

Second Idea: Sketching for Approximation

- A simple approximation algorithm via ε-cover



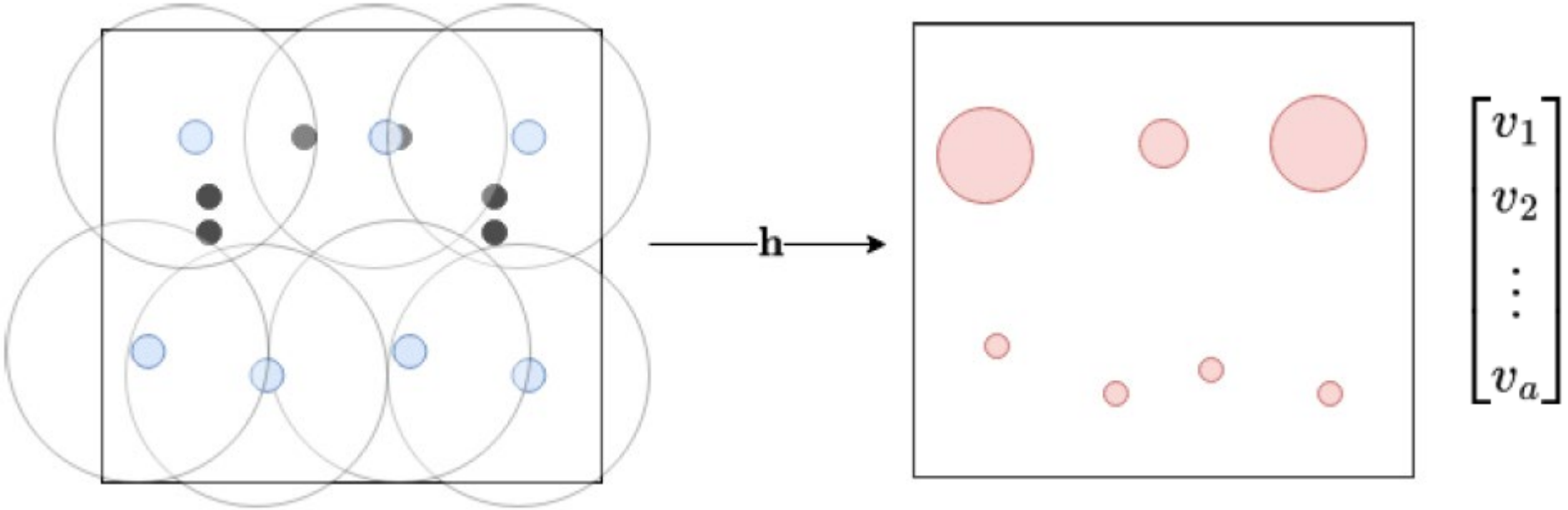
Black dots:
input point set $A \subset R^d$

A weighted point set
$A' = h(A)$;
and $|A'| = a = a(d, \varepsilon)$

such that $d_W(A', B')$ approximates $d_W(A, B)$ within additive error $O(\varepsilon)$

Second Idea: Sketching for Approximation

- A simple approximation algorithm via ε-cover

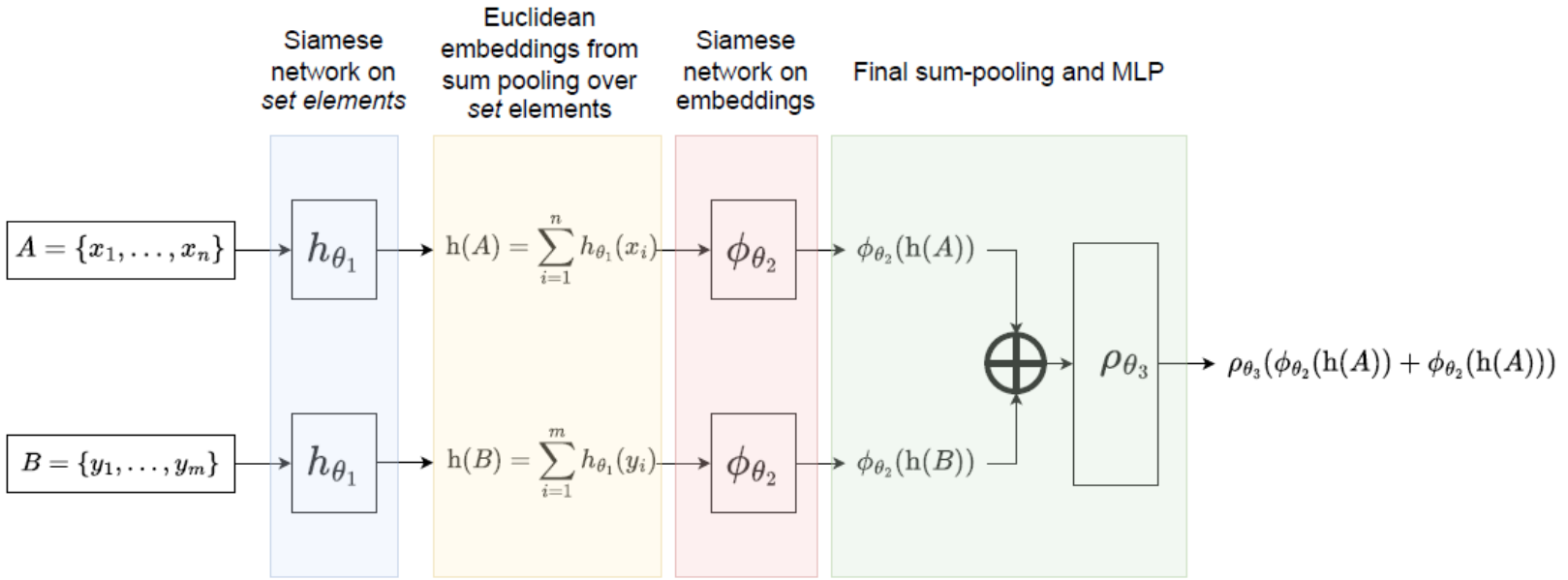


Black dots:
input point set $A \subset R^d$

A weighted point set
$A' = \boldsymbol{h}(A)$;
and $|A'| = a = a(d, \varepsilon)$

$\boldsymbol{h}$: $R^{Nd} \to R^a$ can be expressed as $\boldsymbol{h}(P) = \sum_{x \in P} h(x)$ where h: $R^d \to R^a$

Final Result



Theorem (informal):

For point sets A and B from R^d of cardinality at most N, the neural network of the form $\mathcal{N}_{prodNet}(A,B) = \rho_{\theta_3}\left(\phi_{\theta_2}\left(\sum_{x\in A} h_{\theta_1}(x)\right) + \phi_{\theta_2}\left(\sum_{x\in A} h_{\theta_1}(x)\right)\right)$ can approximate $d_W(A,B)$ within any error $\epsilon > 0$, where $h: R^d \to R^{a(\delta)}$, $\phi: R^{a(\delta)} \to R^{a'}$, and $\rho: R^{a'} \to R$ are all continuous and $a' \approx a(\delta)^2$. Furthermore, the required model complexity (number of parameters) depends only on d and ϵ.

Empirical Results

- Mean relative error between approximation and ground truth Wasserstein distances
- WPCE: [Kawano et al., 2020]
  - SOTA neural approximation of Wasserstein distance via a Siamese autoencoder framework
- $\mathcal{N}_{SDeepSets}$: just Siamese DeepSets, without final outer MLP (modeling ρ)

Dataset	Input size	$\mathcal{N}_{\text{ProductNet}}$ (Ours)	WPCE	$\mathcal{N}_{\text{SDeepSets}}$	Sinkhorn
noisy-sphere-3	[100, 300]	**0.046 ± 0.043**	0.341 ± 0.202	0.362 ± 0.241	0.187 ± 0.232
	[300, 500]	**0.158 ± 0.198**	0.356 ± 0.286	0.608 ± 0.431	0.241 ± 0.325
noisy-sphere-6	[100, 300]	**0.015 ± 0.014**	0.269 ± 0.285	0.291 ± 0.316	0.137 ± 0.122
	[300, 500]	**0.049 ± 0.054**	0.423 ± 0.408	0.508 ± 0.473	0.198 ± 0.181
uniform	256	0.097 ± 0.073	0.120 ± 0.103	0.123 ± 0.092	**0.073 ± 0.009**
	[200, 300]	0.131 ± 0.096	1.712 ± 0.869	0.917 ± 0.869	**0.064 ± 0.008**
ModelNet-small	[20, 200]	0.084 ± 0.077	**0.077 ± 0.075**	0.105 ± 0.096	0.101 ± 0.032
	[300, 500]	**0.111 ± 0.086**	0.241 ± 0.198	0.261 ± 0.245	0.193 ± 0.155
ModelNet-large	2048	**0.140 ± 0.206**	0.159 ± 0.141	0.166 ± 0.129	0.148 ± 0.048
	[1800, 2000]	**0.162 ± 0.228**	0.392 ± 0.378	0.470 ± 0.628	0.188 ± 0.088

Training Time

Table 2: Training time and number of epochs needed for convergence for best model

Dataset		$\mathcal{N}_{\text{ProductNet}}$	WPCE	$\mathcal{N}_{\text{SDeepSets}}$
noisy-sphere-3	Time	6min	1hr 46min	9min
	Epochs	20	100	100
noisy-sphere-6	Time	12min	4hr 6min	1hr 38min
	Epochs	20	100	100
uniform	Time	7min	3hr 36min	1hr 27min
	Epochs	23	100	100
ModelNet-small	Time	7min	1hr 23min	12 min
	Epochs	20	100	100
ModelNet-large	Time	8min	3hr 5min	40min
	Epochs	20	100	100

- For inference time, all the above three learning based are similar, all significantly outperform Sinkhorn approximation as number of points increase.

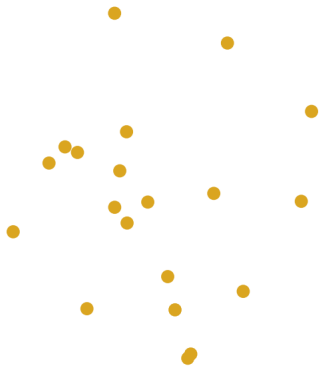
Practical neural algorithmic models:

Example 2: Efficient neural models to approximate a certain family of geometric optimization problems

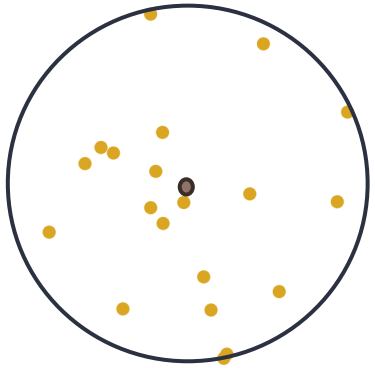
[Chen, Ciolli, Sidiropoulos, **W.**, preprint 2025]

Geometric Shape Fitting Problems

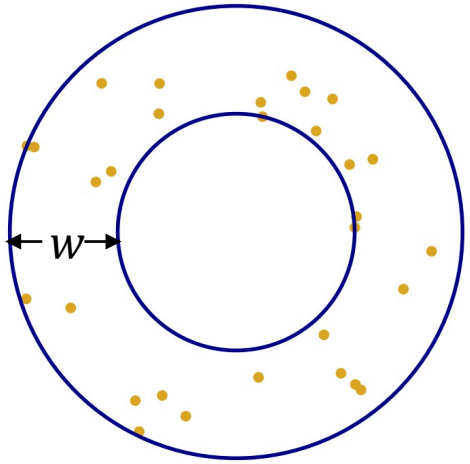
- Input:
  - A set of points $P = \{p_1, \ldots, p_n\} \subset R^d$,
  - A target shape-type (balls / ellipsoids, spherical shells, hyperplane slabs)
- Output:
  - Optimal shape covering input point set P
    - E.g, minimum enclosing ball, min-width spherical shells, min-width hyperplane slabs etc



Point set P



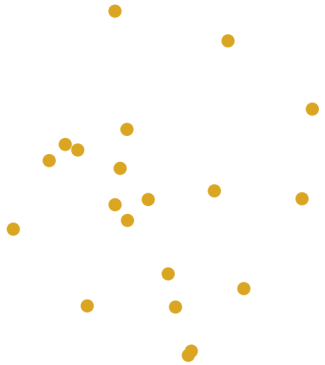
Min-enclosing ball of set P



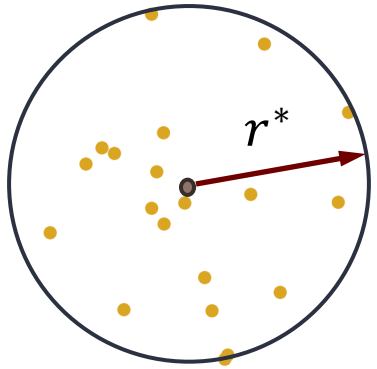
Min-width spherical shell

A Warm-up Example: Minimum Enclosing Ball (MEB)

- Input:
  - A set of points $P = \{p_1, \dots, p_n\} \subset R^d$,
- Output:
  - Radius of the minimum enclosing ball (MEB): $r^*(P) \coloneqq \min \{ r \mid \exists c \in R^d \; s.t. \; P \subseteq Ball(c; r) \}$



Point set P

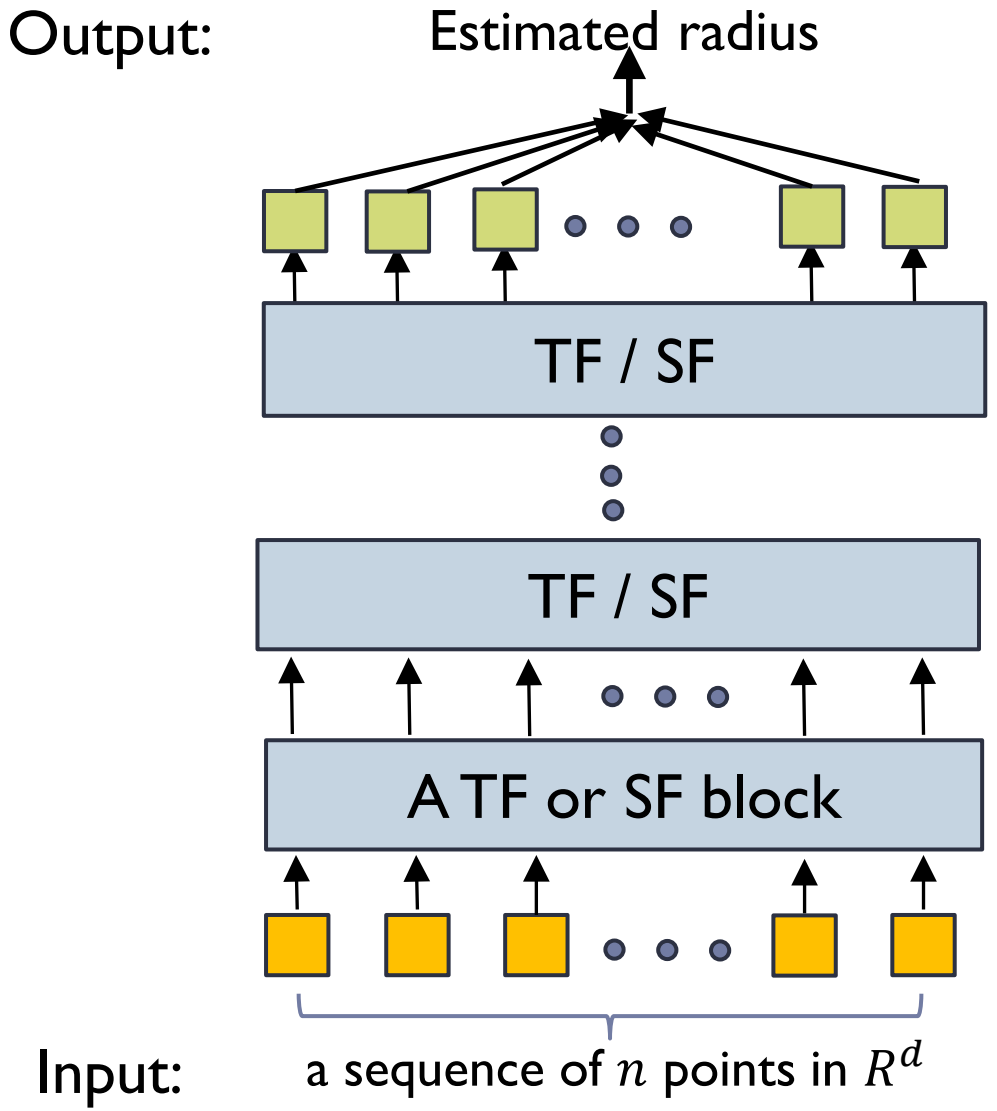


Min-enclosing ball of set P

Neural Models for MEB?

- A natural baseline for $f: R^{nd} \to R$
  - A concatenation of multiple sequence-to-sequence permutation equivariant blocks, followed by a pulling (readout)

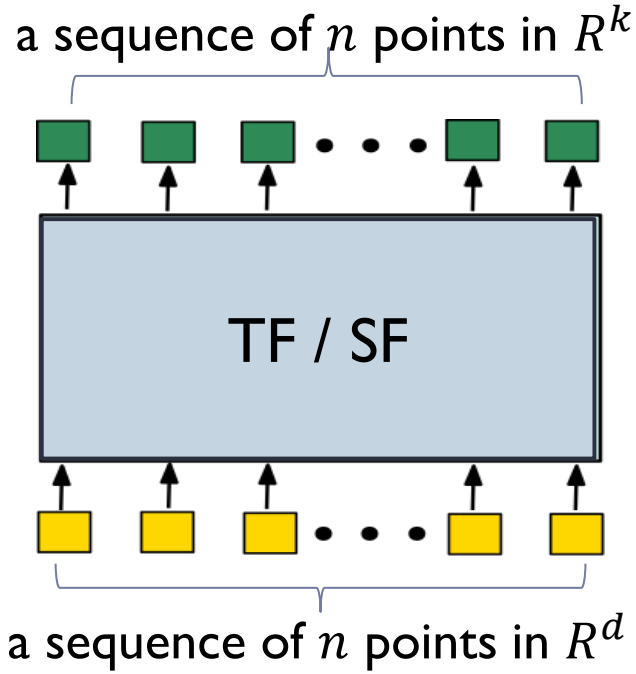
- Choice of seq-to-seq permutation equivariant blocks:
  - Transformer (TF) blocks
    - Quadratic complexity
  - SumFormer (SF) blocks
    - Linear complexity



Neural Models for MEB?

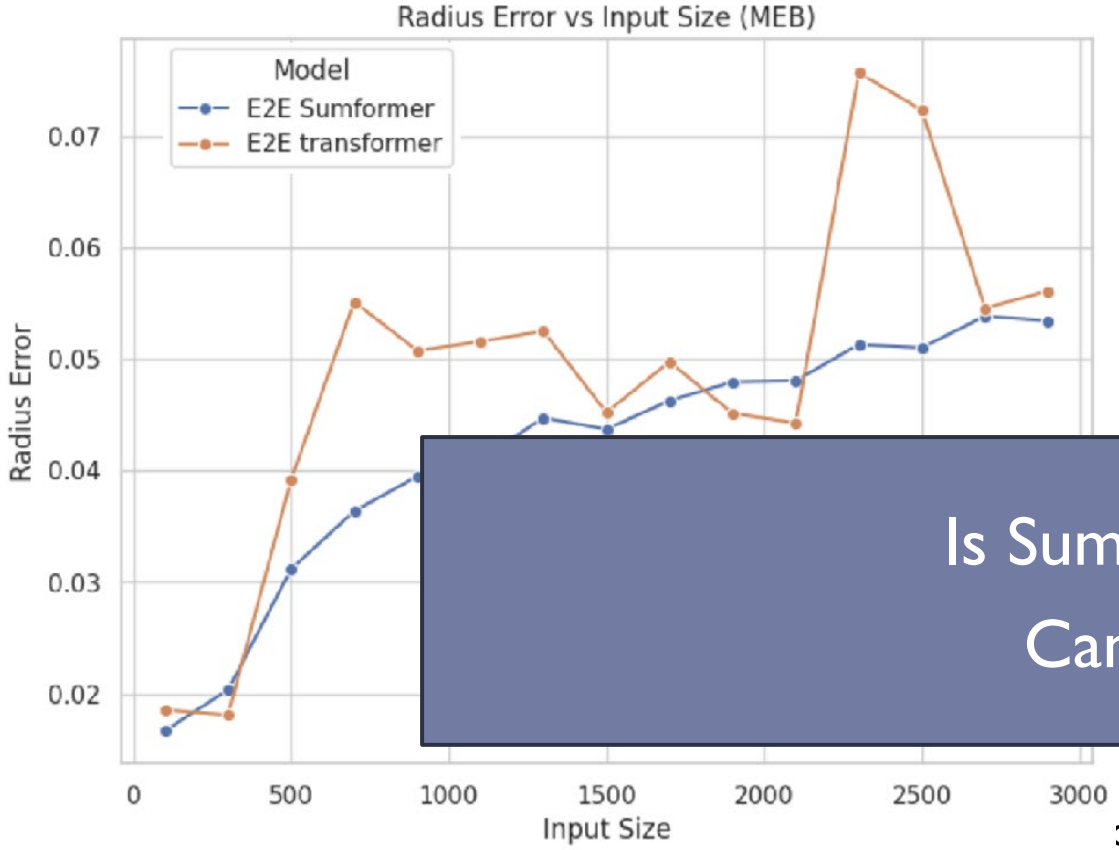
- A natural baseline for $f: R^{nd} \to R$
  - A concatenation of multiple sequence-to-sequence permutation equivariant blocks, followed by a pulling

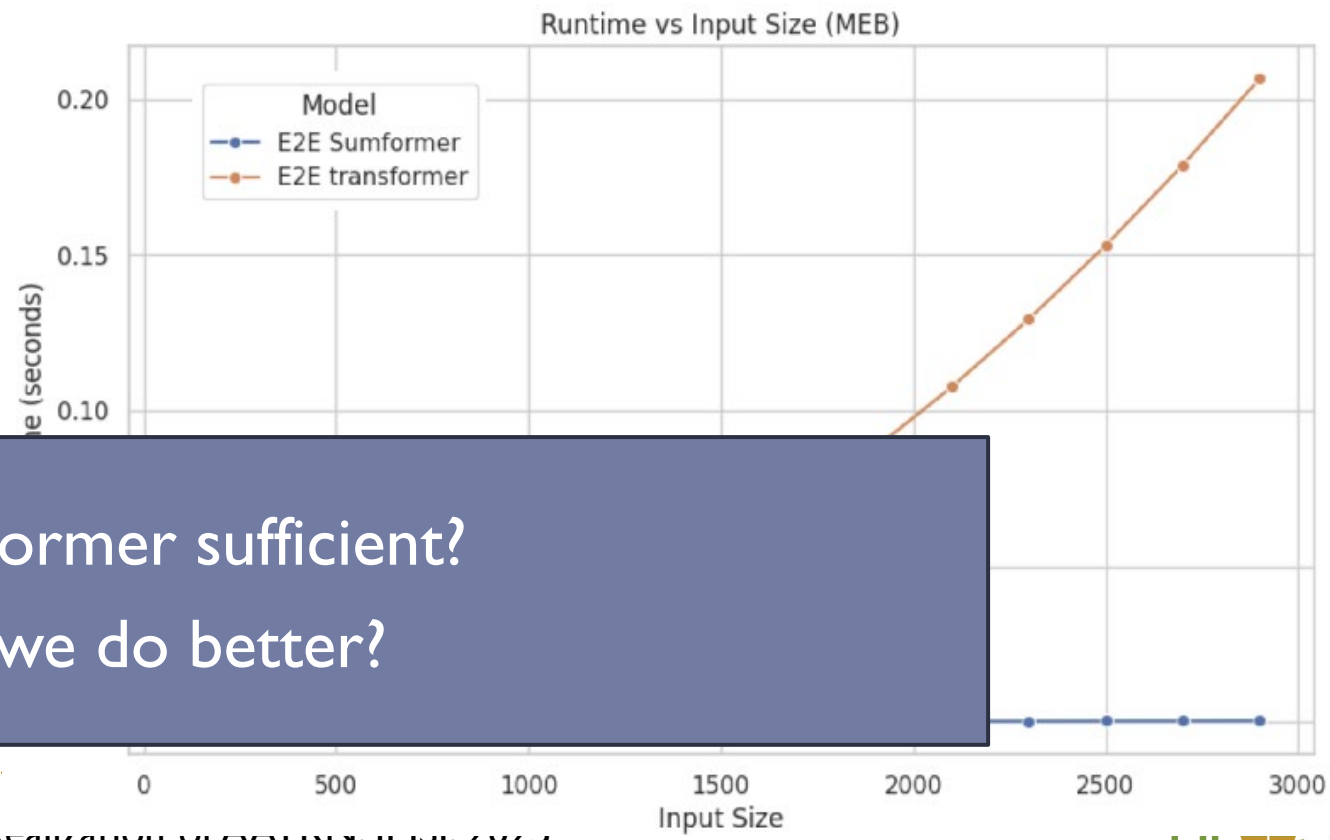
- Choice of seq-to-seq permutation equivariant blocks:
  - Transformer (TF) blocks
    - Quadratic complexity
  - SumFormer (SF) blocks
    - Linear complexity



How do these NNs perform?

- Training: point sets each with 200 points
- Testing: point sets each with 50 to 3K points
- Accuracy: Transformer baseline ≈ Sumformer baseline
- Run-time: Sumformer much faster for larger input



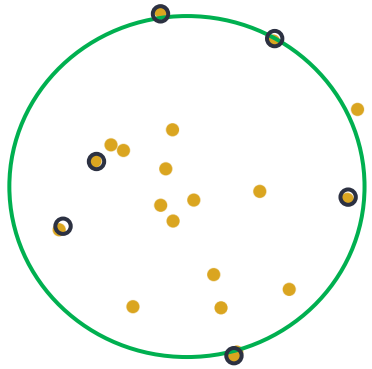


Is SumFormer sufficient?

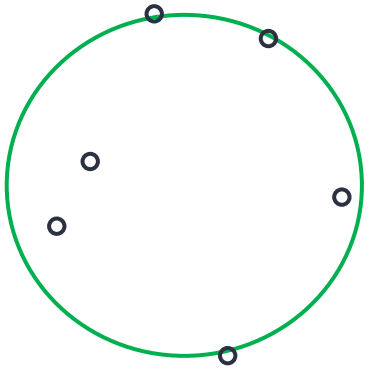
Can we do better?

A Simple Approximation Algorithm for MEB

- Input:
  - A set of points $P = \{p_1, \dots, p_n\} \subset R^d$
- MEB approximation algorithm:
  - Compute $Q \subseteq P$: a ε-kernel of P
    - s.t. $|Q| \ll |P|$
  - Construct a MEB Ball$(c^*(Q); r^*(Q))$ of Q
  - Return $r^*(Q)$, which is an ε-approximation of $r^*(P)$



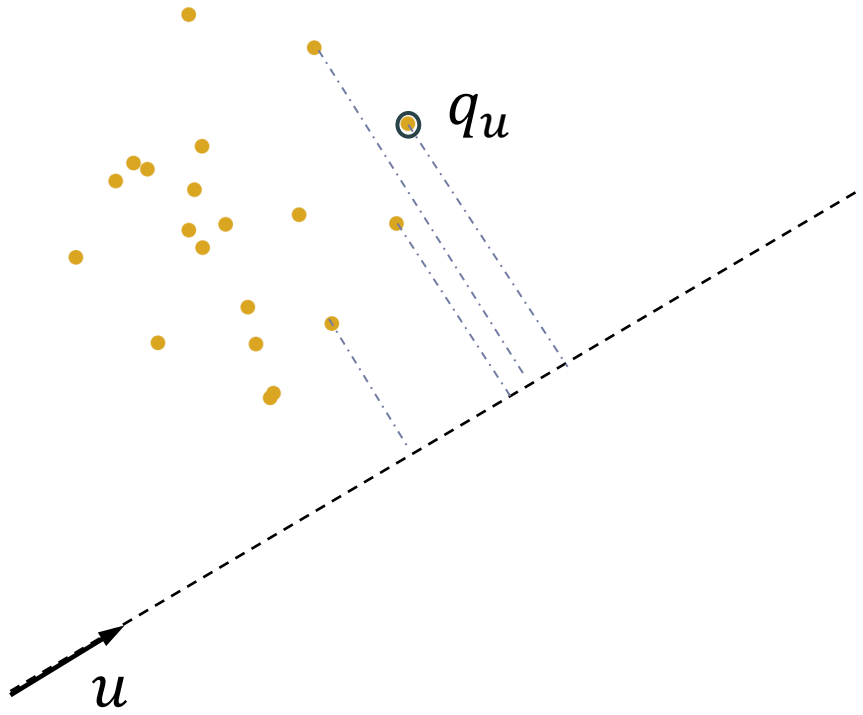
Point set P



ε-kernel Q

Constructing ε-kernel of small sizes

- A simple algorithm
  - Let U be a δ-net of S^{d-1}, for suitable $\delta = O(\varepsilon)$
    - $|U| = O(\frac{1}{\varepsilon^{d-1}})$
  - Initialize $Q = \emptyset$
  - For $u \in U$ do
    - Add point $q_u = \ argmax_{p \in P} \langle p, u \rangle$ to Q
  - Return Q
    - note $|Q| = |U| = O(\frac{1}{\varepsilon^{d-1}})$, which is independent to the size of P



This is not the most efficient algorithm, and big-O hides constant depending on P as well.

Relaxed ε-kernel

- A neural implementation of ε-kernel approximation algorithm ?
- argmax is hard to implement
  - softmax is usually used as a differentiable replacement of argmax
- However, the resulting Q computed via softmax is no longer a ε-kernel

- A relaxed ε-kernel for P is a set of points $Q \subseteq \text{ConvexHull}(P)$ s.t.
  - for any $u \in S^{d-1}$, $w_u(Q)$ ε-approximates $w_u(P)$; i.e.,

$$(1-\varepsilon)w_u(Q) \leq w_u(P) \leq (1+\varepsilon)\, w_u(Q)$$

We show that the relaxed ε-kernel leads to similar theoretical guarantees as the classical ε-kernel theory, both for approximating faithful extent measures, and in the combination of linearization strategy for approximating a broader family of extent measures

Computing Relaxed ε-kernels

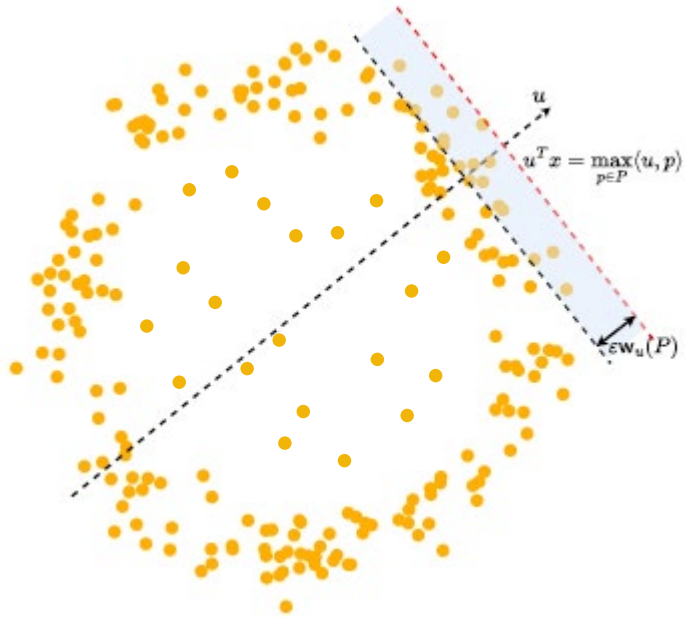
Algorithm 1 Relaxed-ε-kernels

Require: $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d$, k

$\Omega \leftarrow$ set of k directions from $\mathbf{S}^{d-1}$
$Q \leftarrow \{\}$
for $u \in \Omega$ **do**
  $A_u \leftarrow \{\langle u, p\rangle : p \in P\}$
  $w(A_u) \leftarrow \max A_u - \min A_u$
  Compute $\rho_\varepsilon(A_u)$
  $q_u \leftarrow \left(\frac{\rho_\varepsilon(A_u)}{\|\rho_\varepsilon(A_u)\|_1}\right)^T P$
  Q.append(q_u)
end for
return Q

Set $M_{A_u} := \max(A_u) = \max_{p \in P} \langle u, p\rangle$

$$\rho_\varepsilon(A_u) = \begin{pmatrix} \mathrm{ReLU}(\langle u, p_1\rangle + \varepsilon w(A_u) - M_{A_u}) \\ \mathrm{ReLU}(\langle u, p_2\rangle + \varepsilon w(A_u) - M_{A_u}) \\ \vdots \\ \mathrm{ReLU}(\langle u, p_n\rangle + \varepsilon w(A_u) - M_{A_u}) \end{pmatrix}$$



- Relaxed ε-kernels of small size (independent to $n = |P|$ can still be computed easily!

Neural Module for Relaxed ε-kernel Approximation

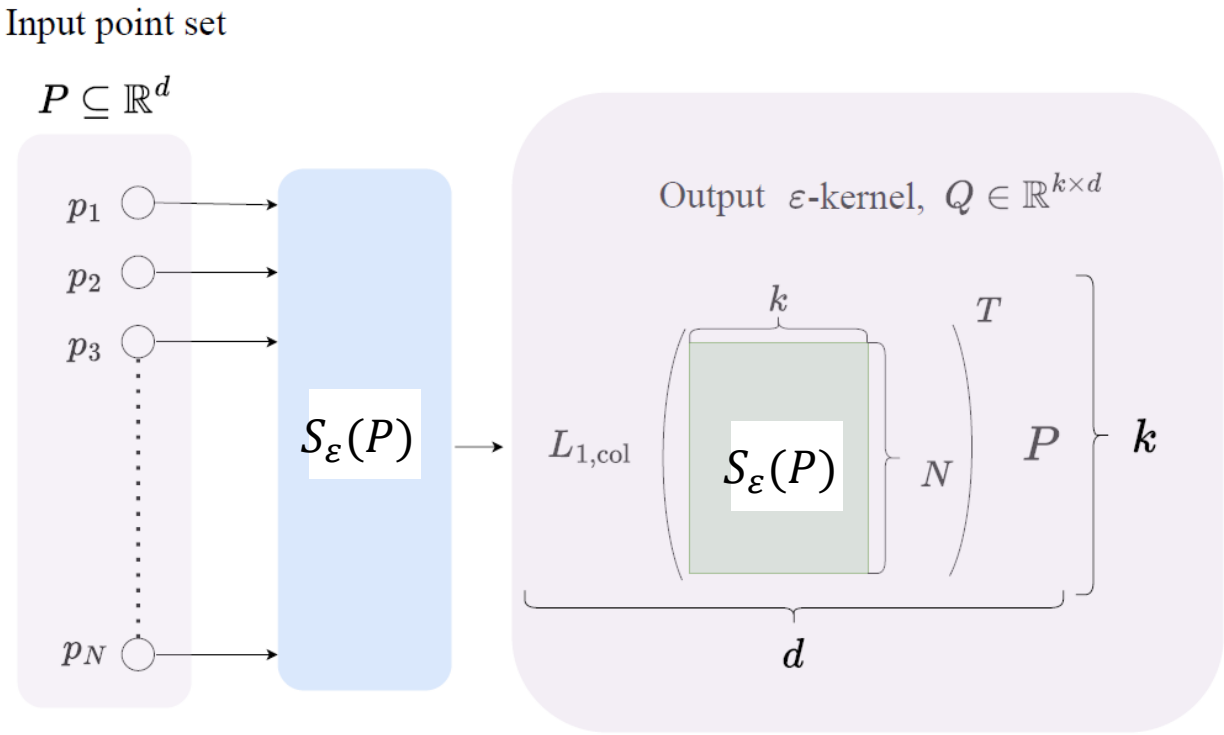
- The following neural model is aligned with the relaxed ε-kernel apprx. Alg.

$$\mathcal{N}_S(P) = L_{1,col}\big(S_\varepsilon(P)\big)^T P$$

where S_ε is a SumFormer block, and $L_{1,col}$ represents column-wise L_1-normalization.

- Intuitively, $\mathcal{N}_S: R^{nd} \to R^{kd}$ maps n input points in $P \subset R^d$ to k points in a relaxed ε-kernel $Q \subset R^d$

ε-kernel approx. neural network



Neural Module for Relaxed ε-kernel Approximation

- The following neural model is aligned with the relaxed ε-kernel apprx. Alg.

$$\mathcal{N}_S(P) = L_{1,col}\big(S_\varepsilon(P)\big)^T P$$

where S_ε is a SumFormer block, and $L_{1,col}$ represents column-wise L_1-normalization.

  - Intuitively, $\mathcal{N}_S: R^{nd} \to R^{kd}$ maps n input points in $P \subset R^d$ to k points in a relaxed ε-kernel $Q \subset R^d$

Informal Theorem [ε-kernel approx. NN]

Given any $\varepsilon, \alpha > 0$, there exists a SumFormer S_ε with only a single block, and bounded model complexity (depending only on ε, α and d), such that, for any α-fat point set $P \subset R^d$ of arbitrary size, $\mathcal{N}_\varepsilon(P)$ is a relaxed ε-kernel for P.

A SumFormer-based NN with a MLP at the end can approximate the target faithful extent measure.

Experiments I: ε-kernel approx. NN

- Effectiveness of our SumFormer based ε-kernel approx. NN $\mathcal{S}_\varepsilon$
  - but with column-wise softmax normalization instead of L_1 normalization
    - $\mathcal{S}_\varepsilon(P) = Softmax\big(S_\varepsilon(P)\big)^T P$
- Comparing with $\mathcal{T}_\varepsilon$, where the SumFormer is replaced by Transformer

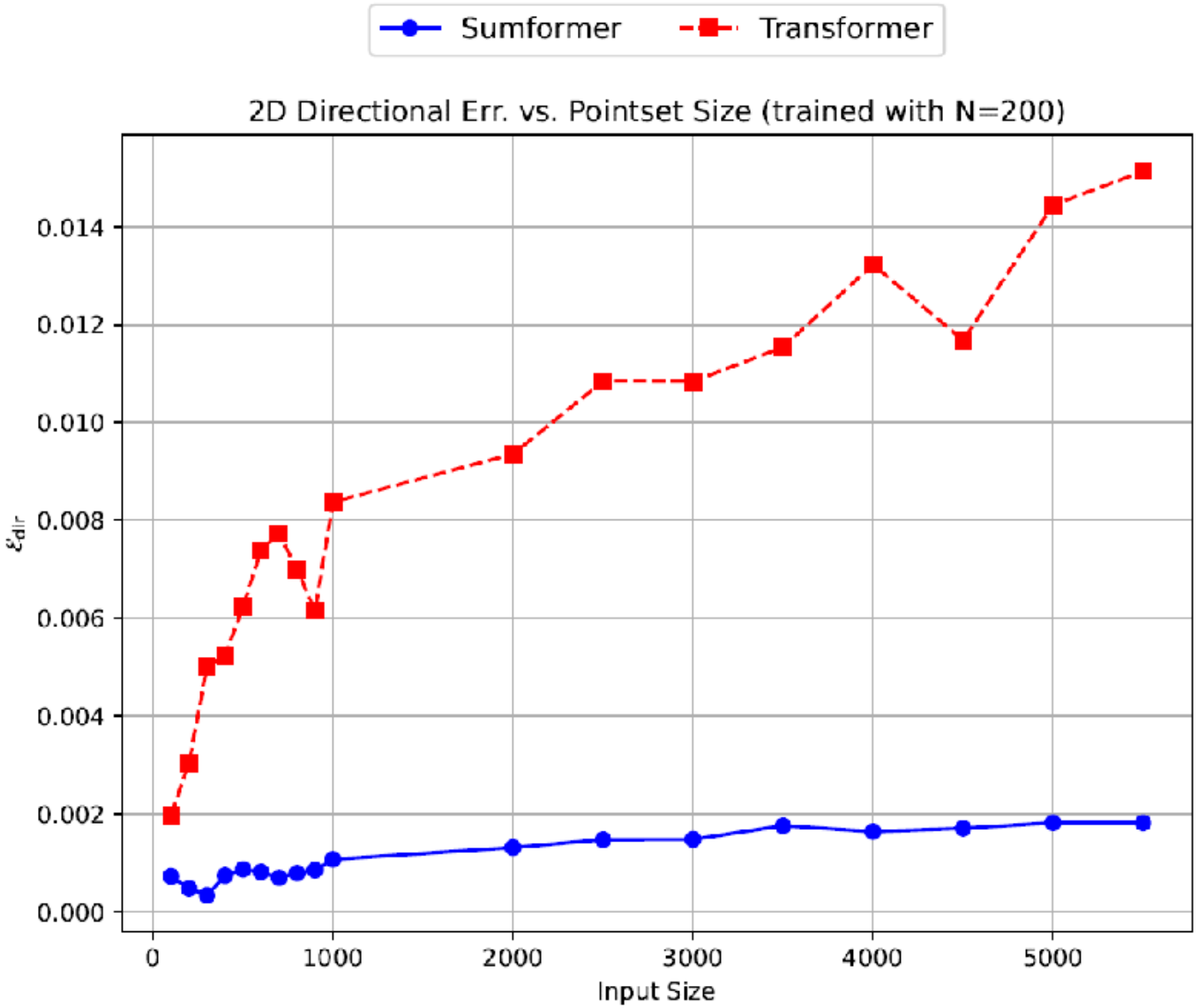
- Point sets in R^3
- Loss is avg directional width error for 1000 directions

- SumFormer based model has much better OOD generalization than Transformer

		Test Sets		
			OOD	
Train Set	**Method**	In-dist.	Synthetic	ModelNet
Ellipse	$\mathcal{S}_\varepsilon$	**0.049**	**0.032**	**0.066**
	$\mathcal{T}_\varepsilon$	0.055	0.106	0.147
Gaussian Mixture	$\mathcal{S}_\varepsilon$	**0.047**	**0.039**	**0.064**
	$\mathcal{T}_\varepsilon$	0.054	0.066	0.250
ModelNet	$\mathcal{S}_\varepsilon$	**0.041**	**0.099**	–
	$\mathcal{T}_\varepsilon$	0.049	0.265	–
Mixed Synthetic	$\mathcal{S}_\varepsilon$	**0.035**	–	**0.055**
	$\mathcal{T}_\varepsilon$	0.042	–	0.117

Experiment II: Size generalization

- Model trained on a collection of point sets in 2D, where the size of each point set is only 200
  - SumFormer based model and transformer based model are trained by same number of epochs
- Tested on point sets of cardinality up to 5000
- SumFormer based model (blue curve) generalizes much better
  - Similar results in higher dimensions



But we are not done yet !

Optimizing for More General Extent Meaures

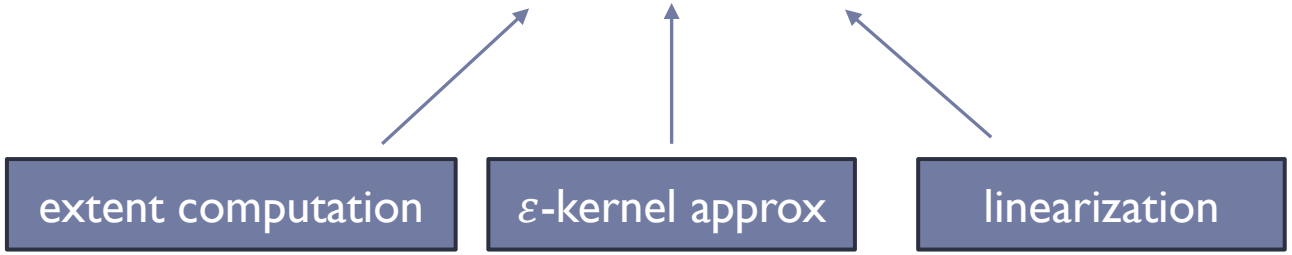
- Previous approach only works for approximating faithful measures
- A more general approximation for extent measures via Linearization technique
  - [Agarwal et al., 2003]

- Goal: approximate a certain extent measure $\mu(P)$ for any input point set $P \subset R^d$
  - Linearization:
    - Lift the set of points $P = \{p_1, \dots, p_n\} \subset R^d$ to a set of linear functions $F = \{f_1, \dots, f_n\}$ over R^m such that the target $\mu(P)$ is the same as $\min_{x \in R^m} \mathcal{E}_F(x)$, the minimum extent of F.
  - Approximate extent in dual space
    - Dualize F to a set of points $F^* \subset R^{m+1}$
    - Approximate extent $\mathcal{E}(G^*)$ from an ε-kernel G^* of F^*
  - Return the approximate extent $\mathcal{E}(G^*)$ as an approximation of $\mu(P)$

Neural Framework for Multiple Extents Approximation

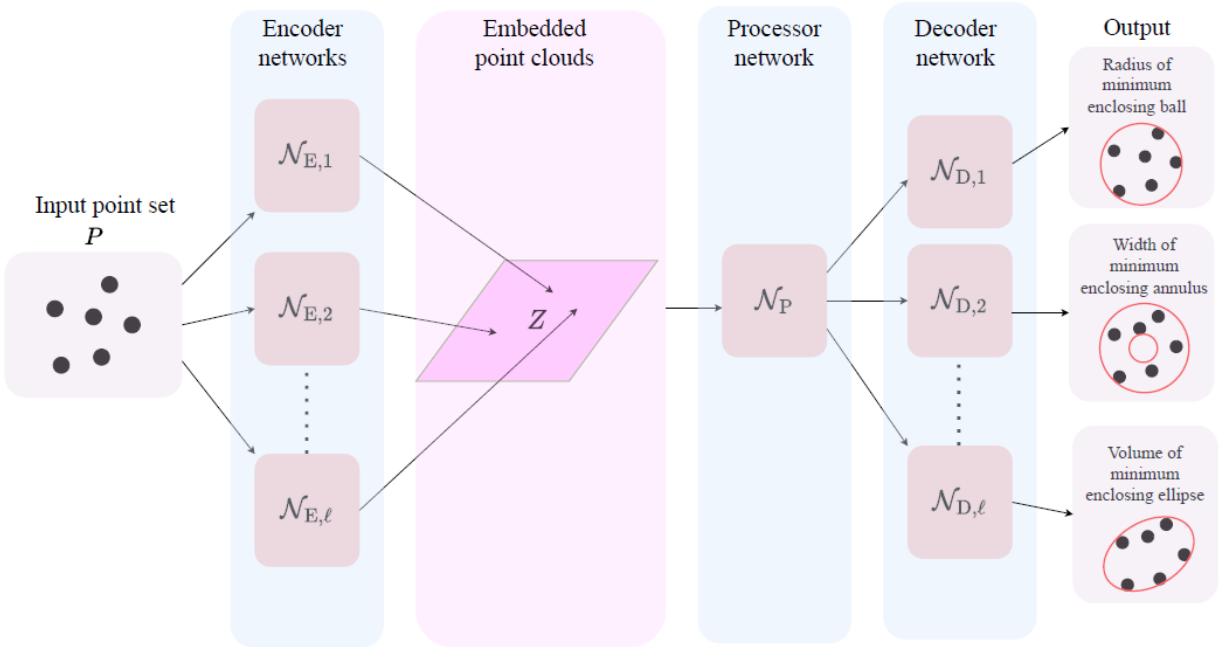
- An Encoder-Processor-Decoder framework that can approximate multiple extent measures where the linearization technique applies.
  - The encoder-processor-decoder is inspired by [Veličković and Blundell, 2021]

$$\mathcal{N}_{extent}(P) = \mathcal{N}_D \circ \mathcal{N}_P \circ \mathcal{N}_E(P)$$



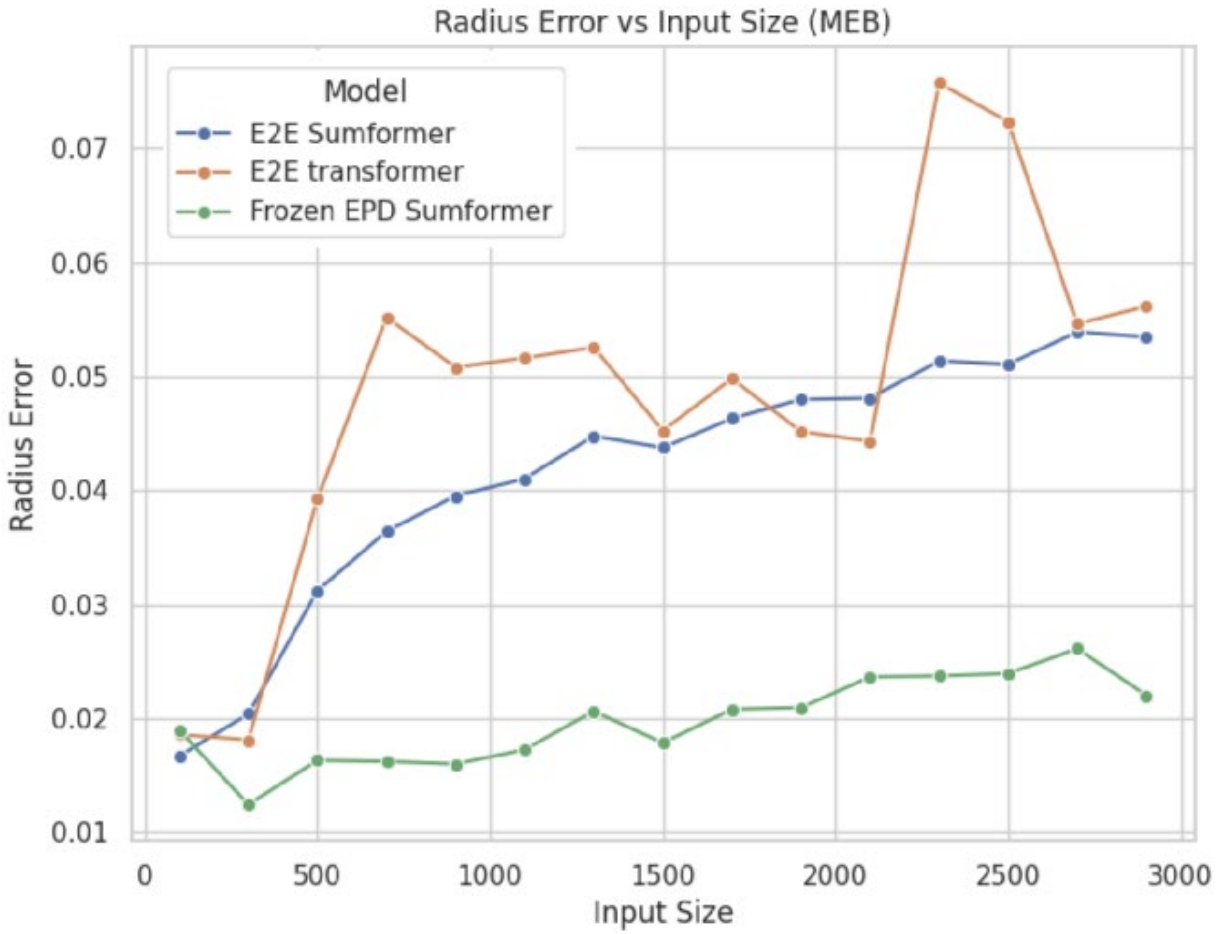
Neural Framework for Multiple Extent Approximation

- An Encoder-Process-Decoder framework that can approximate multiple extent measures where the linearization technique applies.
  - $\mathcal{N}_{extent}(P) = \mathcal{N}_D \circ \mathcal{N}_P \circ \mathcal{N}_E(P)$
    - The processor network $\mathcal{N}_P$ is our ε-kernel approx. NN as introduced earlier
    - Task-specific encoder is implemented by MLP applied pointwise
    - Task-specific decoder is implemented by DeepSet



Experiment III: Neural models for MEB

- Comparing previous E2E Transformer and E2E SumFormer with new neural model (Frozen EPD Sumformer)
  - Much better size generalization
- Comparing with SoTA approximation algorithm:
  - Eps-kernel + Welzl's randomized incremental algorithm [Welzl 1991]
  - Our approach 18x approx. error, but 15x faster
- However, for the more challenging min-width annulus problem
  - Our approach is 19x faster, with similar error



Experiment IV: More Comparison with Neural Approaches

- Performance for MEB (minimum enclosing balls), MEE (minimum enclosing ellipsoid), MEA (minimum enclosing annulus)

- Point sets in R^2
- $\mathcal{S}_{extent}$ (Frozen)
  - proposed encoder-processor-decoder framework, with latent dimension = 5
- $\mathcal{S}_{extent}$ (E2E)
  - encoder-processor-decoder framework, but the processor network N_P is trained end-to-end
- $\mathcal{S}_{Baseline}$ (E2E)
  - simply use SumFormer blocks E2E
- $\mathcal{T}_*$: Transformer-based analogs

		MEB	MEE		MEA
Train Set	**Method**	$\mathcal{E}_r$	$\mathcal{E}_{r,\min}$	$\mathcal{E}_{r,\text{maj}}$	$\mathcal{E}_w$
Synthetic	$\mathcal{S}_{\text{extent}}$ (Frozen)	**0.020**	**0.037**	**0.022**	0.028
	$\mathcal{S}_{\text{extent}}$ (E2E)	0.023	0.056	0.047	**0.022**
	$\mathcal{T}_{\text{extent}}$ (Frozen)	0.099	0.041	0.027	0.454
	$\mathcal{T}_{\text{extent}}$ (E2E)	0.024	0.038	0.022	0.026
	$\mathcal{S}_{\text{Baseline}}$ (E2E)	0.048	0.378	0.426	1.010
	$\mathcal{T}_{\text{Baseline}}$ (E2E)	0.030	0.071	0.047	0.066
SQUID	$\mathcal{S}_{\text{extent}}$ (Frozen)	0.020	**0.056**	0.047	**0.076**
	$\mathcal{S}_{\text{extent}}$ (E2E)	**0.010**	0.078	0.050	0.145
	$\mathcal{T}_{\text{extent}}$ (Frozen)	0.068	0.058	**0.032**	0.096
	$\mathcal{T}_{\text{extent}}$ (E2E)	0.190	0.093	0.045	0.099
	$\mathcal{S}_{\text{Baseline}}$ (E2E)	0.027	0.322	0.250	0.084
	$\mathcal{T}_{\text{Baseline}}$ (E2E)	0.039	0.357	0.049	0.152

Experiment IV: More Comparison with Neural Approaches

- Performance for MEB (minimum enclosing balls), MEE (minimum enclosing ellipsoid), MEA (minimum enclosing annulus)

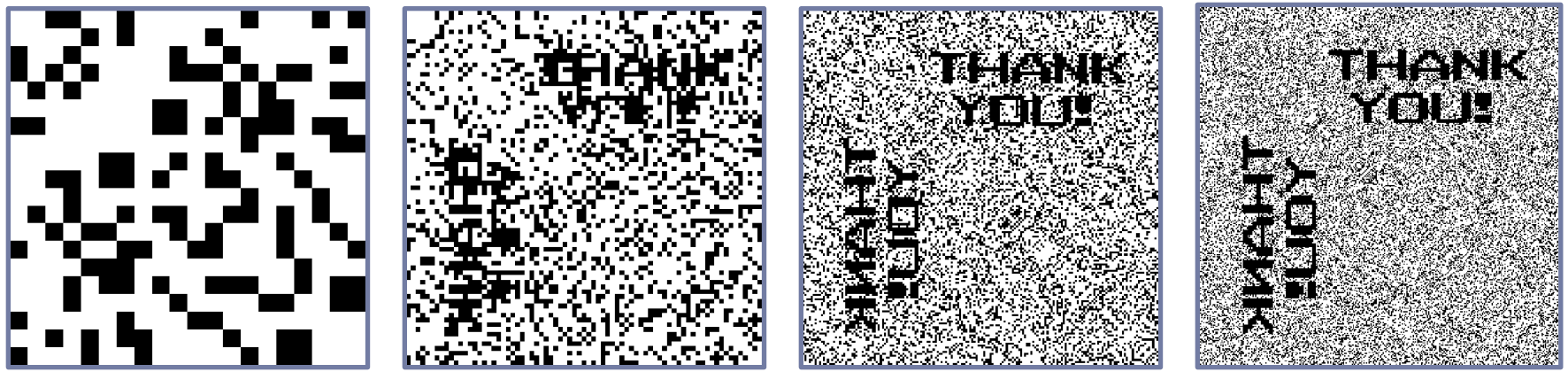
- $\mathcal{S}_{extent}$ (Frozen) usually the best
- $\mathcal{S}_{extent}$(*) usually outperforms analogous $\mathcal{T}_{extent}$(*)
- However, $\mathcal{T}_{baseline}$(E2E) outperforms $\mathcal{S}_{baseline}$(E2E) !
- Intuitively, without alignment, SumFormer sometimes is much worse than Transformer, while with alignment it is better

		MEB	MEE		MEA
Train Set	**Method**	$\mathcal{E}_r$	$\mathcal{E}_{r,\min}$	$\mathcal{E}_{r,\mathrm{maj}}$	$\mathcal{E}_w$
Synthetic	$\mathcal{S}_{\mathrm{extent}}$ (Frozen)	**0.020**	**0.037**	**0.022**	0.028
	$\mathcal{S}_{\mathrm{extent}}$ (E2E)	0.023	0.056	0.047	**0.022**
	$\mathcal{T}_{\mathrm{extent}}$ (Frozen)	0.099	0.041	0.027	0.454
	$\mathcal{T}_{\mathrm{extent}}$ (E2E)	0.024	0.038	0.022	0.026
	$\mathcal{S}_{\mathrm{Baseline}}$ (E2E)	0.048	0.378	0.426	1.010
	$\mathcal{T}_{\mathrm{Baseline}}$ (E2E)	0.030	0.071	0.047	0.066
SQUID	$\mathcal{S}_{\mathrm{extent}}$ (Frozen)	0.020	**0.056**	0.047	**0.076**
	$\mathcal{S}_{\mathrm{extent}}$ (E2E)	**0.010**	0.078	0.050	0.145
	$\mathcal{T}_{\mathrm{extent}}$ (Frozen)	0.068	0.058	**0.032**	0.096
	$\mathcal{T}_{\mathrm{extent}}$ (E2E)	0.190	0.093	0.045	0.099
	$\mathcal{S}_{\mathrm{Baseline}}$ (E2E)	0.027	0.322	0.250	0.084
	$\mathcal{T}_{\mathrm{Baseline}}$ (E2E)	0.039	0.357	0.049	0.152

Concluding Remarks

- Model performance relies on interplay of data type, model architecture, task structure and also optimization (model training)
- Size (OOD) generalization important in handling geometric and topological problems
- Using algorithmic insights combined with ML to develop more principled neural algorithms
  - E.g, using algorithmic scaffolding to design neural models

- General Neural algorithm models for solving hard (e.g., combinatorial optimization) problems remain challenging (both in theory and practice)
  - Optimization of such neural models to achieve generalization widely open
  - Are there “universal” recipes for more broader classes of problems / algorithms ?
  - Composition of such generalizable modules? collaboration of neural algorithmic agents?
  - How to probe a learned model to identify what “procedure” is learned?
  - Recurrent, chain-of-thoughts, or looped mechanism
  - …


THANK
YOU!