End-to-end permutation learning with Hungarian algorithm

Anonymous Authors[1]

Abstract

Permutations arise in many machine learning applications such as keypoint matching in computer vision or multiple-object tracking in RADAR signal processing. A fixed-number unrolling of the Sinkhorn iteration used with deep learning is applied to such tasks recently. We show the fact that the *fixed* length-unrolling scheme inevitably has samples that do not converge and propose the direct use of the Hungarian algorithm by using a straight-through estimator instead of the Sinkhorn iteration to avoid this problem. We conduct experiments to evaluate the algorithm's performance and show it to be comparable to or better than that of the conventional Gumbel-Sinkhorn algorithm.

1. Introduction

The bipartite graph matching problem is embedded as a data-association problem in many machine learning applications. In image processing, (visual) simultaneous localization and mapping (SLAM) (Milz et al., 2018) includes a data-association problem between two image pairs (Sarlin et al., 2019), which is known as keypoint matching. For point cloud sensors, such as RADAR or LIDAR, multiple-object tracking (MOT) methods are used to track motions of objects. In MOT, there is also a data-association problem between the previous motion history and the observed point cloud (Kawachi & Suzuki, 2020).

The bipartite graph matching problem is typically solved using non-differentiable discrete algorithms. The solution of this problem is equivalently expressed as a binary permutation matrix, which has two constraints that row-wise and column-wise sum to be 1 (one-to-one constraint). Row-wise or column-wise softmax (Grover et al., 2019; Milan et al., 2017) has been deployed in neural networks for continuous replacements of permutation matrices and comply with one side of the constraints. Although it is relatively easy to satisfy one side of the constraints, it is hard to comply with both sides simultaneously.

Such tasks have recently been tackled by using the Sinkhorn iteration (Cour et al., 2007; Sinkhorn, 1964; Sinkhorn & Knopp, 1967), which is unrolled in end-to-end neural networks (Kawachi & Suzuki, 2020; Mena et al., 2018; Santa Cruz et al., 2017; Sarlin et al., 2019). However, Sinkhorn iteration includes continuous relaxation errors in training and unrolling truncation errors caused by the *fixed* number of iterations. One study involved a variable unrolling number of the Sinkhorn iteration controlled using a temperature hyperparameter (Cuturi et al., 2019). However, they reported that the number of iterations varies and may exceed 100. As we show later, there are strictly positive matrices that can block convergence for an arbitrary number of iterations. Therefore, the variable-length-iteration approach may consume extra time due to a huge number of serial iteration steps.

Furthermore, the Sinkhorn iteration is likely to have a vanishing gradient problem. It is often considered as the matrix version of the softmax function because the Sinkhorn iteration to an exponentiated vector is the same as the softmax function. We show that the softmax function with temperature has a vanishing gradient problem, which implies the Sinkhorn iteration also has this problem.

To avoid these problems, we propose using the Hungarian algorithm (Kuhn, 1955) with a straight-through estimator (STE) (Bengio et al., 2013; Jang et al., 2016). The Hungarian algorithm outputs a correct format permutation matrix and the STE enables the use of a hard solution in neural networks.

In this work, we show that

- conventional Sinkhorn iteration with a *fixed* unrolling number may not converge and emit unnormalized solutions even for strictly positive matrices. We show that there exist some positive matrices that can delay iteration length for convergence arbitrary.

- neural networks with a permutation layer using the Hungarian algorithm (Kuhn, 1955) (Munkres, 1957) which is trained using an STE achieve accurate data association. This algorithm emits exact permutation ma-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

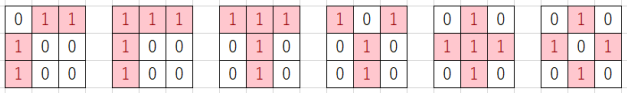


Figure 1. Oscillating Sinkhorn iteration patterns in 3×3 binary matrices. These patterns are not covered by Sinkhorn's theorem, shown in (Sinkhorn, 1964), as they include zeros. Rotational variants are omitted.

trices and avoids the problem of the unrolled Sinkhorn iteration, which has unknown constraint tolerance. We conduct experiments to evaluate the algorithm's performance and found that it is comparable to or better than that of the conventional Gumbel-Sinkhorn algorithm (Mena et al., 2018).

2. Limitation with conventional Sinkhorn iteration

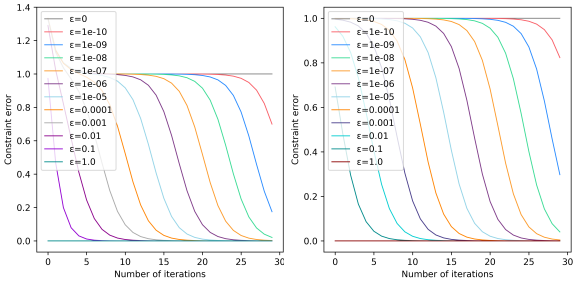
2.1. Non-converging samples

The Sinkhorn iteration, which converts a positive matrix into a doubly stochastic matrix, is widely used as a continuous counterpart of a permutation matrix. The Sinkhorn iteration consists of the iteration of the pair of row-wise normalization $N_r(X) = X \oslash (X\mathbf{1}\mathbf{1}^T)$ and column-wise normalization $N_c(X) = X \oslash (\mathbf{1}\mathbf{1}^T X)$, where X denotes the input matrix, $\oslash$ denotes member-wise division, and $\mathbf{1}$ is the vector that contains the row or column number of 1s. We define the L-step unrolled Sinkhorn operator for a matrix A as $S_l(X) = N_c(N_r(X))$, $S^{(L)}(A) = S_L(S_{L-1}(\cdots S_1(A)\cdots))$, which means an alternate row-wise and column-wise normalization iteration procedure. Usually, the exponent of a real-valued matrix χ is used as the input $A = \exp(\chi)$ to ensure positivity.

The Sinkhorn iteration converges for strictly positive matrix input (Sinkhorn, 1964). However, it often requires infinite iterations to converge. [1] We now give examples that are not suitable for fixed-length unrolling of the Sinkhorn iteration. If the matrix contains zeros, there are some non-converging samples. We show some of the patterns in Fig. 1. Let us introduce a very small positive scalar ϵ to make this matrix positive. Then the matrix must satisfy the Sinkhorn's theorem (Sinkhorn, 1964). We show at least this type of matrix has a non-converging problem with fixed-length iteration. We define the row and column direction constraint errors for a matrix as $\mathrm{Err_c}(A) = \max_j |\mathbf{1} - \sum_i N^{(L)}_{r,i,j}(A)|$ and $\mathrm{Err_r}(A) = \max_i |\mathbf{1} - \sum_j N^{(L)}_{c,i,j}(A)|$, respectively, where $N_r^{(L)}(A) = N_r(S^{(L-1)}(A))$ and $N_c^{(L)}(A) = S^{(L)}(A)$.

[1] For example, $[[a, a], [a, b]]^T, a << b$ requires infinite iterations analytically.

We measure these errors in the Sinkhorn iteration for the non-converging cases with ϵ. Since the errors of one side are always zero because of the normalization, those on the other side are collected. We show the results in Fig. 2.



(a) $\begin{pmatrix} 1 & 1 & 1 \\ 1 & \epsilon & \epsilon \\ 1 & \epsilon & \epsilon \end{pmatrix}$ (b) $\begin{pmatrix} \epsilon & 1 & \epsilon \\ 1 & \epsilon & 1 \\ \epsilon & 1 & \epsilon \end{pmatrix}$

Figure 2. Sinkhorn iteration for non-converging patterns. Constraint error does not decrease for $\epsilon = 0$ because Sinkhorn's theorem does not hold. Remaining error length corresponds to ϵ size. For any fixed unrolling number, there exists ϵ that can block convergence.

From the figure, we can check if the Sinkhorn's theorem is correct under $\epsilon = 0$. For $\epsilon > 0$ cases, however, the error curves versus the number of iterations forms plateaus. This phenomenon means the constraint error remains high before the iteration reaches a certain iteration number and also the number seems to relate to the ϵ's value in the graph. The reason can be understood analytically, and we can explain the number of iterations to converge in the second case. The details are in Appendix D. This property is unsuitable when *fixing* the unrolling number of Sinkhorn iterations in neural networks because there remains a possibility that unnormalized solutions may be emitted even if the input is a strictly positive matrix.

Unnormalized solutions in the Sinkhorn iteration may be more problematic than the continuous relaxation errors. Continuous relaxation can be justified from the Birkoff-von Neumann theorem (Hurlbert, 2008; Păunescu & Rădulescu, 2017) as the doubly stochastic matrices (Birkoff polytope (Baumeister & Ladisch, 2018)) are the convex hull of permutations. However, unnormalized matrices go outside of this theorem and no longer have such explanations.

2.2. Possible vanishing gradient problem

For the sigmoid function, which is a 2-class softmax function, there is a well-known vanishing gradient problem (Goh et al., 2017) caused by the "tail" of the derivative of the function going to zero. In the softmax function with temperature, this problem can be explained analytically (see appendix A

for the details). This property implies that there is a vanishing gradient problem on the Sinkhorn iteration because the Sinkhorn iteration can be considered as an extension of the softmax function. This is explained by the fact that the Sinkhorn iteration for an exponentiated vector trivially converges at one step and is strictly the same as the softmax function.

3. Hungarian algorithm with straight-through estimator

To avoid the problems mentioned in the previous section, we propose using the Hungarian algorithm with an STE. The Hungarian algorithm emits a permutation solution of bipartite graph matching of the following objective function (Kuhn, 1955; Munkres, 1957):

$$\hat{P} = \arg\max_{P} \sum_{i,j} \chi_{i,j} \cdot P_{i,j}, \tag{1}$$

$$s.t. \sum_{i} P_{i,j} = \mathbf{1}^T, {}^\forall j, \sum_{j} P_{i,j} = \mathbf{1}, {}^\forall i, P_{i,j} \in \{0,1\}. \tag{2}$$

We denote the Hungarian algorithm as an operator, $\hat{P} = H(\chi)$. This algorithm does not require the iteration number as a hyper-parameter and is free from the non-converging problem in fixed unrolling of the Sinkhorn iteration. The Hungarian algorithm can be viewed as a function that inputs a matrix and outputs a permutation matrix. We treat this and the argmax function as a pair because of the analogy of the equivalence between the softmax function and the Sinkhorn iteration for a vector. Therefore, the hard versions of the softmax function and the Sinkhorn iteration, which are the argmax function and Hungarian algorithm, are expected to have some similarities. We can visually understand this similarity in Appendix B. Using the Hungarian algorithm in a neural network has a problem in that it is not differentiable. We propose to apply an STE to this problem, which means $\frac{\partial H(\chi)}{\partial \chi} \simeq I$. In this formula, we implicitly vectorize the matrices $\hat{P} = H(\chi)$ and χ. This technique is used to obtain a hard solution of a discrete function (Jang et al., 2016).

We can obtain a doubly stochastic matrix as a posterior delegate of the latent permutation by taking the mean of the permutation samples which are generated from perturbations of the input matrix. This procedure is directly justified from the Birkoff-von Neumann theorem, which declares any doubly stochastic matrix can be obtained from an infinite sample mean of permutation matrices.

Using Gumbel noise to obtain a doubly stochastic matrix is one option because exploiting the Hungarian algorithm with Gumbel noise to obtain a MAP estimator for inter-frame keypoint matching combined with scale-invariant feature transform (SIFT) descriptors using linear models (Li et al., 2013) based on Perturb-and-MAP (Papandreou & Yuille, 2011) has been tested. Therefore, we add Gumbel noise to the linear output of the neural network in the same manner as with the conventional Gumbel-Sinkhorn algorithm (Mena et al., 2018) in the log space. The network structure example is shown in Fig. 5 in Appendix C.

4. Experiments

We basically conformed the Gumbel-Sinkhorn (Mena et al., 2018) settings to implement a neural network to compare the performances of the conventional Gumbel-Sinkhorn and Hungarian-STE (HSTE) algorithm.[2] The procedure of the experiments was as follows.

We first prepared a vector $\hat{\mathbf{x}}$ of N sorted elements (puzzle pieces or numbers) then randomized the order of the elements. This randomization is denoted as an answer permutation matrix $\hat{P}$, which means the input vector is $\mathbf{x} = \hat{P}\hat{\mathbf{x}}$. The elements were independently fed into a neural network consisting of functions $g = g_1 \circ g_2$. Then each element of $\mathbf{x}$ generates N row vectors of size N. We stacked the row vectors to construct an $N \times N$ matrix. We added Gumbel noise to this matrix, divided the matrix by the temperature hyperparameter, and then applied the Hungarian algorithm[3] $H()$ to obtain permutation matrix $P = H(g(\mathbf{x}))$. Finally, we reconstructed the original vector by taking $\tilde{\mathbf{x}} = P^T \cdot \mathbf{x}$.

The evaluation metrics were the same as those in a previous work (Mena et al., 2018), which were categorized into two, *i.e.*, feature-based errors and order-based errors. Feature-based errors can be calculated from $\tilde{\mathbf{x}}$ and $\hat{\mathbf{x}}$, and order-based errors can be calculated from P and $\hat{P}$. For feature-based errors, we used $l1$ and $l2$ errors between the correctly ordered answer and reordered output. For order-based errors, we used $\mathrm{Prop.\,wrong}$, $\mathrm{Prop.\,any\,wrong}$, and $\mathrm{Kendall\,tau}$. The $\mathrm{Prop.\,wrong}$ metric is the order-matching rate between two $N \times N$ permutations P and Q using a member-wise product $\odot$, which is expressed as $\mathrm{Prop.\,wrong} = 1 - \frac{1}{\mathrm{N}} \sum \mathrm{P} \odot \mathrm{Q}$. The $\mathrm{Prop.\,any\,wrong}$ metric is 0 when $\mathrm{Prop.wrong}$ is 0; otherwise, 1. The $\mathrm{Kendall\,tau}$ metric is calculated using Kendall's tau measure (Kendall, 1945), which is a correlation measure for orders.

4.1. MNIST jigsaw puzzle task

We compared the algorithms in the MNIST (LeCun et al., 1998) jigsaw puzzle task used in a previous study (Mena

[2] Our implementation is written in pytorch, and the details are based on https://github.com/google/gumbel_sinkhorn and https://github.com/HeddaCohenIndelman/Learning-Gumbel-Sinkhorn-Permutations-w-Pytorch

[3] We actually used the linear-programming version of the Hungarian algorithm, which is known to be equivalent. (Matousek & Gaertner, 2007)

Table 1. Mean Kendal tau scores of MNIST zigsaw puzzle task in evaluation set

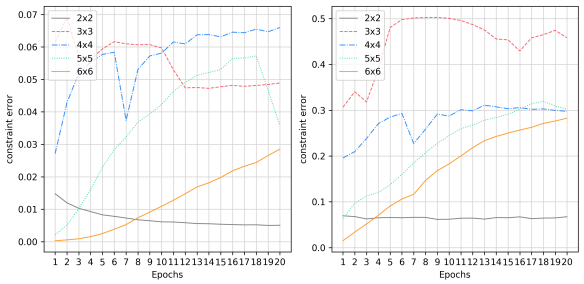
piece size	2x2	3x3	4x4	5x5	6x6
Gumbel-Sinkhorn	1.00	0.66	0.33	0.23	0.16
Hungarian-STE	1.00	0.63	0.29	0.15	0.10

Table 2. Number-sorting-task mean scores

$N =$	15	80	100	120	200	400
Kendall tau						
Sinkhorn	.82	1.	1.	.97	.95	.89
HSTE w/o noise	1.	1.	.70	.61	.86	.86
Gumbel-Sinkhorn	.88	.94	.86	.92	.72	.59
HSTE w/ noise	1.	1.	1.	1.	.96	.80
Prop. any wrong						
Sinkhorn	.70	.00	.10	1.	1.	1.
HSTE w/o noise	.00	.00	1.	1.	1.	1.
Gumbel-Sinkhorn	.60	1.	1.	1.	1.	1.
HSTE w/ noise	.00	.00	.00	.00	1.	1.

et al., 2018). We divided the input image into $M \times M$ "pieces" *i.e.*, $N = M^2$, shuffled the order, individually input the pieces to g_1, which is now a convolutional neural network (CNN), input them to g_2 multi-layer perceptron (MLP), and concatenated each M^2 size of row vectors to construct an $M^2 \times M^2$ sized matrix (see Table 3 in Appendix C for the details of this experiment).

We first confirmed that the conventional Gumbel-Sinkhorn algorithm has a non-converging problem. We show the constraint error curves in Fig. 3 using the definition of the error in Section 2. We do not show the results for the Hungarian-STE algorithm as the errors are trivially always zero. The errors do not decrease to zero in any of the $M \times M$ cases. Moreover, in higher $M \times M$ cases, there was a tendency of the error increasing as the training progressed. A possible explanation for this phenomenon is that higher M generates more similar pieces and the decision of permutation becomes more difficult because there should be many equivalent permutation solutions. If the input matrix is occasionally similar to the oscillation pattern mentioned in Section 2, the output matrix may far from the format of permutation if the fixed-length unrolling of the Sinkhorn iteration is used.



(a) Mean constraint error (b) Max. constraint error

Figure 3. Constraint error curves by epochs in conventional Gumbel-Sinkhorn algorithm in MNIST jigsaw puzzle task

We then evaluated the performances of the two algorithms. The results indicate that the Hungarian-STE and Gumbel-Sinkhorn algorithms had almost comparative results in this task. The results are shown in Table 1 (see Table 4 in Appendix E for full result).

4.2. Number sorting task

We then compared the algorithms in the number sorting task also used in the previous study (Mena et al., 2018). We set $\hat{\mathbf{x}}$ to N real numbers drawn from an uniform distribution $U(0, 1)$ and used MLPs as g.

The results, based on our implementation, are listed in Table 2 (See Table 5 in Appendix E for full result). It is obvious that the Hungarian-STE algorithm with noise outperformed the conventional Gumbel-Sinkhorn algorithm in all metrics, especially with a high N number (see Table 3 in Appendix C for the details of this experiment).

5. Discussion

We confirmed that a non-converging normalization problem occurred in the experiment on the MNIST jigsaw puzzle task with the Sinkhorn iteration and showed clear performance improvement on the number sorting task by simply replacing the Sinkhorn iteration with the Hungarian-STE algorithm to avoid this problem.

The effect of adding noise differs between the (Gumbel-) Sinkhorn and Hungarian-STE algorithms. Although the performance of the Gumbel-Sinkhorn algorithm did not differ whether the noise was added or not, the Hungarian-STE's performance largely improved by adding noise. This result suggests that this noise reduces the effect of the error from the gradient of the straight-through estimator by taking the mean of the final losses, which correspond to the noise.

For the MNIST jigsaw puzzle task, the network used for the conventional Gumbel-Sinkhorn algorithm (Mena et al., 2018) seems too simple to model the MNIST jigsaw puzzle. The scores may improve by searching more richer neural network structures or hyperparameters, which is out-of-focus in this work.

References

Baumeister, B. and Ladisch, F. A property of the Birkhoff polytope. *Algebraic Combinatorics*, 1(2):275–281, 2018.

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

Cour, T., Srinivasan, P., and Shi, J. Balanced graph matching. In *Advances in Neural Information Processing Systems*, pp. 313–320, 2007.

Cuturi, M., Teboul, O., and Vert, J.-P. Differentiable sorting using optimal transport: The Sinkhorn cdf and quantile operator. *arXiv preprint arXiv:1905.11885*, 2019.

Goh, G. B., Hodas, N. O., and Vishnu, A. Deep learning for computational chemistry. *Journal of computational chemistry*, 38(16):1291–1307, 2017.

Grover, A., Wang, E., Zweig, A., and Ermon, S. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*, 2019.

Hurlbert, G. A short proof of the Birkhoff-von Neumann theorem. *preprint (unpublished)*, 2008.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kawachi, Y. and Suzuki, T. Unsupervised auto-encoding multiple-object tracker for constraint-consistent combinatorial problem. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6054–6058, 2020.

Kendall, M. G. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.

Kuhn, H. W. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, K., Swersky, K., and Zemel, R. Efficient feature learning using perturb-and-map. 2013.

Matousek, J. and Gaertner, B. (eds.). *Understanding and Using Linear Programming*. Springer Berlin Heidelberg, 2007.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. Learning latent permutations with Gumbel-Sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

Milan, A., Rezatofighi, S. H., Dick, A., Reid, I., and Schindler, K. Online multi-target tracking using recurrent neural networks. In *Proc. AAAI*, 2017.

Milz, S., Arbeiter, G., Witt, C., Abdallah, B., and Yogamani, S. Visual SLAM for automated driving: Exploring the applications of deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

Munkres, J. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

Papandreou, G. and Yuille, A. L. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *2011 International Conference on Computer Vision*, pp. 193–200. IEEE, 2011.

Păunescu, L. and Rădulescu, F. A generalisation to Birkhoff–von Neumann theorem. *Advances in Mathematics*, 308: 836–858, 2017.

Santa Cruz, R., Fernando, B., Cherian, A., and Gould, S. Deeppermnet: Visual permutation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3949–3957, 2017.

Sarlin, P.-E., DeTone, D., Malisiewicz, T., and Rabinovich, A. Superglue: Learning feature matching with graph neural networks. *arXiv preprint arXiv:1911.11763*, 2019.

Sinkhorn, R. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

Sinkhorn, R. and Knopp, P. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

A. Vanishing gradient in softmax function with temperature

In the softmax function with temperature, vanishing gradient problem can be understood as follows. We express the softmax function with temperature τ as

$$\sigma(\mathbf{x}/\tau) = [\cdots, \frac{e^{x_i/\tau}}{\sum_k e^{x_k/\tau}}, \cdots]^T. \tag{3}$$

The derivative of this function used in the backward procedure using identity matrix I is

$$\frac{\partial \sigma_i(\mathbf{x}/\tau)}{\partial x_j} = \frac{1}{\tau}\sigma_i(\mathbf{x}/\tau)\left(I_{i,j} - \sigma_j(\mathbf{x}/\tau)\right) \tag{4}$$

We focus on the m-th largest and l-th second largest value on $\mathbf{x}$ and obtain

$$\sigma_m(\mathbf{x}/\tau) = \frac{1}{1 + \epsilon_l + \sum_{k \neq m,l} \epsilon_k} \tag{5}$$

$$\epsilon_k = e^{(x_k - x_m)/\tau} \tag{6}$$

The m, m-th gradient is

$$\frac{\partial \sigma_m(\mathbf{x}/\tau)}{\partial x_m} = \tag{7}$$

$$\frac{1}{\tau}\frac{1}{1+\epsilon_l+\sum_{k\neq m,l}\epsilon_k}\left(1-\frac{1}{1+\epsilon_l+\sum_{k\neq m,l}\epsilon_k}\right) \tag{8}$$

For $\mathbf{x}$, this nearly equals 0 when

$$\epsilon_l + \sum_{k\neq m,l}\epsilon_k \simeq 0 \text{ if} \tag{9}$$

$$\epsilon_l \simeq 0 \tag{10}$$

$$x_l - x_m \simeq -\infty \tag{11}$$

This means the nearly argmax condition $x_m >> x_l$. If we are not in this condition, which means the difference from the maximum in $\mathbf{x}$ is nearly constant c, we define $|\mathbf{x}| = K$, then

$$\frac{\partial \sigma_m(\mathbf{x}/\tau)}{\partial x_m} = \tag{12}$$

$$\frac{1}{\tau}\frac{1}{1+(K-1)e^{-c/\tau}}\left(1-\frac{1}{1+(K-1)e^{-c/\tau}}\right) \tag{13}$$

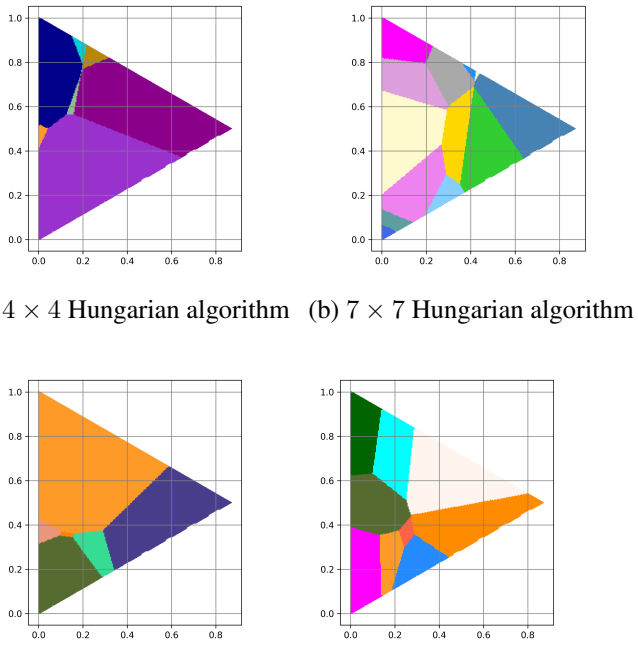
Using $1-1/(1+a) \simeq a$ if $a \simeq 0$,

$$\frac{\partial \sigma_m(\mathbf{x}/\tau)}{\partial x_m} \simeq (K-1)\frac{1}{\tau}e^{-c/\tau} \tag{14}$$

in near low temperature $\tau \simeq 0$. As $e^{-c/\tau}$ goes to 0 far faster than $1/\tau$, the vanishing gradient also occurs in low temperature. In conclusion, the softmax function with temperature has the problems in that we cannot get close the relaxation gap because of the vanishing gradient problem. As 32-bit floating-point operation becomes numerically 0 from around $\exp(-1000)$, $\tau \simeq c/1000$ is the actual lower bound of the calculation.

B. Hungarian algorithm visualization

We visualized the Hungarian algorithm as a function and argmax function in Fig. 4. We took three random matrices from standard Gaussian distribution as $\chi_1, \chi_2, \chi_3 \sim |\mathcal{N}(0,1)|^{D\times D}$ in the Hungarian algorithm or $\chi_1, \chi_2, \chi_3 \sim |\mathcal{N}(0,1)|^{D}$ in the argmax function, made a triangle using these points as the vertices, ran the Hungarian algorithm or argmax function for each internal grid point $p = a\chi_1 + b\chi_2 + (1-a-b)\chi_3$ only if $(1-a-b) \geq 0$ where $a = \{0, 1/200, \cdots, 1\}$ and $b\{0, 1/200, \cdots, 1\}$, remapped into an equilateral triangle by $p' = a[0.5\sqrt{3}, 0.5]^T + b[0,0]^T + (1-a-b)[0,1]^T$ and colored the points using small patches of rectangles based on the kind of permutation matrix or argmax result index. From this visualization, we can see that the argmax function and the Hungarian algorithm have very similar landscapes and similar inputs has the same answer in many cases.



(a) 4×4 Hungarian algorithm (b) 7×7 Hungarian algorithm

(c) 16-dim. argmax function (d) 49-dim. argmax function

Figure 4. Visualizations of Hungarian algorithm and argmax function. Axes are a and b. These two landscapes look very similar. Hungarian algorithm's landscape seems to reflect shape of latent polytope. As argmax function can be used with STE in Gumbel-Softmax (Jang et al., 2016), we argue that it also can be used with STE.

C. Implementation details

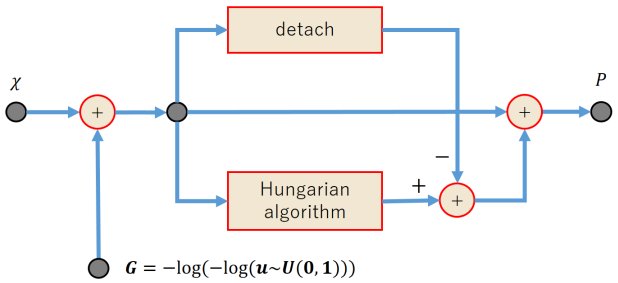


Figure 5. Example of computational graph of Hungarian algorithm with STE and Gumbel noise in typical define-by-run framework. In forward path, "detach" just bypasses input. Hungarian algorithm and "detach" are ignored in backward path. We did not exponentiate noised variable before using Hungarian algorithm due to performance reasons.

D. Analytical explanation of iteration plateaus

We assume the case

$$A = \begin{pmatrix} \epsilon & 1 & \epsilon \\ 1 & \epsilon & 1 \\ \epsilon & 1 & \epsilon \end{pmatrix} \tag{15}$$

However, the same methodology can be applied in any case. Following calculation, we omit the symmetric terms. After row-wise normalization, the matrix becomes

$$N_r(A) = \begin{pmatrix} \epsilon/(1+2\epsilon) & 1/(1+2\epsilon) & \cdot \\ 1/(2+\epsilon) & \epsilon/(2+\epsilon) & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}, \tag{16}$$

the 1st order Taylor approximation for the row-wise normalization result is

$$N'_r(A) = \begin{pmatrix} \epsilon & 1-2\epsilon & \cdot \\ \frac{1}{2} - \frac{1}{4}\epsilon & \frac{1}{2}\epsilon & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}, \tag{17}$$

the column-wise normalization of the approximated row-wise normalization result is

$$N_c(N'_r(A)) \tag{18}$$

$$= \begin{pmatrix} \epsilon/(\frac{1}{2} + \frac{7}{4}\epsilon) & (1-2\epsilon)/(2 - \frac{7}{4}\epsilon) & \cdot \\ (\frac{1}{2} - \frac{1}{4}\epsilon)/(\frac{1}{2} + \frac{7}{4}\epsilon) & (\frac{1}{2}\epsilon)/(2 - \frac{7}{4}\epsilon) & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}, \tag{19}$$

and the 1st order Taylor approximation of the column-wise normalization of the approximated row-wise normalization result is

$$N'_c(N'_r(A)) \tag{20}$$

$$= \begin{pmatrix} 2\epsilon & \frac{1}{2} - \frac{1}{8}\epsilon & \cdot \\ 1 - 4\epsilon & \frac{1}{4}\epsilon & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}. \tag{21}$$

These correspond to the one iteration of the Sinkhorn iteration. We unroll one more iteration. The first step is

$$N_r(N'_c(N'_r(A))) \tag{22}$$

$$= \begin{pmatrix} 2\epsilon/(\frac{1}{2} + \frac{31}{8}\epsilon) & (\frac{1}{2} - \frac{1}{8}\epsilon)/(\frac{1}{2} + \frac{31}{8}\epsilon) & \cdot \\ (1-4\epsilon)/(2 - \frac{31}{4}\epsilon) & (\frac{1}{4}\epsilon)/(2 - \frac{31}{4}\epsilon) & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}, \tag{23}$$

the second step is

$$N'_r(N'_c(N'_r(A))) \tag{24}$$

$$= \begin{pmatrix} 4\epsilon & 1-8\epsilon & \cdot \\ \frac{1}{2} + \frac{29}{16}\epsilon & \frac{1}{8}\epsilon & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}, \tag{25}$$

Table 3. Hyperparameter details

N-number sorting task	
number of test samples	10
number of max. epochs	500
learning rate	0.1
optimizer	Adam
batch size (equals training set size)	10
Gumbel noise samples	10
temperature in Gumbel-Sinkhorn	1.0
Sinkhorn unrolling number L	20
1st (g_1) affine layer's shape	1×32
1st (g_1) nonlinear function	ReLU
2nd (g_2) affine layer's shape	$32 \times N$

$N = M \times M$ MNIST jigsaw puzzle task	
number of max. epochs	20
learning rate	1e-4
optimizer	Adam
batch size	32
Gumbel noise samples	10
temperature in Gumbel-Sinkhorn	1.0
Sinkhorn unrolling number L	20
1st (g_1) 2D conv. layer's hidden layer	64
1st (g_1) 2D conv. layer's kernel size	5
1st (g_1) 2D conv. layer's padding size	2
1st (g_1) nonlinear function	ReLU
1st (g_1) 2D max pooling layer's stride	2
2nd (g_2) affine (w/o bias) layer's shape	$64 \cdot \lfloor \frac{28}{2M} \rfloor^2 \times M^2$

the third step is

$$N_c(N'_r(N'_c(N'_r(A)))) \tag{26}$$

$$= \begin{pmatrix} 4\epsilon/(\frac{1}{2}+\frac{157}{16}\epsilon) & (1-8\epsilon)/(2-\frac{127}{8}\epsilon) & \cdot \\ (\frac{1}{2}+\frac{29}{16}\epsilon)/(\frac{1}{2}+\frac{157}{16}\epsilon) & (\frac{1}{8}\epsilon)/(2-\frac{127}{8}\epsilon) & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}, \tag{27}$$

and the fourth step is

$$N'_c(N'_r(N'_c(N'_r(A)))) \tag{28}$$

$$= \begin{pmatrix} 8\epsilon & \frac{1}{2}-\frac{1}{32}\epsilon & \cdot \\ 1-16\epsilon & \frac{1}{16}\epsilon & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}. \tag{29}$$

Therefore, we can induce the general term of the 1,1 element for example. This is described using the number of normalizations $k, l = \lceil \frac{1}{2}k \rceil$ as

$$2^{k-1} \cdot \epsilon \tag{30}$$

From this, 2^{k-1} grows exponentially but the ϵ is extremely small. This balance seems to cause the plateaus. As the 1,1 element will converge to $\frac{1}{2}$ in the actual iteration, the change point for the 1,1 element will be at the step of

$$k = -\log_2 \epsilon + 2 \tag{31}$$

This approximately equals

$$\epsilon = 0.1, k \simeq 5 \tag{32}$$
$$\epsilon = 0.01, k \simeq 9 \tag{33}$$
$$\epsilon = 0.001, k \simeq 12 \tag{34}$$
$$\epsilon = 0.0001, k \simeq 15 \tag{35}$$
$$\epsilon = 1e-5, k \simeq 19 \tag{36}$$
$$\epsilon = 1e-6, k \simeq 22 \tag{37}$$
$$\epsilon = 1e-7, k \simeq 25 \tag{38}$$
$$\epsilon = 1e-8, k \simeq 29 \tag{39}$$
$$\epsilon = 1e-9, k \simeq 32 \tag{40}$$
$$\epsilon = 1e-10, k \simeq 35 \tag{41}$$

This roughly expresses the iteration plateau effect.

E. Full results of MNIST jigsaw puzzle and number sorting tasks

Table 4. MNIST zigsaw puzzle task results. Up-arrow means higher is better. Values are mean of scores.

	MNIST (train)					MNIST (evaluate)				
	2x2	3x3	4x4	5x5	6x6	2x2	3x3	4x4	5x5	6x6
	Gumbel-Sinkhorn									
↑ Kendall tau	1.00	0.65	0.33	0.22	0.16	1.00	0.66	0.33	0.23	0.16
↓ Prop. wrong	0.00	0.23	0.56	0.78	0.88	0.00	0.23	0.56	0.78	0.88
↓ Prop. any wrong	0.00	0.62	1.00	1.00	1.00	0.00	0.61	1.00	1.00	1.00
↓ $l1$	0.00	0.02	0.06	0.13	0.16	0.00	0.02	0.06	0.13	0.16
↓ $l2$	0.00	0.01	0.03	0.06	0.08	0.00	0.01	0.03	0.06	0.07
↓ constraint error	0.01	0.05	0.07	0.03	0.03	0.01	0.05	0.07	0.04	0.03
	Hungarian-STE									
↑ Kendall tau	1.00	0.62	0.29	0.15	0.11	1.00	0.63	0.29	0.15	0.10
↓ Prop. wrong	0.00	0.27	0.64	0.85	0.92	0.00	0.26	0.63	0.84	0.92
↓ Prop. any wrong	0.00	0.69	1.00	1.00	1.00	0.00	0.67	1.00	1.00	1.00
↓ $l1$	0.00	0.02	0.07	0.15	0.17	0.00	0.02	0.07	0.14	0.17
↓ $l2$	0.00	0.01	0.06	0.12	0.14	0.00	0.01	0.06	0.11	0.14
↓ constraint error	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 5. Number sorting task results. Up-arrow means higher is better. Values are mean of scores.

N=	5	10	15	80	100	120	150	200	250	400
	Sinkhorn									
↑ Kendall tau	1.00	1.00	0.82	1.00	1.00	0.97	0.95	0.95	0.94	0.89
↓ Prop. wrong	0.00	0.00	0.09	0.00	0.00	0.03	0.03	0.04	0.04	0.08
↓ Prop. any wrong	0.00	0.00	0.70	0.00	0.10	1.00	1.00	1.00	1.00	1.00
↓ $100 \times l1$	0.00	0.00	0.23	0.00	0.00	0.03	0.03	0.03	0.03	0.04
↓ $1000 \times l2$	0.00	0.00	0.10	0.00	0.00	0.01	0.01	0.01	0.00	0.01
	Hungarian-STE without Gumbel noise									
↑ Kendall tau	1.00	1.00	1.00	1.00	0.70	0.61	0.98	0.86	0.97	0.86
↓ Prop. wrong	0.00	0.00	0.00	0.00	0.24	0.33	0.01	0.12	0.02	0.11
↓ Prop. any wrong	0.00	0.00	0.00	0.00	1.00	1.00	0.50	1.00	1.00	1.00
↓ $100 \times l1$	0.00	0.00	0.00	0.00	0.24	0.29	0.00	0.04	0.00	0.02
↓ $1000 \times l2$	0.00	0.00	0.00	0.00	0.04	0.06	0.00	0.00	0.00	0.00
	Gumbel-Sinkhorn									
↑ Kendall tau	1.00	1.00	0.88	0.94	0.86	0.92	0.82	0.72	0.77	0.59
↓ Prop. wrong	0.00	0.00	0.08	0.05	0.11	0.07	0.13	0.23	0.19	0.34
↓ Prop. any wrong	0.00	0.00	0.60	1.00	1.00	1.00	1.00	1.00	1.00	1.00
↓ $100 \times l1$	0.00	0.00	0.28	0.03	0.12	0.06	0.10	0.16	0.10	0.12
↓ $1000 \times l2$	0.00	0.00	0.16	0.00	0.03	0.01	0.02	0.04	0.02	0.02
	Hungarian-STE with Gumbel noise									
↑ Kendall tau	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96	0.93	0.80
↓ Prop. wrong	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.06	0.16
↓ Prop. any wrong	0.00	0.00	0.00	0.00	0.00	0.00	0.10	1.00	1.00	1.00
↓ $100 \times l1$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.03
↓ $1000 \times l2$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00