NEURAL SORTING NETWORKS WITH ERROR-FREE DIFFERENTIABLE SWAP FUNCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sorting is a fundamental operation of all computer systems, having been a long-standing significant research topic. Beyond the problem formulation of traditional sorting algorithms, we consider sorting problems for more abstract yet expressive inputs e.g., multi-digit images and image fragments, through a neural sorting network. To learn a mapping from a high-dimensional input to an ordinal variable, the differentiability of sorting networks needs to be guaranteed. In this paper we define a softening error by a differentiable swap function, and develop an error-free swap function that holds non-decreasing and differentiability conditions. Furthermore, a permutation-equivariant Transformer network with multi-head attention is adopted to capture dependency between given inputs and also leverage its model capacity. Experiments on diverse sorting benchmarks demonstrate that our method performs better than or comparable to existing baseline methods.

1 INTRODUCTION

Traditional sorting algorithms (Cormen et al., 2022), e.g., bubble sort, insertion sort, and quick sort, are a well-established approach to arranging given instances in computer science. Since such a sorting algorithm is a basic component to build diverse computer systems, it has been a long-standing significant research area in science and engineering, and the sorting networks (Knuth, 1998; Ajtai et al., 1983), which are structurally designed as an abstract device with a fixed number of wires, have been widely used to perform a sorting algorithm on computing hardwares.

Formally, given an unordered sequence $\mathbf{s} = [s_1, \ldots, s_n] \in \mathbb{R}^n$, we sort $\mathbf{s}$ to an ordered sequence $\mathbf{s}_\mathrm{o}$:

$$\mathbf{s}_\mathrm{o} = \mathbf{P}^\top \mathbf{s}, \tag{1}$$

where a permutation matrix $\mathbf{P} \in \{0, 1\}^{n \times n}$ allows us to transform the sequence in ascending order. Our goal of solving (1) is to compute $\mathbf{P}$ by considering $\mathbf{s}$: $\mathbf{P} = f(\mathbf{s})$, where f is a sorting algorithm. Beyond the formulation of traditional sorting algorithms, the sorting algorithm can be extended to solving a formulation for sorting more abstract and expressive inputs (e.g., multi-digit images and image fragments), each of which contains ordinal information semantically:

$$\mathbf{X}_\mathrm{o} = \mathbf{P}^\top \mathbf{X}, \tag{2}$$

where $\mathbf{X}_\mathrm{o}$ and $\mathbf{X}$ are ordered and unordered inputs, respectively. Note that $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times d}$, where d is an input dimensionality. Let us assume that a mapping g maps an input $\mathbf{x} \in \mathbb{R}^d$ to an ordinal variable $s \in \mathbb{R}$. If g is unclear, computing $\mathbf{P}$ in (2) is more challenging than in (1) because $\mathbf{x}$ is often a highly implicative high-dimensional variable. To find a generalized mapping g where a ground-truth set $\{(\mathbf{X}_\mathrm{o}^{(i)}, \mathbf{P}_\mathrm{gt}^{(i)}, \mathbf{X}^{(i)})\}_{i=1}^N$ is given as training examples, it requires an effective training scheme. In particular, to leverage a gradient-based learning scheme for the adequate mapping, $f([g(\mathbf{x}_1), \ldots, g(\mathbf{x}_n)])$ needs to be differentiable, which is not the case in general. There has been recent research (Grover et al., 2019; Cuturi et al., 2019; Blondel et al., 2020; Petersen et al., 2021; 2022) to tackle the differentiability issue of such a composite function.

In this paper, following a sorting network-based sorting algorithm with differentiable swap functions (DSFs) (Petersen et al., 2021; 2022), we first define a softening error by a sorting network, which indicates a difference between original and smoothed elements. Then, we propose an error-free DSF and develop the sorting network with error-free DSFs and permutation-equivariant networks.

Specifically, to resolve an error accumulation problem that is induced from a soft DSF, our error-free DSF allows us to guarantee a zero error where $\mathbf{X}$ is properly represented as $\mathbf{s}$. In addition, a permutation-equivariant network with multi-head attention (Vaswani et al., 2017) is adopted to capture dependency between high-dimensional inputs and also leverage the model capacity of the neural network, instead of an instance-wise convolutional neural network (CNN).

Before introducing the main sections, we summarize our contributions:

(i) We define a softening error that measures a difference between original and smoothed values;

(ii) We propose an error-free DSF that resolves the error accumulation problem of conventional DSFs and is still differentiable;

(iii) We adopt a permutation-equivariant network with multi-head attention as a mapping from inputs to ordinal variables $g(\mathbf{X})$, unlike $g(\mathbf{x})$;

(iv) We demonstrate that our proposed method is effective in diverse sorting benchmarks, compared to existing baseline methods.

We will make our codes publicly available upon publication.

2 Sorting Networks with Differentiable Swap Functions

Following traditional sorting algorithms such as bubble sort, quick sort, and merge sort (Cormen et al., 2022) and sorting networks that are constructed by a fixed number of wires (Knuth, 1998; Ajtai et al., 1983), a swap function is a key ingredient of sorting algorithms and sorting networks:

$$(x', y') = \mathrm{swap}(x, y), \tag{3}$$

where $x' = \min(x, y)$ and $y' = \max(x, y)$, which makes the order of x and y correct. For example, if $x > y$, then $x' = y$ and $y' = x$. Without loss of generality, we can express $\min(\cdot, \cdot)$ and $\max(\cdot, \cdot)$ to the following equations:

$$\min(x, y) = x\lfloor\sigma(y - x)\rceil + y\lfloor\sigma(x - y)\rceil \quad \text{and} \quad \max(x, y) = x\lfloor\sigma(x - y)\rceil + y\lfloor\sigma(y - x)\rceil, \tag{4}$$

where $\lfloor\cdot\rceil$ rounds to the nearest integer and $\sigma(\cdot) \in [0, 1]$ transforms an input to a bounded value, i.e., a probability over the input. Computing (4) is straightforward, but they are not differentiable. To enable to differentiate a swap function, the soft versions of $\min$ and $\max$ are defined:

$$\overline{\min}(x, y) = x\sigma(y - x) + y\sigma(x - y) \quad \text{and} \quad \overline{\max}(x, y) = x\sigma(x - y) + y\sigma(y - x), \tag{5}$$

where $\sigma(\cdot)$ is differentiable. In addition to its differentiability, a sigmoid function $\sigma(x)$, i.e., a s-shaped function, satisfies the following properties that (i) $\sigma(x)$ is non-decreasing, (ii) $\sigma(x) = 1$ if $x \to \infty$, (iii) $\sigma(x) = 0$ if $x \to -\infty$, and (iv) $\sigma(0) = 0.5$. Moreover, as discussed in the reference (Petersen et al., 2022), the choice of σ affects the performance of neural network-based sorting network in theory as well as in practice. For example, an optimal monotonic sigmoid function, which is visualized in Figure 4, is defined as

$$\sigma_{\mathcal{O}}(x) = \begin{cases} -\frac{1}{16}(\beta x)^{-1} & \text{if } \beta x < -0.25, \\ 1 - \frac{1}{16}(\beta x)^{-1} & \text{if } \beta x > 0.25, \\ \beta x + 0.5 & \text{otherwise}, \end{cases} \tag{6}$$

where β is steepness; see the work (Petersen et al., 2022) for the details of these numerical and theoretical analyses. Here, we would like to emphasize that the important point of such monotonic sigmoid functions, raised in the reference by Petersen et al. (2022), is the strict monotonicity of sigmoid functions. However, as will be discussed in the next section, it induces an error accumulation problem, which degrades the performance of the sorting network.

By either (4) or (5), the permutation matrix $\mathbf{P}$ (henceforth, denoted as $\mathbf{P}_{\mathrm{hard}}$ or $\mathbf{P}_{\mathrm{soft}}$ for (4) or (5), respectively) is computed by the procedure of sorting network:

(i) Building a pre-defined sorting network with a fixed number of wires – a wire is a component for comparing and swapping two elements;

(ii) Feeding an unordered sequence $\mathbf{s}$ into the pre-defined sorting network and computing a wire-wise permutation matrix $\mathbf{P}_i$ for each wire i iteratively;

(iii) Computing the permutation matrix $\mathbf{P}$ by multiplying all the wire-wise permutation matrices.

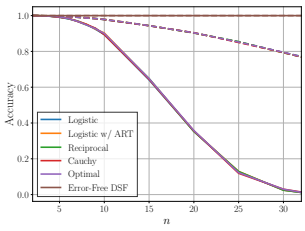

Figure 1: Accuracy, i.e., $\mathrm{acc}_{\mathrm{em}}$ (solid) and $\mathrm{acc}_{\mathrm{ew}}$ (dashed), versus sequence lengths. Each element is uniformly sampled from $[-10, 10]$, and we repeat each experiment set 10,000 times.

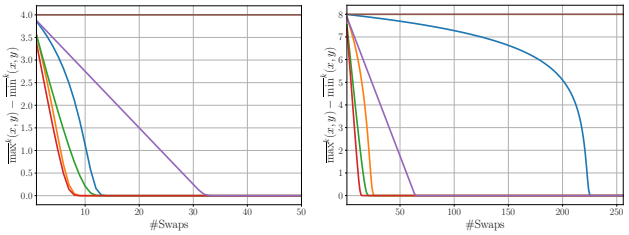

Figure 2: Comparisons to diverse DSFs in terms of how many times a swap function is conducted. Our error-free DSF does not change the original x and y, compared to other DSFs. We set *(left)* $x = 4, y = 0$ or *(right)* $x = 8, y = 0$ initially. We conduct a swap functions k times where $k =$ #Swaps. Legends are the same as the legend presented in Figure 1.

As shown in Figure 3, a set of wires is operated simultaneously, so that each set produces an intermediate permutation matrix $\mathbf{P}_i$ at ith step. Consequently, $\mathbf{P}^\top = \mathbf{P}_1^\top \mathbf{P}_2^\top \cdots \mathbf{P}_k^\top = (\mathbf{P}_k \cdots \mathbf{P}_2 \mathbf{P}_1)^\top$ where k is the number of wire sets, e.g., in Figure 3, $k = 5$.

The doubly-stochastic matrix property of $\mathbf{P}$ is shown by the following proposition:

Proposition 1 (Modification of Lemma 3 in (Petersen et al., 2022))**.** *A permutation matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$ is doubly-stochastic, which implies that $\sum_{i=1}^n [\mathbf{P}]_{ij} = 1$ and $\sum_{j=1}^n [\mathbf{P}]_{ij} = 1$. In particular, regardless of defining a swap function with* min*,* max*,* $\overline{\min}$*, and* $\overline{\max}$*, hard and soft permutation matrices, i.e.,* $\mathbf{P}_{\mathrm{hard}}$ *and* $\mathbf{P}_{\mathrm{soft}}$*, are doubly-stochastic.*

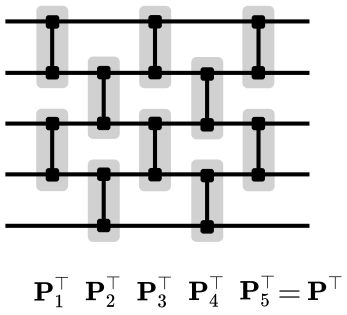

Figure 3: A sorting network

Proof. The proof of this proposition is presented in Section A to satisfy a page limit. □

In Sections 3 and 4, we present an error-free DSF and a neural sorting network with error-free DSFs.

3 Error-Free Differentiable Swap Functions

Before introducing our error-free DSF, we start by describing the motivation of the error-free DSF.

Due to the nature of $\overline{\min}$ and $\overline{\max}$, described in (5), the monotonic DSF changes original input values. For example, if $x < y$, then $x < \overline{\min}(x, y)$ and $\overline{\max}(x, y) < y$ after applying the swap function. It is a serious problem because a change by the DSF is accumulated as the DSF applies iteratively, which is called an error accumulation problem in this paper. As shown in Figure 2, the sigmoid functions such as logistic, logistic with ART, reciprocal, Cauchy, and optimal monotonic functions suffer from this error accumulation problem; see (Petersen et al., 2022) for the details of the respective sigmoid functions. For the case of the Cauchy function, two values are close enough at 9th step in the left panel of Figure 2 and 15th step in the right panel of Figure 2; we present the corresponding step where the difference between two values becomes smaller than 0.001. Here we formally define a softening error, which has been mentioned in this paragraph:

Definition 1. *Suppose that we are given x and y where $x < y$. By* (5)*, these values x and y are softened by a monotonic DSF and they satisfy the following inequalities:*

$$x < x' = \overline{\min}(x, y) \leq y' = \overline{\max}(x, y) < y. \tag{7}$$

Therefore, we define the difference between the original and the softened values, $x' - x$ or $y - y'$:

$$y - y' = y - \overline{\max}(x, y) = x' - x = \overline{\min}(x, y) - x > 0, \tag{8}$$

which is called a softening error in this paper. Without loss of generality, the softening error is $\overline{\min}(x, y) - \min(x, y)$ or $\max(x, y) - \overline{\max}(x, y)$ for any x, y.

Note that (8) is satisfied by $y - \overline{\max}(x, y) = y(1 - \sigma(y - x)) - x\sigma(x - y) = y\sigma(x - y) - x(1 - \sigma(y - x)) = \overline{\min}(x, y) - x$, using (5) and $\sigma(x - y) = 1 - \sigma(y - x)$.

With Definition 1, we are able to specify the seriousness of the error accumulation problem:

Proposition 2. *Suppose that x and y are given and a DSF is applied k times. If $k \to \infty$, error accumulation is $(\max(x, y) - \min(x, y))/2$, under the assumption that $\nabla_x \sigma(x) > 0$.*

Proof. Let $\overline{\min}^k(x, y)$ and $\overline{\min}^k(x, y)$ be minimum and maximum values that swap with $\overline{\min}$ and $\overline{\max}$ is applied k times. By Definition 1, the following inequality is satisfied:

$$\min(x, y) < \overline{\min}^1(x, y) < \cdots < \overline{\min}^k(x, y) \leq \overline{\max}^k(x, y) < \cdots < \overline{\max}^1(x, y) < \max(x, y), \tag{9}$$

under the assumption that $\nabla_x \sigma(x) > 0$. If $k \to \infty$, $\overline{\min}^k(x, y) = \overline{\max}^k(x, y)$. Therefore, a softening error is $(\max(x, y) - \min(x, y))/2$ since $\overline{\max}^k(x, y) = (\min(x, y) + \max(x, y))/2$ by (8). Note that the assumption $\nabla_x \sigma(x) > 0$ implies that $\sigma(\cdot)$ is a strictly monotonic sigmoid function. □

As mentioned in the proof of Proposition 2 and also empirically shown in Figure 2, a swap function with monotonic functions that $\nabla_x \sigma(x)$ is relatively large in general changes the original values x, y significantly compared to one with monotonic functions that $\nabla_x \sigma(x)$ is relatively small – eventually, they become identical quickly in the case of large $\nabla_x \sigma(x)$.

In addition to the error accumulation problem, as presented in Figure 4, such a DSF depends on the scale of $x - y$. If $x < y$ but x and y are close enough, $\sigma(y - x)$ is between 0.5 to 1, which implies that the error can be induced by the scale of $x - y$ (or $y - x$) as well.

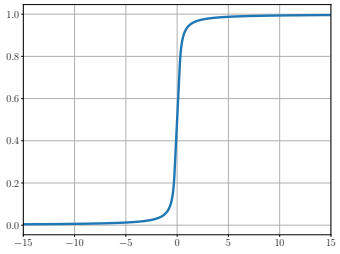

Figure 4: An optimal monotonic sigmoid function

To tackle the aforementioned problem of error accumulation, we propose an error-free DSF, $(x', y') = \text{swap}_{\text{error-free}}(x, y)$:

$$x' = \left(\min(x, y) - \overline{\min}(x, y)\right)_{\text{sg}} + \overline{\min}(x, y), \tag{10}$$

$$y' = \left(\max(x, y) - \overline{\max}(x, y)\right)_{\text{sg}} + \overline{\max}(x, y), \tag{11}$$

where sg indicates that gradients are stopped amid backward propagation, inspired by a straight-through estimator (Bengio et al., 2013). At a step for forward propagation, the error-free DSF produces $x' = \min(x, y)$ and $y' = \max(x, y)$. On the contrary, at a step for backward propagation, the gradients of $\overline{\min}$ and $\overline{\max}$ are used to update learnable parameters. Consequently, our error-free DSF does not smooth the original elements as shown in Figure 2 and ours shows the 100% accuracy for acc_{em} and acc_{ew} (see Section 5 for their definitions) as shown in Figure 1. Compared to our DSF, existing DSFs do not correspond the original elements to the elements that have been compared and fail to achieve reasonable performance as a sequence length increases, in the cases of Figures 1 and 2.

By (4), (5), (10), and (11), we obtain the following equations:

$$\begin{aligned} x' &= \left((x\lfloor\sigma(y-x)\rceil + y\lfloor\sigma(x-y)\rceil) - (x\sigma(y-x) + y\sigma(x-y))\right)_{\text{sg}} + (x\sigma(y-x) + y\sigma(x-y)) \\ &= x\left((\lfloor\sigma(y-x)\rceil - \sigma(y-x))_{\text{sg}} + \sigma(y-x)\right) + y\left((\lfloor\sigma(x-y)\rceil - \sigma(x-y))_{\text{sg}} + \sigma(x-y)\right), \quad (12) \\ y' &= x\left((\lfloor\sigma(x-y)\rceil - \sigma(x-y))_{\text{sg}} + \sigma(x-y)\right) + y\left((\lfloor\sigma(y-x)\rceil - \sigma(y-x))_{\text{sg}} + \sigma(y-x)\right), \quad (13) \end{aligned}$$

which can be used to define a permutation matrix with the error-free DSF. For example, if $n = 2$, a permutation matrix $\mathbf{P}$ over $[x, y]$ is

$$\mathbf{P} = \begin{bmatrix} (\lfloor\sigma(y-x)\rceil - \sigma(y-x))_{\text{sg}} + \sigma(y-x) & (\lfloor\sigma(x-y)\rceil - \sigma(x-y))_{\text{sg}} + \sigma(x-y) \\ (\lfloor\sigma(x-y)\rceil - \sigma(x-y))_{\text{sg}} + \sigma(x-y) & (\lfloor\sigma(y-x)\rceil - \sigma(y-x))_{\text{sg}} + \sigma(y-x) \end{bmatrix}. \tag{14}$$

To sum up, we can describe the following proposition on our error-free DSF, $\text{swap}_{\text{error-free}}(\cdot, \cdot)$:

Proposition 3. *By* (10) *(or* (11)*), the softening error $x' - \min(x, y)$ (or $\max(x, y) - y'$) for an error-free DSF is zero.*

Proof. The proof of this proposition can be found in Section B. □

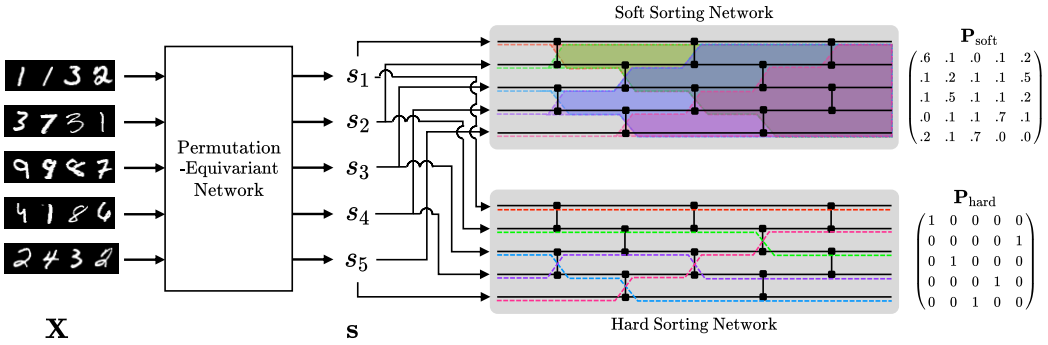

Figure 5: Illustration of our neural sorting network with error-free DSFs. Given high-dimensional inputs $\mathbf{X}$, a permutation-equivariant network produces a vector of ordinal variables $\mathbf{s}$, which is used to be swapped using soft or hard sorting network.

4 Neural Sorting Networks with Error-Free Differentiable Swap Functions

We build our neural network-based sorting network with the error-free DSF and a permutation-equivariant neural network, considering the properties covered in Section 3.

First, we describe a procedure for transforming a high-dimensional input to an ordinal score. Such a mapping $g : \mathbb{R}^d \to \mathbb{R}$, which consists of a set of learnable parameters, has to satisfy a permutation-equivariant property:

$$[g(\mathbf{x}_{\pi_1}), \dots, g(\mathbf{x}_{\pi_n})] = \pi([g(\mathbf{x}_1), \dots, g(\mathbf{x}_n)]), \tag{15}$$

where $\pi_i = [\pi([1, \dots, n])]_i \ \forall i \in [n]$, for any permutation function π. Typically, an instance-wise neural network, which is applied to each element in a sequence given, is permutation-equivariant (Zaheer et al., 2017). Based on this consequence, instance-wise CNNs are employed in differentiable sorting algorithms (Grover et al., 2019; Cuturi et al., 2019; Petersen et al., 2021; 2022). However, such an instance-wise architecture is limited since it is ineffective for capturing essential features from a sequence. Some types of neural networks such as long short-term memory (Hochreiter & Schmidhuber, 1997) and the standard Transformer architecture (Vaswani et al., 2017) are capable of modeling a data sequence, utilizing recurrent connections, scaled dot-product attention, or parameter sharing across elements. While they are powerful for modeling a sequence, they are not obviously permutation-equivariant. Unlike such permutation-variant models, we propose a robust Transformer-based network that satisfies the permutation-equivariant property, inspired by (Lee et al., 2019).

To present our network, we briefly introduce a scaled dot-product attention and multi-head attention:

$$\text{att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_m}}\right)\mathbf{V} \quad \text{and} \quad \text{mha}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1, \dots, \text{head}_h]\mathbf{W}_o, \tag{16}$$

where $\text{head}_i = \text{att}(\mathbf{Q}\mathbf{W}_q^{(i)}, \mathbf{K}\mathbf{W}_k^{(i)}, \mathbf{V}\mathbf{W}_v^{(i)})$, $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times hd_m}$, $\mathbf{W}_q^{(i)}, \mathbf{W}_k^{(i)}, \mathbf{W}_v^{(i)} \in \mathbb{R}^{hd_m \times d_m}$, and $\mathbf{W}_o \in \mathbb{R}^{hd_m \times hd_m}$. Similar to the Transformer network, a series of mha blocks is stacked with layer normalization (Ba et al., 2016) and residual connections (He et al., 2016), and in this paper $\mathbf{X}$ is processed by $\text{mha}(\mathbf{Z}, \mathbf{Z}, \mathbf{Z})$ where $\mathbf{Z} = g'(\mathbf{X})$ or $\mathbf{Z}$ is the output of previous layer; see the appendix for the details of the architecture. Note that $g'(\cdot)$ is an instance-wise embedding layer, e.g., a simple fully-connected network or a simple CNN. Importantly, compared to the standard Transformer model, our network does not include a positional embedding, in order to satisfy the permutation-equivariant property; $\text{mha}(\mathbf{Z}, \mathbf{Z}, \mathbf{Z})$ satisfies (15) for the permutation of $\mathbf{z}_1, \dots, \mathbf{z}_n$ where $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_n]^\top$. The output of our network is $\mathbf{s}$, followed by the last instance-wise fully-connected layer. Finally, as shown in Figure 5, our sorting network is able to produce differentiable permutation matrices over $\mathbf{s}$, $\mathbf{P}_{\text{hard}}$ and $\mathbf{P}_{\text{soft}}$, by (10)/(11) and (5), respectively.

To learn the permutation-equivariant network g, we define both objectives for $\mathbf{P}_{\text{soft}}$ and $\mathbf{P}_{\text{hard}}$:

$$\mathcal{L}_{\text{soft}} = -\sum_{i=1}^{n}\sum_{j=1}^{n}[\mathbf{P}_{\text{gt}} \log \mathbf{P}_{\text{soft}} + (1-\mathbf{P}_{\text{gt}}) \log(1-\mathbf{P}_{\text{soft}})]_{ij}, \tag{17}$$

$$\mathcal{L}_{\text{hard}} = \|\mathbf{P}_{\text{hard}}^{\top}\mathbf{X} - \mathbf{P}_{\text{gt}}^{\top}\mathbf{X}\|_F^2, \tag{18}$$

where $\mathbf{P}_{\text{gt}}$ is a ground-truth permutation matrix. Note that all the operations in $\mathcal{L}_{\text{soft}}$ are entry-wise. Similar to (17), the objective (18) for $\mathbf{P}_{\text{hard}}$ might be designed as the form of binary cross-entropy, which tends to be generally robust in training deep neural networks. However, we struggle to apply the binary cross-entropy for $\mathbf{P}_{\text{hard}}$ into our problem formulation, due to discretized loss values. In particular, the form of cross-entropy for $\mathbf{P}_{\text{hard}}$ can be used to train the sorting network, but degrades its performance in our preliminary experiments. Therefore, we design the objective for $\mathbf{P}_{\text{hard}}$ as $\mathcal{L}_{\text{hard}} = \|\mathbf{P}_{\text{hard}}^{\top}\mathbf{X} - \mathbf{P}_{\text{gt}}^{\top}\mathbf{X}\|_F^2$, which helps us to train the network more robustly.

In addition, using Proposition 4, the objective (18) for $\mathbf{P}_{\text{hard}}$ can be modified by splitting $\mathbf{P}_{\text{hard}}$, $\mathbf{P}_{\text{gt}}$, and $\mathbf{X}$, which is able to reduce the number of possible permutations; please see the associated section for the details of this split strategy. Eventually, our network g is trained by the combined loss $\mathcal{L} = \mathcal{L}_{\text{soft}} + \lambda\mathcal{L}_{\text{hard}}$ where λ is a balancing hyperparameter; the analysis on λ can be found in the appendix. As mentioned above, a landscape of $\mathcal{L}_{\text{hard}}$ is not smooth due to the property of straight-through estimator, even though we use $\mathcal{L}_{\text{hard}}$. Thus, we combine both objectives to the form of a single loss, which is widely adopted in a deep learning community.

5 Experimental Results

We demonstrate the experimental results to show the validity of our method. Our neural network-based sorting network aims to sort two benchmarks: sorting (i) multi-digit images and (ii) image fragments. Unless otherwise specified, an odd-even sorting network is used in the experiments. We measure the performance of each method in acc_{em} and acc_{ew}:

$$\text{acc}_{\text{em}} = \frac{\sum_{i=1}^{N}\bigcap_{j=1}^{n}\mathbf{1}\Big(\Big[\text{argsort}\left(\mathbf{P}_{\text{gt}}^{(i)\top}\mathbf{s}^{(i)}\right)\Big]_j = \left[\text{argsort}\left(\mathbf{P}^{(i)\top}\mathbf{s}^{(i)}\right)\right]_j\Big)}{N}, \tag{19}$$

$$\text{acc}_{\text{ew}} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{n}\mathbf{1}\Big(\Big[\text{argsort}\left(\mathbf{P}_{\text{gt}}^{(i)\top}\mathbf{s}^{(i)}\right)\Big]_j = \left[\text{argsort}\left(\mathbf{P}^{(i)\top}\mathbf{s}^{(i)}\right)\right]_j\Big)}{Nn}, \tag{20}$$

where argsort returns the indices to sort a given vector and $\mathbf{1}(\cdot)$ is an indicator function.

We attempt to match the capacities of the Transformer-based models to the conventional CNNs. As described in Tables 1, 2, and 3, the capacities of the small Transformer-based models are smaller than or similar to the capacities of the CNNs in terms of FLOPs and the number of parameters.

5.1 Sorting Multi-Digit Images

Datasets. We leverage two benchmark datasets to demonstrate the capability of our proposed method. The MNIST dataset (LeCun et al., 1998) is a pertinent dataset for evaluating sorting algorithms. As steadily utilized in the previous work (Grover et al., 2019; Cuturi et al., 2019; Blondel et al., 2020; Petersen et al., 2021; 2022), we create a four-digit dataset by concatenating four images from the MNIST dataset; see Figure 5 for some examples of the dataset. The SVHN dataset (Netzer et al., 2011) also encompasses multi-digit numbers extracted from street view images and is therefore suitable for the sorting task.

Experimental Details. We conduct the experiments 5 times by varying random seeds to report the average of acc_{em} and acc_{ew}, and use an optimal monotonic sigmoid function as DSFs. The performance of each model is measured by a test dataset, by testing the best model determined by a validation dataset. We use the AdamW optimizer (Loshchilov & Hutter, 2018), and train each model for 200,000 steps on the four-digit MNIST dataset and 300,000 steps on the SVHN dataset. Unless otherwise noted, we follow the same value of hyperparameters, e.g., a batch size, from (Petersen et al., 2022) for fair comparisons. The missing details are described in the appendix.

Table 1: Results on sorting four-digit MNIST datasets. The results are measured in $\mathrm{acc}_{\mathrm{em}}$ and $\mathrm{acc}_{\mathrm{ew}}$ (in parentheses). FLOPs is on the basis of a sequence length 3, and Log. and E-F stand for Logistic and Error-Free, respectively. All the values are averaged over 5 runs with different seeds.

Method	Sequence Length 3	5	7	9	15	32	Model	FLOPs	#Param.
NeuralSort	91.9 (94.5)	77.7 (90.1)	61.0 (86.2)	43.4 (82.4)	9.7 (71.6)	0.0 (38.8)	Conv.	130M	855K
Sinkhorn Sort	92.8 (95.0)	81.1 (91.7)	65.6 (88.2)	49.7 (84.7)	12.6 (74.2)	0.0 (41.2)			
Fast Sort & Rank	90.6 (93.5)	71.5 (87.2)	49.7 (81.3)	29.0 (75.2)	2.8 (60.9)	–			
Diffsort Log.	92.0 (94.5)	77.2 (89.8)	54.8 (83.6)	37.2 (79.4)	4.7 (62.3)	0.0 (56.3)			
Diffsort Log. w/ ART	94.3 (96.1)	83.4 (92.6)	71.6 (90.0)	56.3 (86.7)	23.5 (79.4)	0.5 (64.9)			
Diffsort Reciprocal	94.4 (96.1)	85.0 (93.3)	73.4 (90.7)	60.8 (88.1)	30.2 (81.9)	1.0 (66.8)			
Diffsort Cauchy	94.2 (96.0)	84.9 (93.2)	73.3 (90.5)	63.8 (89.1)	31.1 (82.2)	0.8 (63.3)			
Diffsort Optimal	94.6 (96.3)	85.0 (93.3)	73.6 (90.7)	62.2 (88.5)	31.8 (82.3)	1.4 (67.9)			
SortNet with E-F DSFs	94.8 (96.4)	86.9 (94.1)	74.2 (90.9)	62.6 (88.6)	34.7 (83.3)	2.1 (69.2)			
Diffsort Optimal	95.9 (97.1)	90.2 (95.4)	83.9 (94.2)	77.2 (92.9)	57.3 (89.7)	16.3 (81.7)	Transformer-S	130M	665K
SortNet with E-F DSFs	95.9 (97.1)	94.8 (97.5)	90.8 (96.5)	86.9 (95.7)	74.3 (93.6)	37.8 (87.7)			
Diffsort Optimal	96.5 (97.5)	92.6 (96.4)	87.6 (95.3)	82.6 (94.3)	67.8 (92.0)	32.1 (85.7)	Transformer-L	137M	3.104M
SortNet with E-F DSFs	96.5 (97.5)	95.4 (97.7)	92.9 (97.2)	90.1 (96.5)	82.5 (95.0)	46.2 (88.9)			

Table 2: Results on sorting SVHN datasets. FLOPs is computed on the basis of a sequence length 3. All the values are averaged over 5 runs with different seeds.

Method	Sequence Length 3	5	7	9	15	Model	FLOPs	#Param.
Diffsort Log.	76.3 (83.2)	46.0 (72.7)	21.8 (63.9)	13.5 (61.7)	0.3 (45.9)	Conv.	326M	1.226M
Diffsort Log. w/ ART	83.2 (88.1)	64.1 (82.1)	43.8 (76.5)	24.2 (69.6)	2.4 (56.8)			
Diffsort Reciprocal	85.7 (89.8)	68.8 (84.2)	53.3 (80.0)	40.0 (76.3)	13.2 (66.0)			
Diffsort Cauchy	85.5 (89.6)	68.5 (84.1)	52.9 (79.8)	39.9 (75.8)	13.7 (66.0)			
Diffsort Optimal	86.0 (90.0)	67.5 (83.5)	53.1 (80.0)	39.1 (76.0)	13.2 (66.3)			
SortNet with E-F DSFs	86.8 (90.6)	68.9 (84.5)	53.4 (80.4)	40.0 (77.0)	12.0 (65.3)			
Diffsort Optimal	86.5 (90.2)	71.9 (85.4)	60.4 (82.5)	48.0 (79.2)	19.5 (70.5)	Transformer-S	210M	1.223M
SortNet with E-F DSFs	86.6 (90.2)	72.6 (85.7)	62.5 (83.5)	48.6 (79.3)	19.3 (69.6)			
Diffsort Optimal	87.8 (91.1)	75.2 (87.0)	63.8 (83.9)	51.6 (80.4)	23.2 (72.3)	Transformer-L	332M	3.475M
SortNet with E-F DSFs	88.0 (91.2)	74.0 (86.3)	63.9 (83.8)	50.2 (80.1)	21.7 (71.2)			

Results. Tables 1 and 2 show the results of the previous work such as NeuralSort (Grover et al., 2019), Sinkhorn Sort (Cuturi et al., 2019), Fast Sort & Rank (Blondel et al., 2020), and Diffsort (Petersen et al., 2021; 2022), and our method on the MNIST and SVHN datasets, respectively. When we use the conventional CNN as a permutation-equivariant network, our method shows better than or comparable to the previous methods. As we exploit more robust models, e.g., small and large Transformer-based permutation-equivariant models, both the baseline and ours show better results. It is worth noting that the performance gap between the baseline and ours with the error-free DSFs is getting larger when we utilize the Transformer-based models.[1]

5.2 Sorting Image Fragments

Datasets. When inputs are fragments, we use two datasets: the MNIST dataset and the CIFAR-10 dataset (Krizhevsky & Hinton, 2009). Similar to the reference (Mena et al., 2018), we create multiple fragments (or patches) from a single-digit image of the MNIST dataset to utilize themselves as inputs – for example, 4 fragments of size 14×14 or 9 fragments of size 9×9 are created from a single image. Similarly, the CIFAR-10 dataset, which contains various objects (e.g., birds and cats), is split to multiple patches, and then is used to the experiments on sorting image fragments. See Table 3 for the details of fragments we use.

[1]Thanks to many open-source projects, we can easily run the baseline methods. However, it is difficult to reproduce some results due to unknown random seeds. For this reason, we bring the results from the reference (Petersen et al., 2022), and use fixed random seeds, i.e., 42, 84, 126, 168, 210, for our methods.

Table 3: Results on sorting image fragments of MNIST and CIFAR-10. 2×2 and 3×3 indicate the number of fragments, and 14×14, 9×9, 16×16, and 10×10 (in parentheses) indicate the size of image fragments. FLOPs is computed on the basis of the MNIST 2×2 (14×14) case and the CIFAR-10 2×2 (16×16) case. All the values are averaged over 5 runs with different seeds.

Method		**MNIST** 2×2 (14×14)	**MNIST** 3×3 (9×9)	**FLOPs**	**#Param.**	**CIFAR-10** 2×2 (16×16)	**CIFAR-10** 3×3 (10×10)	**FLOPs**	**#Param.**	**Model**
Diffsort	Log.	98.5 (99.0)	5.3 (42.9)	1.498M	84K	56.9 (73.6)	0.8 (27.7)	1.663M	85K	Conv.
	Log. w/ ART	98.4 (99.1)	5.4 (42.9)			56.7 (73.4)	0.7 (27.7)			
	Reciprocal	98.4 (99.2)	5.3 (42.9)			56.7 (73.4)	0.7 (27.8)			
	Cauchy	98.4 (99.2)	5.3 (42.9)			56.9 (73.6)	0.9 (27.9)			
	Optimal	98.4 (99.1)	5.3 (43.0)			56.6 (73.4)	0.7 (27.7)			
SortNet with E-F DSFs		98.4 (99.2)	5.2 (42.6)			56.9 (73.6)	0.8 (28.0)			
Diffsort	Optimal	98.7 (99.3)	5.5 (43.0)	946K	87K	58.0 (74.2)	1.0 (28.4)	1.111M	87K	Transformer
SortNet with E-F DSFs		98.6 (99.2)	5.6 (43.7)			58.1 (74.2)	0.9 (28.3)			

Experimental Details. Similar to the experiments on sorting multi-digit images, an optimal monotonic sigmoid function is used as DSFs. Since the size of inputs is much smaller than the experiments on sorting multi-digit images, which are shown in Section 5.1, we modify the architectures of the CNNs and the Transformer-based models. We reduce the kernel size of convolution layers from five to three and make strides two. Max-pooling operations are removed. Similar to the experiments in Section 5.1, we use the AdamW optimizer (Loshchilov & Hutter, 2018). Moreover, each model is trained for 50,000 steps when the number of fragments is 2×2, i.e., when a sequence length is 4, and 100,000 steps for 3×3 fragments, i.e., when a sequence length is 9. Additional information including the details of neural architectures can be found in the appendix.

Results. Table 3 represents the experimental results on both datasets of image fragments, which are created from the MNIST and CIFAR-10 datasets. Similar to the experiments on sorting multi-digit images in Section 5.1, the more robust model improves the performance of both the baseline and our method, especially in the CIFAR-10 2×2 experiments.

According to the experimental results, we achieve satisfactory performance by applying error-free DSFs, combined loss, and Transformer-based models with multi-head attention. We provide detailed discussion on how they contribute to the performance gain in Section 7, and ablation studies on steepness, learning rate, and a balancing hyperparameter in the appendix.

6 Related Work

We cover the work related to our neural sorting network with error-free DSFs in this section.

Differentiable Sorting Algorithms. To allow us to differentiate a sorting algorithm, Grover et al. (2019) have proposed the continuous relaxation of $\mathrm{argsort}$ operator, which is named NeuralSort. In this work the output of NeuralSort only satisfies the row-stochastic matrix property, although Grover et al. (2019) attempt to employ a gradient-based optimization strategy in learning a neural sorting algorithm. Cuturi et al. (2019) propose the smoothed ranking and sorting operators using optimal transport, which is the natural relaxation for assignments. To reduce the cost of the optimal transport, the Sinkhorn algorithm (Cuturi, 2013) is used. Then, Blondel et al. (2020) have proposed the first differentiable sorting and ranking operators with $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space complexities, which is named Fast Rank & Sort, by constructing differentiable operators as projections on permutahedron. Petersen et al. (2021) have suggested a differentiable sorting network with relaxed conditional swap functions. Recently, the same authors analyze the characteristics of the relaxation of monotonic conditional swap functions, and propose several monotonic swap functions, e.g., Cauchy and optimal monotonic functions (Petersen et al., 2022).

Permutation-Equivariant Networks. A seminal architecture, long short-term memory (Hochreiter & Schmidhuber, 1997) can be used in modeling a sequence without any difficulty, and a sequence-to-sequence model (Sutskever et al., 2014) can be employed to cope with a sequence.

However, as discussed in the work (Vinyals et al., 2016), an unordered sequence can have good orderings, by analyzing the effects of permutation thoroughly. Zaheer et al. (2017) have proposed a permutation-invariant or permutation-equivariant network, named Deep Sets, and proved the permutation invariance and permutation equivariance of the proposed models. By utilizing a Transformer network (Vaswani et al., 2017), Lee et al. (2019) propose a permutation-equivariant network.

7 Discussion & Conclusion

In this section we discuss potential applications, comprehensive analyses of our sorting network, limitations, and future directions, and then conclude our work.

Utilization of Hard Permutation Matrices. While a soft permutation matrix $\mathbf{P}_{\mathrm{soft}}$ is limited to be utilized itself directly, a hard permutation matrix $\mathbf{P}_{\mathrm{hard}}$ is capable of being applied in a case that requires $\mathbf{P}_{\mathrm{hard}}$; each element of $\mathbf{P}_{\mathrm{soft}}^{\top}\mathbf{s}$ is a linear combination of some column of $\mathbf{P}_{\mathrm{soft}}$ and $\mathbf{s}$, but one of $\mathbf{P}_{\mathrm{hard}}^{\top}\mathbf{s}$ is an exact element in $\mathbf{s}$. The experiments demonstrated in Section 5.2 are one of such cases, and it exhibits our strength, not only the performance in $\mathrm{acc}_{\mathrm{em}}$ and $\mathrm{acc}_{\mathrm{ew}}$.

Effects of Multi-Head Attention in the Problem (2)**.** We follow the model architecture, which is used in the previous work (Grover et al., 2019; Cuturi et al., 2019; Petersen et al., 2021; 2022), for the CNNs. However, as shown in Tables 1, 2, and 3, the model is not enough to show the best performance. In particular, whereas the model capacity, i.e., FLOPs and the number of parameters, of the small Transformer models is almost matched to or less than the capacity of the CNNs, the results by the small Transformer models outperform the results by the CNNs. We presume that these performance gains are derived from a multi-head attention's ability to capture long-term dependency and reduce an inductive bias, as widely stated in many recent studies in diverse fields such as natural language processing (Vaswani et al., 2017; Devlin et al., 2018; Brown et al., 2020), computer vision (Dosovitskiy et al., 2021; Liu et al., 2021), and 3D vision (Nash et al., 2020; Zhao et al., 2021). Especially, unlike the instance-wise CNNs, our permutation-equivariant Transformer model utilizes multi-head attentions between instances, so that our model can productively compare instances in a sequence and effectively learn the relative relationship between them.

Analysis on Performance Gains. According to the results in Section 5, we can argue that error-free DSFs, combined loss, and Transformer-based models contribute to better performance appropriately compared to the baseline methods. As shown in Table 1, the performance gains by Transformer-based models are more substantial than the gains by error-free DSFs and combined loss, since multi-head attention is effective in capturing long-term dependency (or dependency between multiple instances in our case) and reducing inductive biases. However, as in the discussion on the utilization of hard permutation matrices, the hard permutation matrices can be used in the case that does not allow us to mix instances in $\mathbf{X}$, e.g., sorting image fragments in Section 5.2.

Limitations. Since our method is based on a sorting network, the ideas in this work are limited to sorting algorithms, which implies that it is not easy to devise a neural network-based approach to solving general problems in computer science, e.g., combinatorial optimization, with our ideas.

Further Study of Differentiable Sorting Algorithms. Differentiable sorting algorithms are empowered by encouraging us to train a mapping from an abstract input to an ordinal score using supervision on permutation matrices. However, this line of studies is limited to a problem of sorting high-dimensional data that contains clear information of orderings, e.g., multi-digit numbers. As the further study of differentiable sorting, we can expand this framework to sorting more ambiguous data, which contains implicitly ordinal information.

In this paper, we define a softening error, which is induced by a monotonic DSF, and show several evidences of the error accumulation problem. To resolve the error accumulation problem, we propose an error-free DSF, inspired by a straight-through estimator. Moreover, we provide the simple theoretical and empirical analyses that our error-free DSF successfully achieves a zero error and also holds non-decreasing and differentiability conditions. By combining all the components, we propose our neural sorting network with the error-free DSF. Finally we demonstrate that our methods are better than or comparable to other baseline algorithms in diverse sorting benchmarks.

REPRODUCIBILITY STATEMENT

We describe the details of experiments and implementation in Sections 5, D, and E. Moreover, the supplementary material includes our implementation and an installation guide. The final configurations, i.e., neural architectures and hyperparameters, of the experiments conducted in this work are presented in the scripts included in the supplementary material.

REFERENCES

M. Ajtai, J. Komlós, and E. Szemerédi. An O(n log n) sorting network. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pp. 1–9, Boston, Massachusetts, USA, 1983.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

M. Blondel, O. Teboul, Q. Berthet, and J. Djolonga. Fast differentiable sorting and ranking. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 950–959, Virtual, 2020.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 1877–1901, Virtual, 2020.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 4 edition, 2022.

M. Cuturi. Sinkhorn distances: lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 26, pp. 2292–2300, Lake Tahoe, Nevada, USA, 2013.

M. Cuturi, O. Teboul, and J.-P. Vert. Differentiable ranking and sorting using optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, Vancouver, British Columbia, Canada, 2019.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Virtual, 2021.

A. Grover, E. Wang, A. Zweig, and S. Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, New Orleans, Louisiana, USA, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, Nevada, USA, 2016.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

D. E. Knuth. *The art of computer programming*, volume 3. Addison-Wesley Professional, 2 edition, 1998.

A. Krizhevsky and G. E. Hinton. Learning multiple layers of features from tiny images. Technical report, Computer Science Department, University of Toronto, 2009.

Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 1998.

J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 3744–3753, Long Beach, California, USA, 2019.

Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin Transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 10012–10022, Virtual, 2021.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Vancouver, British Columbia, Canada, 2018.

G. E. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning latent permutations with Gumbel-Sinkhorn networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Vancouver, British Columbia, Canada, 2018.

C. Nash, Y. Ganin, S. M. A. Eslami, and P. W. Battaglia. PolyGen: An autoregressive generative model of 3D meshes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 7220–7229, Virtual, 2020.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *Neural Information Processing Systems Workshop on Deep Learning and Unsupervised Feature Learning*, Granada, Spain, 2011.

F. Petersen, C. Borgelt, H. Kuehne, and O. Deussen. Differentiable sorting networks for scalable sorting and ranking supervision. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 8546–8555, Virtual, 2021.

F. Petersen, C. Borgelt, H. Kuehne, and O. Deussen. Monotonic differentiable sorting networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Virtual, 2022.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, Montreal, Quebec, Canada, 2014.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 5998–6008, Long Beach, California, USA, 2017.

O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, 2016.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 3391–3401, Long Beach, California, USA, 2017.

H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun. Point transformer. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 16259–16268, Virtual, 2021.

APPENDICES

From now, we describe the contents missing in the main article.

A PROOF OF PROPOSITION 1

Proof. If two elements at indices i and j are swapped by a single swap function, $[\mathbf{P}]_{kk} = 1$ for $k \in [n]\backslash\{i, j\}$, $[\mathbf{P}]_{kl} = 0$ for $k \neq l$, $k, l \in [n]\backslash\{i, j\}$, $[\mathbf{P}]_{ii} = [\mathbf{P}]_{jj} = p$, and $[\mathbf{P}]_{ij} = [\mathbf{P}]_{ji} = 1-p$, where p is the output of sigmoid function, i.e., $p = \sigma(y - x)$ or $p = \lfloor \sigma(y - x) \rceil$. Since the multiplication of doubly-stochastic matrices is still doubly-stochastic, Proposition 1 is true. □

B PROOF OF PROPOSITION 3

Proof. According to Definition 1, given x and y, the softening error $x' - \min(x, y)$ is expressed as

$$\begin{aligned} x' - \min(x, y) &= \big(\min(x, y) - \overline{\min}(x, y)\big)_{\text{sg}} + \overline{\min}(x, y) - \min(x, y) \\ &= \min(x, y) - \overline{\min}(x, y) + \overline{\min}(x, y) - \min(x, y) = 0, \end{aligned} \tag{21}$$

while a forward pass is applied. The proof for $\max(x, y) - y'$ is omitted because it is obvious. □

C SPLIT STRATEGY TO REDUCE THE NUMBER OF POSSIBLE PERMUTATIONS

As presented in (14) and Proposition 1, the permutation matrix for the error-free DSF is a discretized doubly-stochastic matrix, which can be denoted as $\mathbf{P}_{\text{hard}}$, in a forward pass, and is differentiable in a backward pass. Here, we show an interesting proposition of $\mathbf{P}_{\text{hard}}$:

Proposition 4. *Let $\mathbf{s} \in \mathbb{R}^n$ and $\mathbf{P}_{\text{hard}} \in \mathbb{R}^{n \times n}$ be an unordered sequence and its corresponding permutation matrix to transform to $\mathbf{s}_{\text{o}}$, respectively. We are able to split $\mathbf{s}$ to two subsequences $\mathbf{s}_1 \in \mathbb{R}^{n_1}$ and $\mathbf{s}_2 \in \mathbb{R}^{n_2}$ where $\mathbf{s}_1 = [\mathbf{s}]_{1:n_1}$ and $\mathbf{s}_2 = [\mathbf{s}]_{1:n_2}$. Then, $\mathbf{P}_{\text{hard}}$ is also split to $\mathbf{P}_1 \in \mathbb{R}^{n_1 \times n_1}$ and $\mathbf{P}_2 \in \mathbb{R}^{n_2 \times n_2}$, so that $\mathbf{P}_1$ and $\mathbf{P}_2$ are (discretized) doubly-stochastic.*

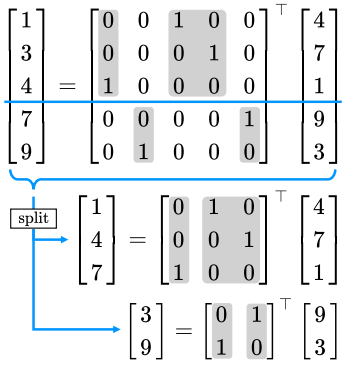

Figure 6: A split process of $\mathbf{P}$

Proof. Since a split does not change the relative order of elements in the same split and entries in the permutation matrix is zero or one, a permutation matrix can be split as shown in Figure 6. Moreover, multiple splits are straightforwardly available. □

In contrast to $\mathbf{P}_{\text{hard}}$, it is impossible to split $\mathbf{P}_{\text{soft}}$ to sub-block matrices since such sub-block matrices cannot satisfy the property of doubly-stochastic matrix, which is discussed in Proposition 1. Importantly, Proposition 4 does not show a possibility of the recoverable decomposition of the permutation matrix, which implies that we cannot guarantee the recovery of decomposed matrices to the original matrix. Regardless of the existence of recoverable decomposition, we intend to reduce the number of possible permutations with sub-block matrices, rather than holding the large number of possible permutations with the original permutation matrix. Therefore, by Proposition 4, relative relationships between instances with a smaller number of possible permutations are more distinctively learnable than the relationships with a larger number of possible permutations, preventing a sparse correct permutation among a large number of possible permutations.

D DETAILS OF ARCHITECTURES

We describe the details of the neural architectures used in our paper, as shown in Tables 4, 5, 6, 7, 8, 9, 10, 11, 12, and 13. For the experiments of image fragments, we omit some of the architectures with particular fragmentation, since they follow the same architectures presented in Tables 10, 11, 12, and 13. Only differences are the size of inputs, and therefore the respective sizes of the first fully-connected layers are changed.

Table 4: Architecture of the convolutional neural networks for the four-digit MNIST dataset

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	1×32	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	32×64	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Fully-connected	12544×64	–	–
ReLU	–	–	–
Fully-connected	64×1	–	–

Table 5: Architecture of the Transformer-Small models for the four-digit MNIST dataset

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	1×32	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	32×64	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Fully-connected	12544×16	–	–
Transformer Encoder	16×16		#layers 6, #heads 8
ReLU	–	–	–
Fully-connected	16×1	–	–

Table 6: Architecture of the Transformer-Large models for the four-digit MNIST dataset

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	1×32	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	32×64	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Fully-connected	12544×64	–	–
Transformer Encoder	64×64		#layers 8, #heads 8
ReLU	–	–	–
Fully-connected	64×1	–	–

Table 7: Architecture of the convolutional neural networks for the SVHN dataset

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	3×32	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	32×64	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	64×128	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	128×256	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Fully-connected	2304×64	–	–
ReLU	–	–	–
Fully-connected	64×1	–	–

Table 8: Architecture of the Transformer-Small models for the SVHN dataset

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	3×32	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	32×64	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	64×64	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	64×128	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Fully-connected	1152×32	–	–
Transformer Encoder	32×32		#layers 6, #heads 8
ReLU	–	–	–
Fully-connected	32×1	–	–

Table 9: Architecture of the Transformer-Large models for the SVHN dataset

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	3×32	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	32×64	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	64×128	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Convolutional	128×256	5×5	strides 1, padding 2
ReLU	–	–	–
Max-pooling	–	–	pooling 2, strides 2
Fully-connected	2304×64	–	–
Transformer Encoder	64×64		#layers 8, #heads 8
ReLU	–	–	–
Fully-connected	64×1	–	–

Table 10: Architecture of the convolutional neural networks for the MNIST dataset of 4 image fragments of size 14×14

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	1×32	3×3	strides 2, padding 1
ReLU	–	–	–
Convolutional	32×64	3×3	strides 2, padding 1
ReLU	–	–	–
Fully-connected	1024×64	–	–
ReLU	–	–	–
Fully-connected	64×1	–	–

Table 11: Architecture of the Transformer models for the MNIST dataset of 4 image fragments of size 14×14

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	1×32	3×3	strides 2, padding 1
ReLU	–	–	–
Convolutional	32×32	3×3	strides 2, padding 1
ReLU	–	–	–
Fully-connected	512×16	–	–
Transformer Encoder	16×16		#layers 1, #heads 8
ReLU	–	–	–
Fully-connected	16×1	–	–

Table 12: Architecture of the convolutional neural networks for the CIFAR-10 dataset of 4 image fragments of size 16×16

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	3×32	3×3	strides 2, padding 1
ReLU	–	–	–
Convolutional	32×64	3×3	strides 2, padding 1
ReLU	–	–	–
Fully-connected	1024×64	–	–
ReLU	–	–	–
Fully-connected	64×1	–	–

Table 13: Architecture of the Transformer models for the CIFAR-10 dataset of 4 image fragments of size 16×16

Layer	Input & Output (Channel) Dimensions	Kernel Size	Details
Convolutional	3×32	3×3	strides 2, padding 1
ReLU	–	–	–
Convolutional	32×32	3×3	strides 2, padding 1
ReLU	–	–	–
Fully-connected	512×16	–	–
Transformer Encoder	16×16		#layers 1, #heads 8
ReLU	–	–	–
Fully-connected	16×1	–	–

E Details of Experiments

As described in the main article, we use three public datasets: MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011), and CIFAR-10 (Krizhevsky & Hinton, 2009). Unless otherwise specified, the learning rate $10^{-3.5}$ is used for the CNN architectures and the learning rate 10^{-4} is used for the Transformer-based architectures; see the scripts for the exact learning rates we use in the experiments. Learning rate decay is applied to multiply 0.5 in every 50,000 steps for the experiments on sorting multi-digit images and every 20,000 steps for the experiments on sorting image fragments. Moreover, we balance two objectives for $\mathbf{P}_{\mathrm{hard}}$ and $\mathbf{P}_{\mathrm{soft}}$ by multiplying 1, 0.1, 0.01, or 0.001; see the scripts included in our implementation for the respective values for all the experiments. For random seeds, we pick five random seeds for all the experiments: 42, 84, 126, 168, and 210; these values are picked without any trials. Other missing details can be found in our implementation, which is included in the supplementary material.

F Ablation Study on Steepness and Learning Rate

We present ablation studies on steepness and learning rate for the experiments on sorting the multi-digit MNIST dataset, as shown in Tables 14, 15, 16, 17, 18, and 19. For these experiments, a random seed 42 is only used due to numerous experimental settings. Also, we use balancing hyperparameters λ as 1.0, 1.0, 0.1, 0.1, 0.1, and 0.1 for sequence lengths 3, 5, 7, 9, 15, and 32, respectively. Since there are many configurations of steepness, learning rate, and a balancing hyperparameter, we cannot include all the configurations here. The final configurations we use in the experiments are described in the supplementary material.

Table 14: Ablation study on steepness and learning rate for a sequence length 3. lr stands for learning rate.

$\log_{10}$ lr	Steepness 2	4	6	8	10	12	14
-4.0	89.2 (92.6)	91.5 (94.2)	92.1 (94.6)	92.3 (94.7)	92.7 (95.0)	92.1 (94.6)	92.7 (95.0)
-3.5	93.6 (95.5)	93.4 (95.5)	94.5 (96.2)	93.9 (95.9)	94.4 (96.1)	94.5 (96.2)	94.6 (96.3)
-3.0	95.3 (96.8)	95.1 (96.6)	95.0 (96.5)	94.4 (96.2)	94.3 (96.1)	94.7 (96.4)	94.8 (96.5)
-2.5	94.5 (96.2)	94.3 (96.1)	93.7 (95.6)	93.8 (95.7)	93.7 (95.7)	93.7 (95.6)	94.1 (95.9)

Table 15: Ablation study on steepness and learning rate for a sequence length 5. lr stands for learning rate.

$\log_{10}$ lr	Steepness 14	16	18	20	22	24	26
-4.0	78.3 (90.2)	76.3 (89.3)	79.5 (90.9)	78.7 (90.4)	78.4 (90.4)	77.3 (89.8)	79.0 (90.6)
-3.5	85.1 (93.3)	83.6 (92.7)	84.8 (93.1)	85.5 (93.5)	83.4 (92.5)	85.6 (93.6)	84.1 (92.8)
-3.0	86.9 (94.1)	85.2 (93.3)	86.2 (93.8)	85.7 (93.6)	85.4 (93.5)	85.0 (93.2)	85.0 (93.3)
-2.5	83.1 (92.3)	83.2 (92.4)	83.1 (92.4)	81.7 (91.8)	83.3 (92.5)	82.4 (92.1)	82.5 (92.1)

Table 16: Ablation study on steepness and learning rate for a sequence length 7. lr stands for learning rate.

$\log_{10}$ lr	Steepness 23	25	27	29	31	33	35
-4.0	53.5 (82.9)	57.3 (84.6)	61.0 (86.0)	58.3 (85.1)	66.1 (88.0)	57.3 (84.6)	61.6 (86.4)
-3.5	71.6 (90.0)	73.5 (90.8)	68.1 (88.6)	72.9 (90.5)	72.2 (90.2)	71.4 (89.9)	74.2 (91.0)
-3.0	74.4 (90.9)	72.8 (90.4)	74.2 (90.8)	69.2 (89.0)	69.9 (89.3)	73.0 (90.4)	73.0 (90.5)
-2.5	68.0 (88.6)	67.2 (88.1)	68.1 (88.6)	64.5 (86.9)	66.8 (88.0)	70.4 (89.3)	66.2 (87.9)

Table 17: Ablation study on steepness and learning rate for a sequence length 9. lr stands for learning rate.

$\log_{10}$ lr	Steepness 26	28	30	32	34	36	38
-4.0	34.0 (77.7)	37.8 (79.7)	37.3 (79.4)	45.9 (83.0)	46.4 (83.1)	36.2 (78.9)	45.3 (82.5)
-3.5	51.8 (84.8)	59.7 (87.5)	58.3 (87.5)	61.0 (88.2)	60.2 (88.1)	54.6 (85.8)	56.3 (86.6)
-3.0	62.5 (88.8)	60.5 (88.0)	61.8 (88.4)	63.2 (88.9)	61.5 (88.2)	62.2 (88.6)	65.0 (89.5)
-2.5	57.4 (86.8)	58.7 (87.2)	56.5 (86.4)	55.3 (86.1)	52.8 (85.0)	46.9 (82.1)	52.1 (84.6)

Table 18: Ablation study on steepness and learning rate for a sequence length 15. lr stands for learning rate.

$\log_{10}$ lr	Steepness 19	21	23	25	27	29	31
-4.0	3.6 (62.7)	7.2 (67.6)	4.9 (64.8)	2.6 (60.9)	2.9 (61.5)	2.5 (60.2)	6.8 (68.3)
-3.5	10.3 (71.5)	11.9 (73.0)	7.6 (68.6)	22.3 (78.7)	7.1 (68.7)	8.2 (69.6)	10.6 (71.8)
-3.0	24.2 (79.8)	24.1 (80.1)	29.7 (81.9)	32.1 (82.7)	23.9 (79.9)	31.6 (82.1)	31.0 (82.0)
-2.5	30.5 (81.4)	28.2 (80.0)	18.9 (76.8)	29.8 (80.9)	19.8 (77.8)	15.4 (75.4)	24.6 (79.6)

Table 19: Ablation study on steepness and learning rate for a sequence length 32. lr stands for learning rate.

$\log_{10}$ lr	Steepness 118	120	122	124	126	128	130
-4.0	0.0 (46.4)	0.0 (45.8)	0.0 (46.4)	0.0 (43.1)	0.0 (42.9)	0.0 (49.4)	0.0 (48.9)
-3.5	0.2 (60.5)	0.3 (61.8)	0.7 (64.8)	0.3 (62.7)	0.0 (56.0)	0.3 (62.7)	0.2 (59.8)
-3.0	0.6 (63.1)	0.5 (63.6)	0.4 (59.8)	0.8 (65.8)	0.2 (58.1)	0.1 (56.0)	0.1 (57.1)
-2.5	0.5 (62.8)	0.5 (62.3)	0.5 (62.5)	0.1 (58.3)	0.3 (61.2)	0.1 (59.5)	0.2 (58.1)

G Ablation Study on Balancing Hyperparameters

Table 20: Ablation study on a balancing hyperparameter λ

λ	Sequence Length					
	3	5	7	9	15	32
1.000	94.9 (96.5)	86.1 (93.8)	73.7 (90.6)	60.8 (87.9)	10.8 (67.8)	0.0 (31.8)
0.100	94.8 (96.4)	86.4 (93.8)	75.3 (91.2)	65.6 (89.6)	33.7 (83.0)	1.2 (63.0)
0.010	94.9 (96.5)	87.2 (94.2)	76.3 (91.6)	65.6 (89.5)	32.9 (82.8)	0.8 (62.6)
0.001	95.2 (96.7)	86.6 (94.0)	75.3 (91.2)	64.6 (89.1)	34.4 (83.2)	1.4 (66.4)
0.000	94.8 (96.4)	86.6 (93.9)	76.4 (91.7)	64.0 (89.0)	33.0 (82.6)	1.3 (66.0)

We conduct an ablation study on a balancing hyperparameter in the experiments on sorting the four-digit MNIST dataset, as shown in Table 20. For these experiments, we use steepness 6, 20, 29, 32, 25, and 124 for sequence lengths 3, 5, 7, 9, 15, and 32, respectively; these values are obtained from the reference (Petersen et al., 2022). Also, we use a learning rate 10^{-3} and five random seeds 42, 84, 126, 168, and 210, for the experiments in Table 20. As mentioned in Section F, we cannot include all configurations in the appendix, but we present our final settings in the supplementary material.