Stacking Deep Set Networks and Pooling by Quantiles

Zhuojun Chen [1] **Xinghua Zhu** [1] **Dongzhe Su** [1] **Justin C. I. Chuang** [1]

Abstract

We propose Stacked Deep Sets and Quantile Pooling for learning tasks on set data. We introduce Quantile Pooling, a novel permutation-invariant pooling operation that synergizes max and average pooling. Just like max pooling, quantile pooling emphasizes the most salient features of the data. Like average pooling, it captures the overall distribution and subtle features of the data. Like both, it is lightweight and fast. We demonstrate the effectiveness of our approach in a variety of tasks, showing that quantile pooling can outperform both max and average pooling in each of their respective strengths. We also introduce a variant of deep set networks that is more expressive and universal. While Quantile Pooling balances robustness and sensitivity, Stacked Deep Sets enhances learning with depth.

1. Introduction

Deep learning has made remarkable strides in various domains by leveraging the representational power of neural networks. A fundamental challenge in this field is the effective processing and learning from set-structured data (Szabó et al., 2016). The seminal works of "Deep Sets" (Zaheer et al., 2017), "Pointnet" (Qi et al., 2017a) and "Set Transformer" (Lee et al., 2019) have propelled advancements in this area by demonstrating that permutation-invariant functions can model set-based data effectively.

Pooling operations are essential in neural network architectures for summarizing variable-sized, unordered set data. Max pooling, highlighted in Pointnet, excels in robustness by capturing the most significant features, while average pooling computes a mean that better represents the overall data distribution, leading to a more generalized feature aggregation. However, such pooling techniques come with their limitations and trade-offs, many of which have been well studied in the literature (Wagstaff et al., 2019; Bueno & Hylton, 2021; Wagstaff et al., 2022).

The search for a more suitable pooling function for set data has prompted the evaluation of alternatives. Learned pooling methods, which adaptively weight set elements based on their importance, show promise (Lee et al., 2019; Zhao et al., 2021; Naderializadeh et al., 2021; Bartunov et al., 2022; Zhang et al., 2019). However, their high computational expense limits practicality for many applications (refer to Appendix A.4 for a comparison).

In our observation on efficient pooling operations, we find that while max pooling is effective in highlighting dominant features, it can neglect subtle yet informative aspects of the data, leading to potential information loss. On the other hand, average pooling, despite its inclusive approach, risks diminishing unique features by uniformly averaging all points. This balance poses a significant challenge in handling complex data distributions where a complete representation demands acknowledgement of both subtle nuances and prominent characteristics.

This paper introduces a new paradigm of pooling for neural networks in the form of *Quantile Pooling*, a method that synergizes the extremes of max and average pooling. Quantile pooling has the potential to offer a dynamic balance, selectively emphasizing the most relevant aspects of the data depending on the context, a flexibility that is lacking in traditional pooling methods. Building on the *Stacked Deep Sets* architecture, initially introduced by Zaheer et al. (2017) but not fully investigated before, our approach showcases that it surpasses the performance of other pooling strategies and network architectures while maintaining computational efficiency.

2. The Quantile Pooling Family

Consider a probability space $(\Omega, \mathcal{F}, P)$ and a random variable $X : \Omega \to \mathbb{R}$ with cumulative distribution function $F_X : \mathbb{R} \to [0, 1]$. X has bounded support $[a, b]$, where $a = \inf\{x \in \mathbb{R} : P(X \leq x) > 0\}$ and $b = \sup\{x \in \mathbb{R} : P(X \geq x) > 0\}$. A symmetric pooling function $\mathcal{P}$ over the distribution of X can be represented by an operator that aggregates the values of X, taking into account the symmetry of the underlying distribution.

[1]ASTRI, Hong Kong, China. Correspondence to: Zhuojun Chen <georgechen@astri.org>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

Let us define a functional $\mathcal{P} : \mathcal{F} \to \mathbb{R}$ such that for a set of real numbers $\{x_1, x_2, \ldots, x_n\}$ with associated probabilities $\{p_1, p_2, \ldots, p_n\}$, we have:

$$\mathcal{P}(x_1, x_2, \ldots, x_n; p_1, p_2, \ldots, p_n) = \int_0^1 Q_X(p) d\nu(p)$$

where $Q_X(p)$ is the quantile function, the inverse of F_X, and ν is a measure on $[0, 1]$ that reflects the pooling strategy. The measure ν is absolutely continuous with respect to the Lebesgue measure μ, with a density function f_ν that satisfies $f_\nu(p) \geq 0$ for all $p \in [0, 1]$ and $\int_0^1 f_\nu(p) d\mu(p) = 1$. Thus, we can define:

Definition 2.1. For a given quantile $q \in [0, 1]$, the quantile pooling operation $\mathcal{P}_q(X)$ is defined as:

$$\mathcal{P}_q(X) = \int_0^1 Q_X(p) f_{\nu_q}(p)\, dp \tag{1}$$

where $f_{\nu_q}(p)$ is the density function associated with measure ν, specifically in the case of δ quantile pooling, a Dirac delta function $\delta(p - q)$ centered at q.

We endow flexibility to f_ν by not restricting it to be δ.

Lemma 2.2. *Quantile pooling approximates max pooling, as $q \to 1$, with a density function $f_{\nu_\epsilon}(p)$ concentrated in $[1 - \epsilon, 1]$:*

$$\lim_{q \to 1} \mathcal{P}_q(X) = Q_X(1) \equiv \max(X)$$

This means the measure ν concentrates all its mass at the supremum of the support of X, and the pooling operation $\mathcal{P}_q(X)$ evaluates the quantile function $Q_X(p)$ at $p = 1$, which is equivalent to max pooling.

Lemma 2.3. *Quantile pooling with a density function $f_{\nu_\epsilon}(p)$ spread over an interval $[q - \epsilon, q + \epsilon]$ approximates average pooling as ϵ approaches 0.5:*

$$\lim_{\epsilon \to 0.5} \mathcal{P}_\epsilon(X) = \mathbb{E}[X]$$

This means the density function $f_{\nu_\epsilon}(p) = 1$.

In this framework, some examples of pooling functions are:

Max Pooling: $f_\nu(p) = \delta(p - 1)$, where δ is the Dirac delta function. This captures the maximum value of X, as $\mathcal{P}$ effectively evaluates $Q_X(1)$.

Average Pooling: $f_\nu(p) = 1$, the constant function, reflecting the uniform distribution. This yields the expected value (mean) of X.

δ Quantile Pooling: $f_\nu(p) = \delta(p - q)$ for a given quantile q. This selects the q-th quantile of X. This is refered to as δ quantile pooling.

Attention Pooling: $f_\nu(p) = \frac{A(p)}{\int_0^1 A(p)\, dp}$, where A is a learnable function that assigns weights to each quantile. The pooling function used in Set Transformer (Lee et al., 2019) is an instance of this.

Relaxed Quantile Pooling: Relaxed from δ quantile pooling, $f_\nu(p) = \frac{1}{2\epsilon}$ for $p \in [q - \epsilon, q + \epsilon]$ and 0 otherwise. We find this to be a more generalizable form of quantile pooling.

In fact, various pooling functions can be crafted by selecting alternate density functions f_ν, enabling the capture of diverse data distribution aspects suited to specific applications or domains. Our focus will be on relaxed quantile pooling in this paper.

Notably, sum pooling does not fit naturally into this framework unless we relax ν which is a probability measure to a finite measure. However, if the cardinality is constant, sum pooling is equivalent to average pooling (Bueno & Hylton, 2021).

3. Characteristics

Bueno & Hylton's theorem (2021) on the limitations of max and average pooling reveals critical insights into the representation power of these pooling functions, specifically in the context of certain metrics.

We delve deeper into the efficacy of max and average pooling in capturing perturbations, a vital characteristic for handling real-world data, and assess quantile pooling's ability to adeptly conbine their benefits.

3.1. Max Pooling

The advantages of max pooling are well-documented and supported by extensive literature (Qi et al., 2017b; Liu et al., 2020; Ma et al., 2022), underscoring its widespread use in various applications. Max pooling preserves the topological properties of the input set, such as the norm or convexity, as it select the most salient feature of the data. Nevertheless, there are still limitations for being a max pooler.

Since max pooling focuses solely on the maximal values, it is inherently robust to outliers and noise that do not represent the extreme points of the set. That said, we find that max pooling is not as effective at capturing the overall distribution and detailed contrast of the data.

Let $X = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}$ and let $x^* = \max(X)$. Consider a perturbation $\varepsilon > 0$ such that $x^* > x_i + \varepsilon$ for all $i \neq i^*$. For max pooling, we can state the following:

Remark 3.1. Max pooling is invariant to perturbations of the form $x_i \leftarrow x_i + \varepsilon$ for all $i \neq i^*$, where i^* is the index of the maximum element.

On the other hand, consider a perturbed set $X' = X \cup \{x_i +$

$\varepsilon \,|\, x_i \in X, i \neq i^*\}$ where perturbed elements are added to the set.

Remark 3.2. Max pooling is insensitive to the addition of the ε-perturbed elements:

$$\max(X') = \max(X)$$

Remark 3.1 ties in with the robustness of max pooling, as it is invariant to perturbations in the form of noise or outliers. Remark 3.2 marks the limitation of max pooling, as it is unable to capture the addition of nuanced features to the set.

3.2. Average Pooling

Average pooling is a global summary of the data distribution, as it considers all the elements of the set equally. Consider adding just one perturbed element $x_n+\varepsilon$ to the set X, which becomes $X' = \{x_1, x_2, \ldots, x_n, x_n + \varepsilon\}$.

Remark 3.3. Average pooling is sensitive to the addition of the ε-perturbed element:

$$\text{avg}(X') = \frac{1}{n+1}\left(\sum_{i=1}^{n} x_i + x_n + \varepsilon\right)$$

Compared to max pooling, average pooling is more affected by perturbations such as noise or outliers, However when the perturbations are features, average pooling does a better job at capturing them.

Section 5.2 will demonstrate that average pooling is more sensitive to the order of elements in a particular type of sets.

3.3. Quantile Pooling

Quantile pooling serves as a versatile intermediary between average and max pooling. By adjusting its parameters ϵ and q, it can be fine-tuned to prioritize either sensitivity or robustness. The pooling function transitions from an average pooling behavior ($\epsilon \to 0.5$, $q \to 0.5$) to a max pooling behavior ($\epsilon \to 0$, $q \to 1$) with ease. This shift is illustrated in Figure 1, highlighting the adjustable nature of quantile pooling.

Yet, we establish that only a positive ϵ is necessary for the below to hold:

Lemma 3.4. *Let $\mathcal{P}_{\max}$ be the max pooling function and $\mathcal{P}_{q,\epsilon}$ the relaxed quantile pooling function with quantile level q and relaxation parameter ϵ. Let $\Theta_g^{\mathcal{F}}, \Theta_h^{\mathcal{F}}, \Theta_g^{\mathcal{G}}, \Theta_h^{\mathcal{G}}$ be continuous element-wise functions. $\exists\Theta_g^{\mathcal{F}}, \exists\Theta_h^{\mathcal{F}}, \forall\Theta_g^{\mathcal{G}}, \forall\Theta_h^{\mathcal{G}}$, which define set functions $\mathcal{F}(X) = \Theta_g^{\mathcal{F}}\left(\mathcal{P}_{q,\epsilon}\left(\Theta_h^{\mathcal{F}}(X)\right)\right)$ and $\mathcal{G}(X) = \Theta_g^{\mathcal{G}}\left(\mathcal{P}_{\max}\left(\Theta_h^{\mathcal{G}}(X)\right)\right)$ such that $\forall\xi > 0$, $\forall q \in (0,1)$,*

$$|\mathcal{F}(X) - \mathcal{G}(X)| < \xi$$

iff $\epsilon > 0$.

This means that quantile pooling networks can approximate max pooling networks arbitrarily well, as long as $\epsilon > 0$, regardless of q. The proof is provided in Appendix A.3, which is based on Qi's UAT (Qi et al., 2017a) and by proving a continuity of the relaxed quantile pooling $\mathcal{P}_{q,\epsilon}$ w.r.t X.

However, being able to approximate is a theoretical aspect. More practically, we can alter the readiness of quantile pooling to approximate certain functions by tuning the parameters q and ϵ. As we can see in Figure 1, quantile pooling accuracies plunge when approaching max pooling in the sorting task. Providentially, these parameters prove not too tricky to tune, as we find that relaxing the quantile pooling with $q = 0.95$ and $\epsilon = 0.05$ generally yields strong results:

$$f_\nu(p) = \begin{cases} \frac{1}{0.1}, & \text{if } p \in [0.9, 1] \\ 0, & \text{otherwise} \end{cases}$$

This pooling function strikes a balance between max pooling and average pooling by capturing key data features while accounting for the broader distribution, offering adaptability for diverse datasets.

4. Implementation

4.1. Quantile Pooling Layer

Consider the pooling layer operates on a tensor $\mathbf{X} \in \mathbb{R}^{N\times D}$ and outputs a vector $\mathbf{y} \in \mathbb{R}^D$. To approach the quantile pooling function, the quantile range $[q-\epsilon, q+\epsilon]$ is divided into k equal intervals, which determines the granularity of the approximation. This results in a vector of quantile levels $\mathbf{q} \in \mathbb{R}^k$, where $\mathbf{q} = \{q - \epsilon + \frac{2\epsilon}{k}, q - \epsilon + \frac{4\epsilon}{k}, \ldots, q + \epsilon\}$. Instead of computing all k quantiles for all D dimensions, we divide the input tensor $\mathbf{X}$ into k segments along D, emulating sampling evenly across D. For each segment D_i, the corresponding subset $\mathbf{X}_{[:,D_i]}$ is processed to calculate the quantile values. This results in k quantile-pooled vectors $\mathbf{y}_{D_i} \in \mathbb{R}^{1\times D/k}$, which are concatenated to form the output vector $\mathbf{y}$, such that $\mathbf{y} = \|_{i=1}^{k}\ \mathbf{y}_{D_i}$, where $\|$ denotes the concatenation operation. We find that $k = 16$ is sufficient for most tasks.

To compute the quantile values, we refer to Hyndman and Fan's algorithm (Hyndman & Fan, 1996), which involves calculating a real-valued index h, which represents the position of the q-quantile within the ordered sample. When h is an integer, the h-th smallest value, denoted x_h, is taken as the quantile estimate. If h is not an integer, interpolation between $x_{\lfloor h \rfloor}$ and $x_{\lceil h \rceil}$ is used to determine the estimate.

The selection of the q and ϵ is critical for the generalization of the pooling operation across various tasks. For the experiments in this paper, unless otherwise specified, we fix q at 0.95 and ϵ at 0.05, as this configuration has demonstrated a broad generalizability across different datasets and tasks.

While some tasks, such as sorting clusters (Section 5.2), may benefit from a more task-specific selection of p and ϵ, we have deliberately refrained from extensive hyperparameter optimization in those areas.

Notwithstanding, an exploratory (see Figure 1) attempt was made to endow the quantile level q with the capacity to be learned from the data. However, empirical results indicate that the learned values of $\mathbf{q}$ do not deviate significantly from their initializations, suggesting a lack of incentive for the model to adjust this parameter during training. We conjecture that this phenomenon arises due to the static nature of the pooling function, which lacks the dynamic weighting mechanism inherent in more sophisticated architectures such as the self-attention mechanism in transformers (Vaswani et al., 2017). As a result, the introduction of a learnable $\mathbf{q}$ did not yield substantial improvements in our experiments. Therefore, our implementation of quantile pooling is non-parameterized.

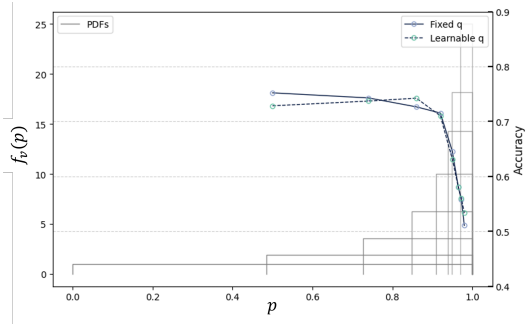

Figure 1. Transitioning of quantile pooling functions and learning of quantile vector $\mathbf{q}$. Varying the quantile level q and the relaxation parameter ϵ results in a smooth transition between max pooling and average pooling. Learning experiments are conducted on the sorting clusters problem in Section 5.2.

4.2. Stacked Deep Sets

We propose an architecture that extends the concept of deep set networks through a series of transformational and aggregational layers, each designed to extract features at varying levels of abstraction. The depth of the network is facilitated by stacking these layers, with each layer l comprising a transformation followed by an aggregation step.

A pivotal aspect of the architecture is the incorporation of residual learning (He et al., 2016), which is designed to refine the learning process within deep networks. An vanilla residual learning mechanism can be expressed as follows:

Given the output $\mathbf{h}^{(l)} \in \mathbb{R}^{N\times D}$ from the l-th layer, where D is the feature dimension, the subsequent layer output $\mathbf{h}^{(l+1)} \in \mathbb{R}^{D}$ is computed by:

$$\mathbf{h}^{(l+1)} = \mathbf{h}^{(l)} + \mathbf{W}^{(l)} \left[\mathbf{h}^{(l)} \oplus \mathcal{P}\left(\Theta^{(l)}\left(\mathbf{h}^{(l)}\right)\right)\right] \quad (2)$$

where $\oplus$ denotes vector concatenation, $\mathbf{W}^{(l)} \in \mathbb{R}^{D\times 2D}$ represents the learnable weight matrix associated with the l-th layer, $\mathcal{P}$ signifies the quantile pooling operation, and $\Theta^{(l)}$ embodies the multi-layer perceptron (MLP) that effects a non-linear transformation.

Each layer l applies a non-linear transformation $\Theta^{(l)}$ to $\mathbf{h}^{(l)}$, followed by the relaxed quantile pooling operation $\mathcal{P}$ for feature aggregation. The aggregation is concatenated with the original output $\mathbf{h}^{(l)}$, and the ensemble is linearly transformed via $\mathbf{W}^{(l)}$, producing an enriched feature representation that is summed with $\mathbf{h}^{(l)}$ to yield $\mathbf{h}^{(l+1)}$. Note that the operation $\mathbf{W}\,(h \oplus g)$ can be done efficiently by $h\mathbf{W}^{\intercal}_{0:D} + g\mathbf{W}^{\intercal}_{D:2D}$, where $h \in \mathbb{R}^{N\times D}$ and $g \in \mathbb{R}^{1\times D}$.

In a variant of the above architecture, we introduce a modification to the transformation and aggregation mechanism within each layer.

Let $\mathbf{g}^{(l)} = \mathcal{P}\left(\Theta^{(l)}\left(\mathbf{h}^{(l)}\right)\right)$ be the pooled feature, then the output of the l-th layer $\mathbf{h}^{(l+1)}$ is given by:

$$\mathbf{h}^{(l+1)} = \mathbf{h}^{(l)} + \mathbf{W}^{(l)} \left[\left(\mathbf{h}^{(l)} - \mathbf{g}^{(l)}\right) \oplus \Theta^{(l)}_{g}\left(\mathbf{g}^{(l)}\right)\right] \quad (3)$$

Here, $\mathbf{g}^{(l)}$ represents the pooled feature that is derived from the l-th layer's output, and $\Theta^{(l)}_{g}$ is the multi-layer perceptron applied to $\mathbf{g}^{(l)}$. The introduction of $\Theta^{(l)}_{g}$ is intended to provide a richer and more expressive feature representation by applying a non-linear transformation to the pooled feature. The subtraction $(\mathbf{h}^{(l)} - \mathbf{g}^{(l)})$ aims to isolate the residual information, thereby enhancing the model's capacity to discern and learn from the subtle nuances in the data. This in some way shares a similar motivation with Chen's recycling procedure (Chen et al., 2022).

This modification has negligible impact on the computational complexity of the architecture (2% additional forward time), for the added $\Theta^{(l)}_{g}(\mathbf{g}^{(l)})$ is a function of the pooled feature $\mathbf{g}^{(l)}$, which is $1/N$ of the computation of $\Theta^{(l)}$ applied to $\mathbf{h}^{(l)}$. Most of the extra computation comes from the subtraction $(\mathbf{h}^{(l)} - \mathbf{g}^{(l)})$, which is also minor.

The improvement of this modification over the vanilla implementation is universal across all tasks, which is demonstrated in Appendix A.7. This architecture enables the extraction of more complex feature representations. Experiments in the next section show it outperforms Deep Sets and often exceeds Set Transformer's learned pooling.

Scaling: The architecture can be scaled to a deeper network by stacking more layers as described by Equation (3), and we observe a linear improvement in performance with an increased number of layers across many tasks. In Appendix A.5, we demonstrate the scaling of the architecture to as many as 256 layers. For experiments in the main text, we only use 3 layers.

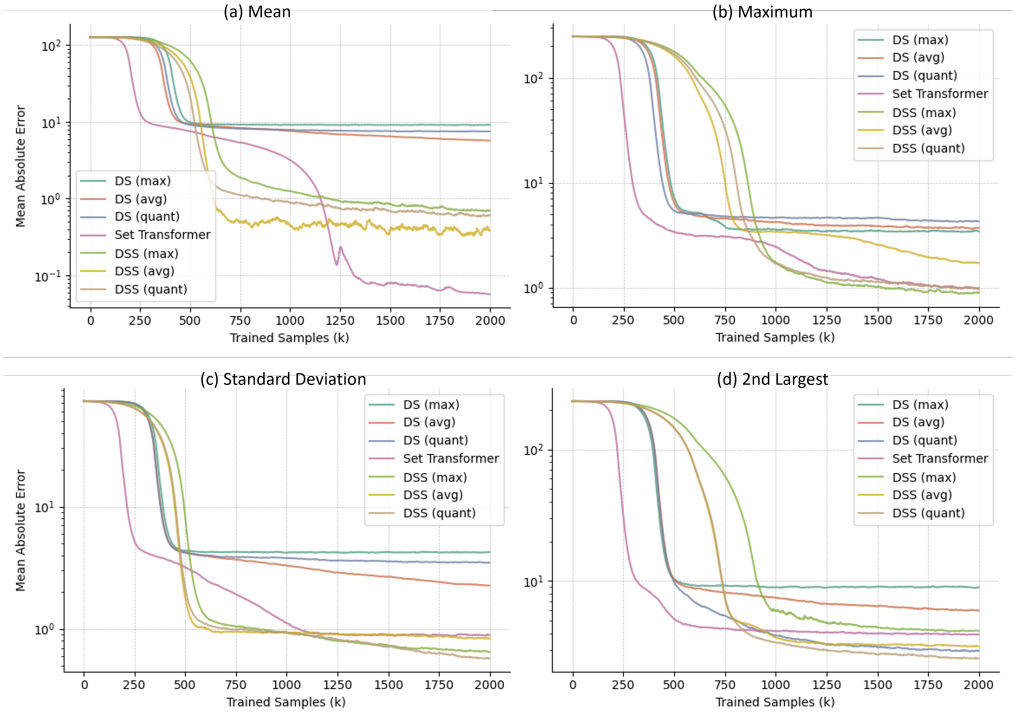

Figure 2. Sample statistic regression results. Refer to Table 1 for the summary of the results.

5. Experiments

Experiments are conducted to reveal the capability of different pooling methods in a variety of tasks. Detailed experimental settings are provided in Appendix A.11.

5.1. Sample Statistic Regression

We formulate a simple regression problem to demonstrate the effectiveness of quantile pooling.

Sample mean regression: Given a set of samples $X \subset \mathbb{N}$, $X \sim \mathcal{U}\{0, 1, 2, \ldots, 255\}$, we want to predict the mean of the samples $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$. To avoid trivial solutions, the input set is first converted to one-hot encoding.

In this task, Set Transformer with PMA outperforms all other methods by a large margin, the task still seeming trivial for it. This will explain why Set Transformer also performs better than Stacked Deep Sets in the Mixture of Gaussians task in Section 5.4. Average pooling is superior to max pooling, while quantile pooling is not far behind average pooling.

Sample maximum regression: With the same setting as above, this task is to predict the maximum of the samples.

In this task, max pooling is the best, followed by quantile pooling and PMA.

Sample standard deviation regression: With the same setting as above, this task is to predict the standard deviation of the samples.

This is a task not trivial for all methods. Quantile pooling demonstrates the best performance, followed by max pooling.

Sample 2nd largest regression: With the same setting as above, this task is to predict the second largest of the samples.

In this task, quantile pooling is the best, with surprisingly simple Deep Sets (DS) utilizing quantile pooling as a close runner-up. Overall, this is not an easy task for any method, as evidenced by the comparatively large errors observed.

Discussion: The training curves presented in Figure 2 and the summarized results in Table 1 provide valuable insights into the performance of various pooling strategies when dealing with set-structured data. The Set Transformer, utilizing Pooling by Multihead Attention (PMA), consistently outperforms all Deep Sets (DS) variations, corroborating the findings from Lee et al. (2019). While different pooling methods exhibit distinct strengths within the Deep Sets architecture, its overall capability appears somewhat constrained. Conversely, the Stacked Deep Sets (DSS) architecture significantly improves upon Deep Sets and frequently surpasses the Set Transformer, with the notable exception of mean prediction.

Within the Stacked Deep Sets framework, average pooling

excels at estimating means, max pooling at capturing maximum values, and quantile pooling at identifying the second-largest elements. When it comes to the more nuanced task of predicting the standard deviation, quantile pooling emerges as the superior method, capturing the underlying variability of the data with remarkable precision.

Table 1. Comparison of Pooling Methods Across Regression Tasks. The integer in each cell represents the rank of the pooling method in the corresponding task, where a rank of 1 indicates the best performance. "DS" stands for Deep Sets, "SET TR" stands for Set Transformer, "DSS" stands for Deep Sets Stacked (stacked Deep Sets), "PMA" stands for Pooling by Multihead Attention used in Set Transformer.

NET	POOLING	MEAN	MAX	STD	2ND MX
DS	MAX	7	5	7	7
DS	AVG	5	6	5	6
DS	QUANT	6	7	6	2
SET TR	PMA	1	=2	4	4
DSS	MAX	4	1	2	5
DSS	AVG	2	4	3	3
DSS	QUANT	3	=2	1	1

Figure 3. Visualisation of sorting clusters. For visualization purposes, this perturbation is sampled from a normal distribution.

5.2. Sorting Clusters

We formulate a sorting problem as empirical evidence to justify the characteristics of pooling functions discussed in Section 3. The sorting problem serves as a practical scenario to understand how different pooling strategies handle perturbations and maintain the relative order within a set.

We first construct a multiset X by sampling n integers from a uniform distribution over $\mathbb{N}$. Repeated elements are included to ensure that the multiset forms small clusters. The multiset is then perturbed by adding a uniform noise to each element, with each integer $x_i \in X$ subjected to a uniform perturbation $\epsilon_i \sim U(-0.1, 0.1)$, such that the perturbed multiset $X' = \{x_i + \epsilon_i | x_i \in X\}$. For example, a multiset $X = \{1, 1, 2, 2, 3\}$ is perturbed by $\epsilon = \{-0.01, 0.01, 0.0, 0.09, -0.05\}$, which after perturbation becomes $X' = \{0.99, 1.01, 2.0, 2.09, 2.95\}$. The minimum distance of X', denoted $d_{min}(X')$, is deliberately kept small to create a tightly bound set:

$$d_{min}(X') = \inf\{|x_i - x_j| : x_i, x_j \in X'\} \leq \delta$$

where δ is a small positive number, such that $d_{min}(X) \geq 1 \gg \delta \geq d_{min}(X')$, signifying the maximum allowable distance between any two elements in X to maintain the compactness of the clusters.

The task is to predict the sorted index (ranking) of each element in the perturbed set X', which makes it an element-wise classification problem. The performance metric for this task is the prediction accuracy of the sorted indices.

Results and discussion: Stacked Deep Sets utilizing average and quantile pooling surpass the performance of those with max pooling and Set Transformer. These findings align with the theoretical insights discussed in Section 3. The tendency of average pooling to be sensitive to minor changes enables it to preserve the order within smaller data clusters. Quantile pooling, particularly with a high quantile (i.e., $q = 0.95, \epsilon = 0.05$) effectively retains the intricate structure of these clusters while also accounting for the broader distribution, leading to its superior results. In this experiment, quantile pooling is slightly better than average pooling. Moreover, we observe that expanding the network in both depth and width further improves quantile pooling's performance over average pooling (similar to Figure 10).

In contrast, max pooling tends to fall short in these scenarios due to its disregard for subtle differences among set elements, opting to highlight only the most dominant value. The performance of the Set Transformer shows that, despite its ability to assign varying weights to different elements through its attention mechanism, it might possess limitations akin to max pooling.

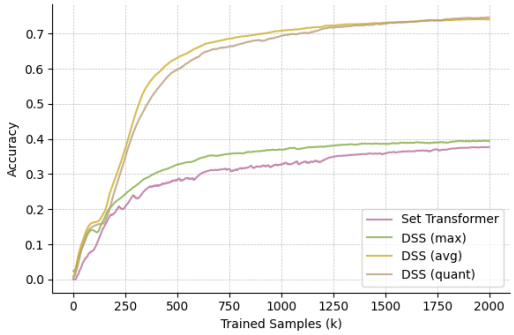

Figure 4. Sorting clusters results.

5.3. Russian-dollable Envolopes

The problem of Russian-dollable Envelopes provides a robust testbed for evaluating the capacity of neural architectures to understand and encode hierarchical spatial relationships. Consider a collection of envelopes $E = \{e_1, e_2, \ldots, e_n\}$, where each envelope e_i is characterized by its continuous width and height $e_i = (w_i, h_i) \in \mathbb{R}^2$. An envelope e_i can be placed inside another envelope e_j if and only if both dimensions of e_i are strictly smaller than those of e_j, formally $w_i < w_j$ and $h_i < h_j$. The objective is to identify the longest sequence of envelopes $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ such that each envelope can be nested within the subsequent one, according to the aforementioned condition. This sequence represents the maximum number of Russian-dollable envelopes.

The task requires discerning intricate orderings based on two-dimensional continuous variables and constructing a maximal chain in this ordered set:

$$\max_{\pi}\{k : e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(k)} \mid (w_{\pi(i)} < w_{\pi(i+1)}) \wedge (h_{\pi(i)} < h_{\pi(i+1)})\}$$

where π is a permutation of the indices $\{1, 2, \ldots, n\}$ and k is the length of the longest nesting sequence.

We formulate this problem as a classification task, where the objective is to predict the number of envelopes in the longest nesting sequence. The performance metric for this task is the prediction accuracy of the number of envelopes in the longest nesting sequence. Note that the input to the model is again unordered, and the envelope dimensions are sampled continuously from a uniform distribution unlike in Section 5.2.

Results and discussion: The training curves are presented in Figure 5. Quantile pooling outshines others, whereas Set Transformer lagging behind all by a large margin. This task is emblematic of problems where spatial hierarchies are paramount. Notably sorting is a sub-problem here. Max pooling proves better than average pooling in complex scenarios without perturbations. Crucially, quantile pooling bypasses the constraints of max and average pooling, highlighting its resilience and flexibility. Furthermore, we observe advantage of quantile pooling over max pooling widens as we scale the network (Appendix A.5).

5.4. Mixture of Gaussians

The Mixture of Gaussians (MoG) problem is a classic clustering task that has been extensively studied in the literature. It is a problem also presented as a set problem in the original Set Transformer paper (Lee et al., 2019). The objective is to determine the set of parameters that maximize the likelihood of the observed data.

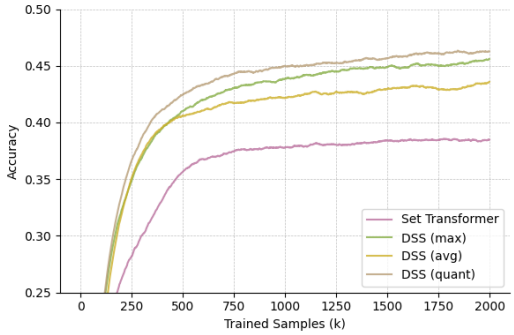

Figure 5. Russian-dollable envelopes results.

Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$, where each $x_i \in \mathbb{R}^d$ is drawn from a mixture of Gaussian distributions, the task is to find the optimal parameters θ^* that maximize the log-likelihood function:

$$\theta^*(X) = \arg\max_{\theta} \log p(X; \theta)$$

The parameter set θ typically includes the means, covariances, and mixing coefficients of the Gaussian components in the mixture. The log-likelihood of the data under a MoG model is given by:

$$\log p(X; \theta) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i; \mu_k, \Sigma_k) \right)$$

where K is the number of Gaussian components, π_k are the mixing coefficients such that $\sum_{k=1}^{K} \pi_k = 1$, $\mathcal{N}(x|\mu, \Sigma)$ denotes the Gaussian distribution with mean μ and covariance matrix Σ, and μ_k and Σ_k are the parameters of the k-th Gaussian component.

Results and discussion: Quantile pooling exhibits the most promising results among Stacked Deep Sets, suggesting that quantile pooling's ability to capture nuanced distributional features is particularly advantageous for modeling the complex structure of MoGs. Max pooling also performs well, whilst average pooling slightly lag behind. This suggests this task is not simple statistic modeling, as it requires learning of sometimes convoluted clusters (Lee et al., 2019) which complicates its nature. The Set Transformer architecture with PMA achieves the best performance, attributable to its excellent ability to model the most important statistic, mean, as demonstrated in Section 5.1.

5.5. Point Cloud Classification

In 3D data analysis, classifying point clouds is a key challenge. We explore quantile pooling on ModelNet40 (Wu et al., 2015)—a benchmark dataset with 40 classes of 3D objects. Unlike methods that use point relationships (typically

Table 2. Mixture of Gaussians Results

NET	POOLING	LOG LIKELIHOOD
ORACLE	-	-1.4778
SET TR	PMA	**-1.4867**
DSS	MAX	-1.5186
DSS	AVG	-1.5501
DSS	QUANT	-1.4937

Table 3. Point Cloud Classification Results

NET	POOLING	ACCURACY
DS	MAX	0.8956 ± 0.0024
DS	AVG	0.8638 ± 0.0032
DS	QUANT	0.8956 ± 0.0014
SET TR	PMA	0.8853 ± 0.0024
DSS	MAX	0.9030 ± 0.0035
DSS	AVG	0.8997 ± 0.0012
DSS	QUANT	$\mathbf{0.9071 \pm 0.0022}$

Pointnet++ (Qi et al., 2017b)), we test our models without explicit consideration of the spatial relationships between points.

We also test quantile pooling within PointMLP (Ma et al., 2022), a strong baseline and well-established framework that does consider spatial point relations using KNN, by substituting all its max pooling layers with quantile pooling.

Results and discussion: In our experiments within Stacked Deep Sets, quantile pooling outperformed both max and average pooling, as detailed in Table 3. This indicates that quantile pooling excels at capturing deep semantic features in complex datasets. Though average pooling works so poorly with Deep Sets, it gains a significant boost in performance when used in Stacked Deep Sets.

In subsequent tests on the PointMLP architecture, replacing max pooling with quantile pooling produced a significant boost in classification accuracy. These results highlight quantile pooling's adaptability and effectiveness as a robust component in point cloud neural networks.

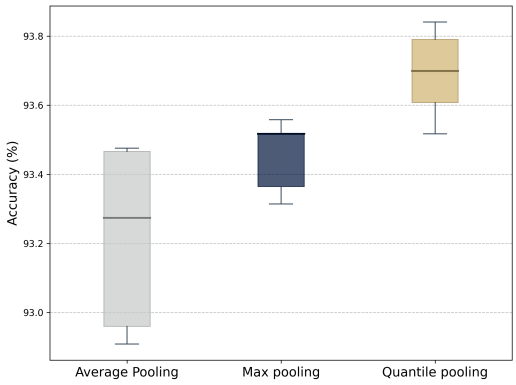

Figure 6. Classification accuracy for ModelNet40 using average, max and quantile pooling in the PointMLP model.

6. Related Work

Two cardinal approaches for obtaining permutation invariant encodings of sets is average and max pooling (Zaheer et al., 2017; Qi et al., 2017a). It has been found that the latent space dimensionality of the result of the pooling operation needs to be at least as large as the number of inputs in order to guarantee universal function approximation (Wagstaff et al., 2019). Limitations of both pooling methods have been identified (Bueno & Hylton, 2021).

Janossy Pooling generalizes coordinate-wise pooling to incorporate higher-order interactions between set elements (Murphy et al., 2019). However, its effectiveness in improving approximation results in the general case is uncertain (Wagstaff et al., 2022).

Principal Neighbourhood Aggregation (PNA) (Corso et al., 2020) addresses the limitations of individual pooling operators by combining four different pooling operators and three scaling strategies for graph models.

Equilibrium Aggregation generalizes the functional form of the aggregation operator beyond traditional pooling methods (Bartunov et al., 2022). PSWE introduces a geometrically-interpretable pooling method using sliced-Wasserstein distances (Naderializadeh et al., 2021).

The Set Transformer (Lee et al., 2019) is a neural network architecture that uses self-attention to aggregate information from the set elements. Although transformers have been shown to be effective in modeling long-range dependencies in sequential data (Brown et al., 2020), they may not be as ready to be trained on set-structured data.

Combining different pooling functions is an artless yet favorable strategy in numerous works, which we do not list here. Appendix A.10 demonstrates the unique merits of quantile pooling over combination of max and average pooling.

7. Conclusion

We have demonstrated that quantile pooling is a robust and flexible pooling strategy. We have also proposed a novel architecture, Stacked Deep Sets, that extends the Deep Sets framework to enable the extraction of more complex feature representations. Stacked Deep Sets and quantile pooling prove to be a simple, efficient, and powerful combination for modeling set-structured data.

Acknowledgements

This work is supported by the Smart Traffic Fund (STF) for the application aspects in the project titled "*Evaluation of Smart Mobility Roadside Infrastructure for Connected Autonomous Vehicles (PSRI/52/2210/RA_CAV)*," granted by Hong Kong Transport Department, which facilitated the application aspects of the study.

This work is also supported by the Innovation and Technology Fund (ITF) for the project titled "*Generative Pretrained Large Traffic Model for Multi-Modal Traffic Data Understanding (ARD/309)*," granted by Hong Kong Innovation and Technology Commission, which specifically supported the design of the methodology.

The authors would like to acknowledge the support from both the Innovation and Technology Fund & Hetao Shenzhen-Hong Kong Science and Technology Innovation Cooperation Zone Shenzhen Park Project (No.HTHZQSWS-KCCYB-2023042) for the research and development contributions.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Bartunov, S., Fuchs, F. B., and Lillicrap, T. P. Equilibrium aggregation: Encoding sets via optimization. In *Uncertainty in Artificial Intelligence*, pp. 139–149. PMLR, 2022.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Bueno, C. and Hylton, A. On the representation power of set pooling networks. *Advances in Neural Information Processing Systems*, 34:17170–17182, 2021.

Chen, J., Kakillioglu, B., Ren, H., and Velipasalar, S. Why discard if you can recycle?: A recycling max pooling module for 3d point cloud analysis. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 559–567, 2022.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33: 13260–13271, 2020.

Csáji, B. C. et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Hyndman, R. J. and Fan, Y. Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365, 1996.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 3744–3753, 2019.

Liu, Z., Hu, H., Cao, Y., Zhang, Z., and Tong, X. A closer look at local aggregation operators in point cloud analysis. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*, pp. 326–342. Springer, 2020.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ma, X., Qin, C., You, H., Ran, H., and Fu, Y. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. *arXiv preprint arXiv:2202.07123*, 2022.

Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019.

Naderializadeh, N., Comer, J. F., Andrews, R., Hoffmann, H., and Kolouri, S. Pooling by sliced-wasserstein embedding. *Advances in Neural Information Processing Systems*, 34:3389–3400, 2021.

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

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017b.

Szabó, Z., Sriperumbudur, B. K., Póczos, B., and Gretton, A. Learning theory for distribution regression. *The Journal of Machine Learning Research*, 17(1):5272–5311, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wagstaff, E., Fuchs, F., Engelcke, M., Posner, I., and Osborne, M. A. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pp. 6487–6494. PMLR, 2019.

Wagstaff, E., Fuchs, F. B., Engelcke, M., Osborne, M. A., and Posner, I. Universal approximation of functions on sets. *The Journal of Machine Learning Research*, 23(1): 6762–6817, 2022.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zhang, Y., Hare, J., and Prügel-Bennett, A. Fspool: Learning set representations with featurewise sort pooling. *arXiv preprint arXiv:1906.02795*, 2019.

Zhao, H., Jiang, L., Jia, J., Torr, P. H., and Koltun, V. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 16259–16268, 2021.

A. Appendix

A.1. Mathematical Background

In this appendix, we provide a derivation of the generalized quantile pooling integral in Equation (1).

For a continuous random variable X, the cumulative distribution function (CDF), denoted as $F_X(x)$, is defined as the probability that the random variable X will take a value less than or equal to x:

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} f_X(t)\, dt \tag{4}$$

where f_X represents the probability density function (PDF) of X.

The quantile function $Q_X(p)$ is the functional inverse of the CDF, $F_X(x)$, and is defined for $p \in [0, 1]$ as:

$$Q_X(p) = F_X^{-1}(p) = \inf\{x \in \mathbb{R} : F_X(x) \geq p\} \tag{5}$$

The expected value of X, traditionally calculated using the PDF, can alternatively be expressed using the quantile function:

$$E[X] = \int_0^1 Q_X(p)\, dp \tag{6}$$

This expression for the expected value is derived from the relationship between the expected value and the integral of the CDF.

We can generalize the concept of expected value to include a measure that assigns weights to each quantile, denoted as $d\nu(p)$. This leads us to define a generalized pooling function $\mathcal{P}(X)$:

$$\mathcal{P}(X) = \int_0^1 Q_X(p)\, d\nu(p) \tag{7}$$

Here, $d\nu(p)$ could represent any finite measure on the interval $[0, 1]$. In the special case where $d\nu(p)$ is the Lebesgue measure (the standard measure assigning length to intervals on the real line), the generalized pooling function $\mathcal{P}(X)$ simplifies to $E[X]$.

This general pooling function can be interpreted as a weighted average where the weights are given by the measure ν. If the measure ν is absolutely continuous with respect to the Lebesgue measure μ, there exists a density function $f_\nu(p)$ such that for any measurable set A, $\nu(A) = \int_A f_\nu(p)\, d\mu(p)$. Consequently, the pooling function $\mathcal{P}(X)$ can be expressed in terms of the Lebesgue measure as:

$$\mathcal{P}(X) = \int_0^1 Q_X(p) f_\nu(p)\, d\mu(p) \tag{8}$$

As the Lebesgue measure on $p \in [0, 1]$ is equivalent to the uniform distribution, the pooling function $\mathcal{P}(X)$ can be expressed as:

$$\mathcal{P}(X) = \int_0^1 Q_X(p) f_\nu(p)\, dp \tag{9}$$

In this case, $f_\nu(p)$ represents the Radon-Nikodym derivative of ν with respect to μ, often interpreted as the density of ν relative to μ. When $f_\nu(p) = 1$, the measure ν coincides with the Lebesgue measure, and the pooling function $\mathcal{P}(X)$ simplifies to $E[X]$, assuming $Q_X(p)$ represents a probability density function.

A.2. Proof of Lemma 2.3

Lemma 2.3: Quantile pooling with a density function $f_{\nu_\epsilon}(p)$ spread over an interval $[q-\epsilon, q+\epsilon]$ approximates average pooling as ϵ approaches 0.5.

Proof. Consider the density function $f_{\nu_\epsilon}(p)$ defined over the interval $[q-\epsilon, q+\epsilon]$ as follows:

$$f_{\nu_\epsilon}(p) = \begin{cases} 1/(2\epsilon), & \text{for } p \in [q-\epsilon, q+\epsilon] \\ 0, & \text{otherwise} \end{cases}$$

The pooling operation $\mathcal{P}_\epsilon(X)$ is then:

$$\mathcal{P}_\epsilon(X) = \int_0^1 Q_X(p) f_{\nu_\epsilon}(p)\, dp$$

Because of the interval $[q-\epsilon, q+\epsilon]$ being contained within $[0,1]$ for all $\epsilon \in [0, 0.5]$, as ϵ approaches 0.5, the interval $[q-\epsilon, q+\epsilon]$ covers the entire support of p, and the density function $f_{\nu_\epsilon}(p)$ converges to a uniform density over $[0,1]$, i.e. $f_{\nu_\epsilon}(p) \to 1$. Therefore, the pooling operation $\mathcal{P}_\epsilon(X)$ converges to the mean of X:

$$\lim_{\epsilon \to 0.5} \mathcal{P}_\epsilon(X) = \mathbb{E}[X]$$

□

A.3. Proof of Lemma 3.4

Lemma 3.4: Let $\mathcal{P}_{\max}$ be the max pooling function and $\mathcal{P}_{q,\epsilon}$ the relaxed quantile pooling function with quantile level q and relaxation parameter ϵ. Let $\Theta_g^{\mathcal{F}}, \Theta_h^{\mathcal{F}}, \Theta_g^{\mathcal{G}}, \Theta_h^{\mathcal{G}}$ be continuous element-wise functions. $\exists \Theta_g^{\mathcal{F}}, \exists \Theta_h^{\mathcal{F}}, \forall \Theta_g^{\mathcal{G}}, \forall \Theta_h^{\mathcal{G}}$, which define set functions $\mathcal{F}(X) = \Theta_g^{\mathcal{F}}\left(\mathcal{P}_{q,\epsilon}\left(\Theta_h^{\mathcal{F}}(X)\right)\right)$ and $\mathcal{G}(X) = \Theta_g^{\mathcal{G}}\left(\mathcal{P}_{\max}\left(\Theta_h^{\mathcal{G}}(X)\right)\right)$ such that $\forall \xi > 0, \forall q \in (0,1)$,

$$|\mathcal{F}(X) - \mathcal{G}(X)| < \xi$$

iff $\epsilon > 0$.

Proof. We first prove $\mathcal{P}_{q,\epsilon}$ is continuous with respect to X when $\epsilon > 0$. To do that, we need to establish that small changes in the random variable X lead to small changes in the pooling operation $\mathcal{P}_q(X)$.

By the definition of Equation (1), the pooling operation over a quantile q with a smoothing window ϵ can be written as:

$$\mathcal{P}_q(X) = \int_{q-\epsilon}^{q+\epsilon} Q_X(p) \frac{1}{2\epsilon}\, dp$$

Consider a sequence of random variables $\{X_n\}$ converging in distribution to X. This means that the cumulative distribution functions F_{X_n} converge to F_X at every continuity point of F_X. Since Q_X is the inverse of F_X, we can also say that Q_{X_n} will converge to Q_X at every continuity point of Q_X, which is almost everywhere because Q_X is non-decreasing and hence has at most countably many points of discontinuity.

Let's take two random variables X and X' with quantile functions Q_X and $Q_{X'}$ respectively, such that X and X' are close in distribution. Then their quantile functions will also be close,

Given the definition of the relaxed quantile pooling:

$$\begin{aligned} |\mathcal{P}_q(X) - \mathcal{P}_q(X')| &= \left| \int_{q-\epsilon}^{q+\epsilon} Q_X(p) \frac{1}{2\epsilon}\, dp - \int_{q-\epsilon}^{q+\epsilon} Q_{X'}(p) \frac{1}{2\epsilon}\, dp \right| \\ &= \left| \int_{q-\epsilon}^{q+\epsilon} \frac{Q_X(p) - Q_{X'}(p)}{2\epsilon}\, dp \right| \end{aligned}$$

$$\leq \int_{q-\epsilon}^{q+\epsilon} \left| \frac{Q_X(p) - Q_{X'}(p)}{2\epsilon} \right| dp$$

Since Q_X and $Q_{X'}$ are close due to X and X' being close in distribution, and the factor $\frac{1}{2\epsilon}$ is constant, the integral on the right-hand side will be small if $\epsilon > 0$. This shows that $\mathcal{P}_q(X)$ is indeed continuous with respect to X.

Besides, applying continuous functions $\Theta_g^{\mathcal{F}}, \Theta_h^{\mathcal{F}}$ to $\mathcal{P}_{q,\epsilon}$ will not change the continuity, we can conclude that $\mathcal{F}(X)$ is continuous with respect to X.

Next, we apply the UAT from Pointnet (Qi et al., 2017a), which states $\mathcal{G}(X)$ is a universal approximator of all continuous functions $h : \mathcal{X} \to \mathbb{R}$ on compact sets for any $\xi > 0$:

$$|h(x) - \mathcal{G}(x)| < \xi$$

Given that $\mathcal{P}_{q,\epsilon}$ is a continuous function w.r.t X when $\epsilon > 0$, we can state $\forall \Omega_g^{\mathcal{F}}, \forall \Omega_h^{\mathcal{F}}, \exists \Omega_g^{\mathcal{G}}, \exists \Omega_h^{\mathcal{G}}$ such that

$$|\Omega_g^{\mathcal{F}} \left(\mathcal{P}_{q,\epsilon} \left(\Omega_h^{\mathcal{F}}(X) \right) \right) - \Omega_g^{\mathcal{G}} \left(\mathcal{P}_{\max} \left(\Omega_h^{\mathcal{G}}(X) \right) \right) | < \xi$$

, written as

$$|\mathcal{F}_\Omega(X) - \mathcal{G}_\Omega(X)| < \xi$$

We define any continuous neural network Λ_A to transform the input X to $X_A = \Lambda_A(X)$ such that X_A also satisfies the definition of X, and then we have,

$$|\mathcal{F}_\Omega(X_A) - \mathcal{G}_\Omega(X_A)| < \xi$$

We define any continuous neural network Λ_B to transform the output space of $\mathcal{G}_\Omega$. By continuity, we have:

$$|\Lambda_B(\mathcal{F}_\Omega(X_A)) - \Lambda_B(\mathcal{G}_\Omega(X_A))| < \xi$$

We can define $\mathcal{G}(X)$ as:

$$\mathcal{G}(X) = \Lambda_B(\Omega_g^{\mathcal{G}} \left(\mathcal{P}_{\max} \left(\Omega_h^{\mathcal{G}}(\Lambda_A(X)) \right) \right))$$

$\forall \Lambda_A, \forall \Lambda_B$, let $\Theta_g^{\mathcal{G}} = \Lambda_B \circ \Omega_g^{\mathcal{G}}, \Theta_h^{\mathcal{G}} = \Omega_h^{\mathcal{G}} \circ \Lambda_A$.

$$\mathcal{G}(X) = \Theta_g^{\mathcal{G}} \left(\mathcal{P}_{\max} \left(\Theta_h^{\mathcal{G}}(X) \right) \right)$$

Given that Λ_A and Λ_B are continuous neural networks, by the Universal Approximation Theorem (Hornik et al., 1989; Csáji et al., 2001), Λ_A and Λ_B can approximate any continuous transformations that the compositions $\Theta_g^{\mathcal{G}}$ and $\Theta_h^{\mathcal{G}}$ represent, within any desired level of accuracy on compact subsets of their domain.

On the other hand, we can define $\mathcal{F}(X)$ as:

$$\mathcal{F}(X) = \Lambda_B(\Omega_g^{\mathcal{F}} \left(\mathcal{P}_{q,\epsilon} \left(\Omega_h^{\mathcal{F}}(\Lambda_A(X)) \right) \right))$$

Let $\Theta_g^{\mathcal{F}} = \Lambda_B \circ \Omega_g^{\mathcal{F}}, \Theta_h^{\mathcal{F}} = \Omega_h^{\mathcal{F}} \circ \Lambda_A$, we have:

$$\mathcal{F}(X) = \Theta_g^{\mathcal{F}} \left(\mathcal{P}_{q,\epsilon} \left(\Theta_h^{\mathcal{F}}(X) \right) \right)$$

We can state $\exists \Theta_g^{\mathcal{F}}, \exists \Theta_h^{\mathcal{F}}, \forall \Theta_g^{\mathcal{G}}, \forall \Theta_h^{\mathcal{G}}$,

$$|\Theta_g^{\mathcal{F}} \left(\mathcal{P}_{q,\epsilon} \left(\Theta_h^{\mathcal{F}}(X) \right) \right) - \Theta_g^{\mathcal{G}} \left(\mathcal{P}_{\max} \left(\Theta_h^{\mathcal{G}}(X) \right) \right) | < \xi$$

, which is

$$|\mathcal{F}(X) - \mathcal{G}(X)| < \xi$$

This completes the proof.

□

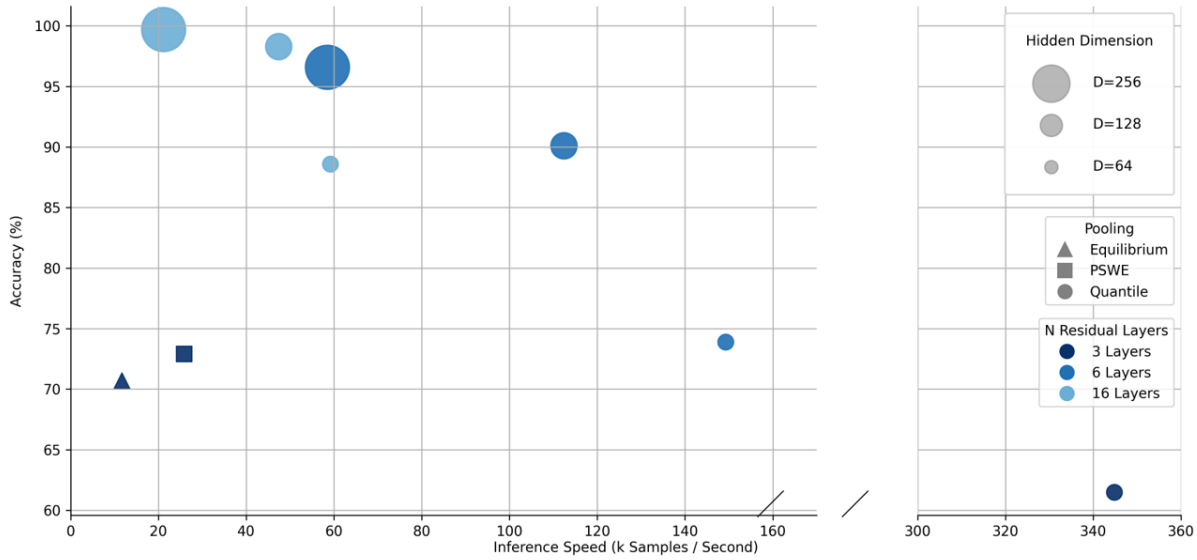

Figure 7. Comparing the performance of different pooling methods on the sorting clusters problem.

A.4. Comparative Analysis of Parameterized Pooling Methods

We have compared the representational effectiveness and computational efficiency of various pooling strategies, including max pooling, average pooling, attention-based pooling, learned pooling strategies, and our proposed quantile pooling.

Parameterized pooling methods are designed to learn a data-driven way of aggregating features, which can potentially lead to better task performance. However, this often comes at the cost of increased computational complexity. We compare two notable parameterized pooling methods:

Equilibrium Pooling: This method incorporates a small ResNet-like architecture within the pooling mechanism and employs a small optimization loop during each forward pass. This iterative process, while expressive, is inherently slow due to the multiple layers of transformations and the optimization routine required for each input set.

PSWE (Pooling by Sliced-Wasserstein Embedding): PSWE requires sorting and indexing over the entire input set , followed by two separate linear layers. Sorting and indexing large sets is computationally intensive, and the linear transformations add significant computational overhead.

In contrast to these complex parameterized methods, quantile pooling operates on a much simpler principle. It computes statistical quantiles, which involve minimal processing akin to finding the max or average values. The process does not involve learning any additional parameters specific to the pooling operation, thus avoiding the overhead associated with iterative optimization or complex sorting operations.

Parameterized methods not only are computationally expensive, but also introduce additional layers of neurons which can also be beneficial to quantile pooling models. In an experiment on the sorting clusters problem (Section 5.2), we compare the performance of these pooling methods. Results are shown in Figure 7.

When quantile pooling compares to Equilibrium Pooling and PSWE without adding any additional layers, it is 30 times faster than Equilibrium pooling and 13 times faster than PSWE, while maintaining around 86% of the accuracy (with less parameters). However, when quantile pooling is also enhanced with additional layers or dimensions, it can achieve much better performance without getting near the computational complexity of Equilibrium Pooling or PSWE.

This brings us to the scaling of our proposed Stacked Deep Sets architecture.

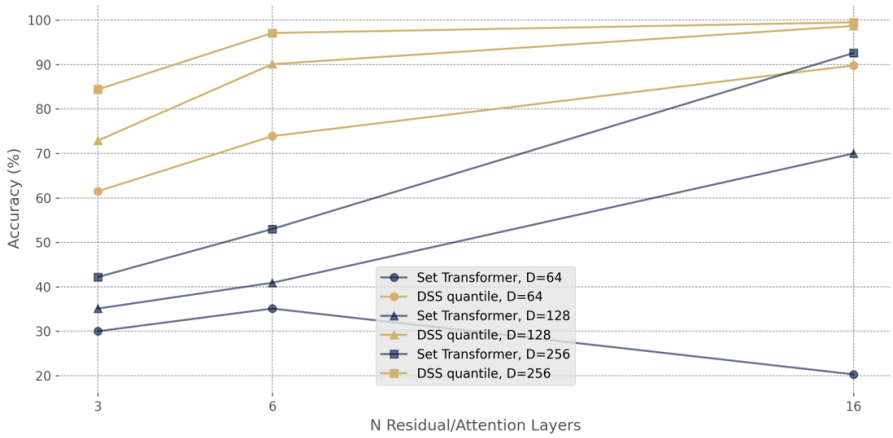

Figure 8. Scaling of Stacked Deep Sets and Set Transformer on the sorting clusters problem.

A.5. Scaling of Stacked Deep Set Networks

We investigate the scaling of the Stacked Deep Sets architecture by varying the number of layers and the hidden dimensionality of each layer.

In our analysis, Stacked Deep Sets consistently outperforms Set Transformer on the sorting clusters problem (Section 5.2), as detailed in Figure 8. To scale Set Transformer, we increase the number of multihead self-attention layers, as well as the hidden dimensionality of each layer. While both architectures scale positively, Set Transformer's performance declines at higher depths with 64 hidden dimensions, suggesting a complex interplay between layer count and dimensionality. Conversely, Stacked Deep Sets achieves near-perfect accuracy with 16 layers and 256 hidden dimensions. Across all scales, Stacked Deep Sets outperforms Set Transformer.

We then evaluate Stacked Deep Sets with various pooling methods on the more challenging Russian-dollable envelopes problem (Section 5.3), as shown in Figure 9. Despite the increased difficulty, Stacked Deep Sets demonstrates linear scaling. Across all scales, quantile pooling outperforms max pooling, which in turn outdoes average pooling.

Scaling Stacked Deep Sets to 256 layers with 256 hidden dimensions (Figure 10) shows continued performance improvements and an increasing advantage for quantile pooling over max pooling at greater depths.

Scaling DS? Attempting to scale basic Deep Sets with both the number of layers and the hidden dimensionality results in hardly any improvement in performance. This starkly contrasts with the scalable Stacked Deep Sets, which consistently exhibit performance gains with scaling, highlighting the importance of architectural choices in scaling neural networks.

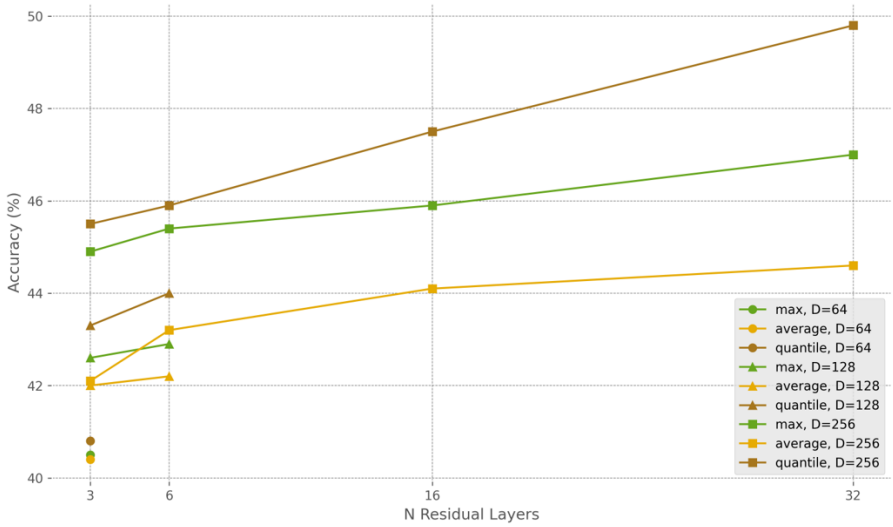

Figure 9. Scaling of Stacked Deep Sets with different pooling methods on the Russian-dollable envelopes problem.

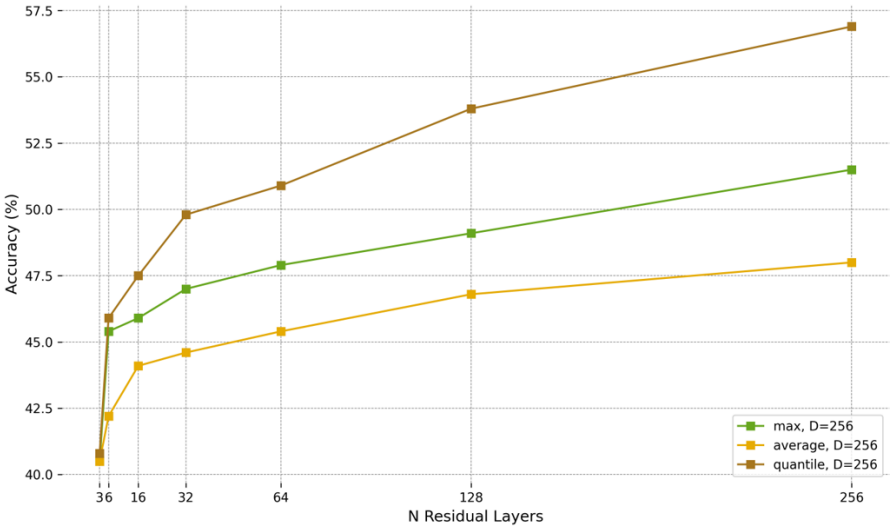

Figure 10. Scaling Stacked Deep Sets to 256 layers

A.6. Network Architecture Details

To implement the Stacked Deep Sets architecture, we first use a three-layer MLP, with batch normalization and ReLU activation, to map the input set to a higher-dimensional space $h^{(0)}$. Recall the residual layers:

$$\mathbf{g}^{(l)} = \mathcal{P}\left(\Theta^{(l)}\left(\mathbf{h}^{(l)}\right)\right)$$

$$\mathbf{h}^{(l+1)} = \mathbf{h}^{(l)} + \mathbf{W}^{(l)}\left[\left(\mathbf{h}^{(l)} - \mathbf{g}^{(l)}\right) \oplus \Theta_g^{(l)}\left(\mathbf{g}^{(l)}\right)\right]$$

For $\Theta_g^{(l)}$, we use a three-layer MLP with the same hidden dimension D, while for $\Theta^{(l)}$, we employ another three-layer MLP with a hidden dimension of $D/2$ in the middle layer. Shrinking the hidden dimension of $\Theta^{(l)}$ is to improve the computational efficiency for a large number of layers. We do not shrink the hidden dimension of $\Theta_g^{(l)}$ because it is only applied to the global feature vector, and thus does not affect the overall computational complexity of the model.

Output layers are implemented as a three-layer MLP, ending without any activation function. If the task requires element-wise outputs, the network architecture described suffices. Otherwise, the final output is taken by averaging of all element-wise outputs. All MLPs come with batch normalization and ReLU activation following each linear layer.

A.7. Comparison of Vanilla Stacking

We summarize the comparison of stacking Equation (2) (vanilla) and Equation (3) in Table 4. Note that Equation (3) compared to vanilla stacking has just around 2% of additional forward pass time.

Table 4. Comparison of vanilla stacking and Stacked Deep Sets

METHOD	POOLING	SORTING (ACCURACY)	ENVELOPES (ACCURACY)	MOG (LOG LIKELIHOOD)
DSS (VANILLA)	QUANT	0.7259	0.4550	-1.5053
DSS	QUANT	**0.7419**	**0.4624**	**-1.4937**

A.8. Inference Time Comparison

We compare the inference time of Stacked Deep Sets with different pooling methods and different choices of k in Appendix A.8. It is shown that quantile pooling has comparable inference time to max pooling and average pooling, and the inference time is little sensitive to the choice of k. Such increases in inference time is not certainty; indeed, we foresee potential enhancements in implementation that could effective mitigate this aspect. We note that the choice of k depends on

set cardinality and hidden dimensionality. Higher cardinality would benefit from a larger k, while larger k would require higher hidden dimensionality to work properly.

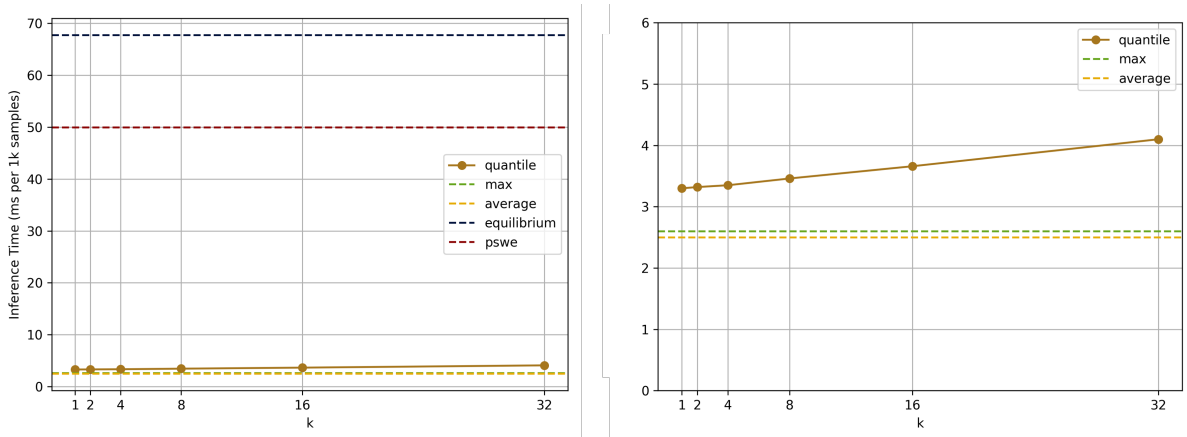

Figure 11. Inference time comparison of Stacked Deep Sets with different pooling methods and different choices of k. Left: comparing different pooling methods. Right: zooming in to quantile vs max vs average pooling.

A.9. Complexity of Quantile Pooling

The computational complexity of quantile pooling can be efficiently managed using algorithmic strategies suited to the specific requirements of identifying the q-th largest element. One common approach involves employing a heap-based algorithm, which maintains a min-heap of size q and operates with a complexity of $O(N \log q)$, where N is the set cardinality. This method is particularly effective when q is small relative to N, as is often the case in practice.

Alternatively, the Quickselect algorithm presents a lower complexity of $O(N)$ on average. In practical scenarios, despite the theoretically higher complexity, the heap-based approach often proves to be faster. This can be attributed to the smaller constant factors and the reduced overhead in managing a heap of limited size compared to handling the entire set.

A.10. Mix Pooling

We explore naive combinations of different pooling methods, which we refer to as mix pooling. Our objective was to assess whether a straightforward combination of max and average pooling could match or surpass the performance of our proposed quantile pooling method. The combination takes place through division of the dimensionality and subsequent concatenation of the outputs. The results demonstrate that while mixing max and average pooling does not enhance performance beyond that achieved with quantile pooling, integrating quantile pooling with either max or average pooling significantly boosts performance. This indicates that quantile pooling imparts unique and advantageous properties not attainable through mere combinations of max and average pooling. All pooling methods retain unique advantages and can often complement each other, enhancing the overall performance of the model by leveraging their distinct strengths.

Table 5. Comparison of mix pooling

POOLING	SORTING	ENVELOPES	POINT CLOUD
AVG	0.7336 ± 0.0056	0.4313 ± 0.0029	0.9322 ± 0.0025
MAX	0.4016 ± 0.0045	0.4527 ± 0.0021	0.9345 ± 0.0010
AVG + MAX	0.7270 ± 0.0100	0.4593 ± 0.0028	0.9352 ± 0.0012
QUANT	0.7419 ± 0.0034	0.4624 ± 0.0021	$\mathbf{0.9369 \pm 0.0012}$
QUANT + AVG	$\mathbf{0.8089 \pm 0.0049}$	0.4583 ± 0.0024	0.9338 ± 0.0023
QUANT + MAX	0.7170 ± 0.0042	$\mathbf{0.4651 \pm 0.0034}$	0.9366 ± 0.0005

A.11. Experiment Details

A.11.1. Sample Statistic Regression

We formulate a simple regression problem to demonstrate the effectiveness of quantile pooling. To construct the dataset, we first sample n integers from a uniform distribution over $\mathbb{N}$ ($X \sim \mathcal{U}\{0, 1, 2, \ldots, 255\}$). We always sample $n = 32$ integers, without replacement. We then convert the set to one-hot encoding, such that each integer $x_i \in X$ is represented by a vector of length 256, with the x_i-th element being 1 and all other elements being 0. This encoding avoids trivial solutions by simply learning the identity function.

Number of residual layers is set to 3, with a hidden dimension of 64. We use a batch size of 1000 so that after 2000 training steps, the model has seen 2 million samples. We use the AdamW optimizer (Loshchilov & Hutter, 2017), which is a modification of the Adam optimizer (Kingma & Ba, 2014) that decouples the weight decay from the gradient updates., with a learning rate of 0.0001 and weight decay of 0.1. Learning rate is decayed by a factor of 0.5 every 600 steps.

A.11.2. Sorting Clusters

We first construct a multiset X by sampling n integers from a uniform distribution over $\mathbb{N}$. Repeated elements are included to ensure that the multiset forms small clusters. We sample $n = 32$ integers from $\mathcal{U}\{0, 1, 2, 3\}$. We then perturb the multiset by adding a uniform noise to each element, with each integer $x_i \in X$ subjected to a uniform perturbation $\epsilon_i \sim U(-0.1, 0.1)$, such that the perturbed multiset $X' = \{x_i + \epsilon_i | x_i \in X\}$.

As this task is more challenging, we increase the hidden dimension of the residual layers to 128. The rest of the training configurations are the same as in Appendix A.11.1.

A.11.3. Russian-dollable Envelopes

Algorithm 1 Maximum Envelopes

Input: a list of envelopes with dimensions (w, h)
Output: the maximum number of envelopes that can be nested
Ensure each envelope is represented by a pair of width and height (w, h)
Sort the envelopes by width w in ascending order and then by height h in descending order
Initialize an empty list res
for each height h of the sorted envelopes **do**
  Find the position p to insert h in res, keeping res in sorted order
  if p is equal to the size of res **then**
    Append h to the end of res
  else
    Replace the element at index p in res with h
  end if
end for
return the size of res

We construct a dataset of Russian-dollable envelopes by sampling n pairs of numbers from a uniform distribution over $\mathbb{R}^2$. We sample $n = 32$ pairs of numbers from a uniform distribution over the interval $[0, 1)^2$. Ground truth labels are computed using Algorithm 1.

This task is found to be so much harder that we increase the hidden dimension to 256 and the number of residual layers to 6 to get a smooth and stable training curve. The rest of the training configurations are the same as in Appendix A.11.1.

A.11.4. Mixtures of Gaussians

This experiment follows the same setting as in (Lee et al., 2019), except that we fix the number of samples to 300 and we use a batch size of 64. We use 3 residual layers in the Stacked Deep Sets architecture, with a hidden dimension of 256, same as the Set Transformer architecture. Remarkably, we find using batch normalization is detrimental to the performance, and increasing the number of layers does not help on this task. This suggests that there is an identity function as a component in the solution space, and the trick is to use it to capture the mean of the data. Batch normalization obviously destroys this

identity function, and increasing the number of layers makes the function harder to find.

A.11.5. Point Cloud Classification

We use the same setting as in (Ma et al., 2022). We report the results over 6 runs. For the Stacked Deep Sets architecture, we use 6 residual layers with a hidden dimension of 512, the same as Set Transformer.