2nd Symposium on Advances in Approximate Bayesian Inference, 2019 1–7

Neural Permutation Processes

Ari Pakman
Yueqi Wang
Liam Paninski
Columbia University

Abstract

We introduce a neural architecture to perform amortized approximate Bayesian inference over latent random permutations of two sets of objects. The method involves approximating permanents of matrices of pairwise probabilities using recent ideas on functions defined over sets. Each sampled permutation comes with a probability estimate, a quantity unavailable in MCMC approaches. We illustrate the method in sets of 2D points and MNIST images.

1. Introduction

Posterior inference in generative models with discrete latent variables presents well-known challenges when the variables live in combinatorially large spaces. In this work we focus on the popular and non-trivial case where the latent variables represent random permutations. While inference in these models has been studied in the past using MCMC techniques (Diaconis, 2009) and variational methods (Linderman et al., 2018; Mena et al., 2018), here we propose an amortized approach, whereby we invest computational resources to train a model, which later is used for very fast posterior inference (Gershman and Goodman, 2014). Unlike the variational autoencoder approach (Kingma and Welling, 2013), in our case we do not learn a generative model. Instead, the latter is postulated (through its samples) and posterior inference is the main focus of the learning phase. This approach has been recently explored in sundry contexts, such as Bayesian networks (Stuhlmüller et al., 2013), sequential Monte Carlo (Paige and Wood, 2016), probabilistic programming (Ritchie et al., 2016; Le et al., 2016), neural decoding (Parthasarathy et al., 2017) and particle tracking (Sun and Paninski, 2018).

Our method is inspired by the technique introduced in (Pakman and Paninski, 2018) to perform amortized inference over discrete labels in mixture models. The basic idea is to use neural networks to express posteriors in the form of multinomial distributions (with varying support) in terms of fixed-dimensional, distributed representations that respect the permutation symmetries imposed by the discrete variables.

After training the neural architecture using labeled samples from a particular generative model, we can obtain independent approximate posterior samples of the permutation posterior for any new set of observations of arbitrary size. These samples can be used to compute approximate expectations, as high quality importance samples, or as independent Metropolis-Hastings proposals.

2. Amortizing Permutations

Let us consider the generative model

$$N \sim p(N) \qquad c_{1:N} \sim p(c_{1:N}) \qquad x_{c_i}, y_i \sim p(x_{c_i}, y_i) \tag{1}$$

Here $p(c_{1:N}) = 1/N!$ is a uniform distribution over permutations, with the random variable

$$c_i \in \{1, \dots, N\}, \quad c_i \neq c_j \text{ for } i \neq j$$

denoting that x_{c_i} is paired with y_i. As a concrete example, think of y_i as a noise-corrupted version of a permuted sample x_{c_i}. Given two sets of N data points $\mathbf{x} = \{x_i\}$, $\mathbf{y} = \{y_i\}$, we are interested in iid sampling the posterior of the c_i's, using a decomposition

$$p(c_{1:N}|\mathbf{x}, \mathbf{y}) \quad = \quad p(c_1|\mathbf{x}, \mathbf{y}) p(c_2|c_1, \mathbf{x}, \mathbf{y}) \dots p(c_N|c_{1:N-1}, \mathbf{x}, \mathbf{y}); \tag{2}$$

note now that $p(c_N|c_{1:N-1}, \mathbf{x}, \mathbf{y}) = 1$, since the last point y_N is always matched with the last unmatched point among the x_i's. A generic factor in (2) is

$$p(c_n|c_{1:n-1}, \mathbf{x}, \mathbf{y}) = \frac{p(c_1 \dots c_n, \mathbf{x}, \mathbf{y})}{\sum_{c'_n} p(c_1, \dots, c'_n, \mathbf{x}, \mathbf{y})} \tag{3}$$

where c_n takes values in $\{1, \dots, N\}$ not taken by $c_{1:n-1}$. Consider first

$$p(c_1, \dots, c_n, \mathbf{x}, \mathbf{y}) \quad = \quad (N!)^{-1} p(x_{c_1}, y_1) \dots p(x_{c_n}, y_n) R(c_{n+1:N}, \mathbf{x}, \mathbf{y}|c_1 \dots c_n) \tag{4}$$

where we defined

$$R(c_{n+1:N}, \mathbf{x}, \mathbf{y}|c_1 \dots c_n) = \sum_{\{c_{n+1} \dots c_N \in s_n\}} \prod_{i=n+1}^{N} p(x_{c_i}, y_i) \tag{5}$$

and $s_n = \{1 \dots N\}/\{c_1 \dots c_n\}$ is the set of available indices after choosing $c_{1:n}$. Note that R in (5) is the permanent of a $(N-n)\times(N-n)$ matrix, an object whose computation is known to be a #P problem (Valiant, 1979). Inserting (4) into (3) gives

$$p(c_n|c_{1:n-1}, \mathbf{x}, \mathbf{y}) = \frac{p(x_{c_n}, y_n) R(c_{n+1:N}, \mathbf{x}, \mathbf{y}|c_1 \dots c_n)}{\sum_{c'_n} p(x_{c'_n}, y_n) R(c_{n+1:N}, \mathbf{x}, \mathbf{y}|c_1 \dots c'_n)} \,. \tag{6}$$

Note that (6) does not depend on $\{x_{c_i}\}_{i=1}^{n-1}$ or $\{y_i\}_{i=1}^{n-1}$, except for restricting the allowed values for c_n. Now, the function R in (5) depends on the unmatched points $\{x_{c_i}\}_{i=n+1}^{N}$, and $\{y_i\}_{i=n+1}^{N}$, in such a way that it is invariant under separate permutations of the elements of each set. Following (Zaheer et al., 2017; Gui et al., 2019), these permutation symmetries can be captured by introducing functions $h : \mathbb{R}^{d_x} \to \mathbb{R}^{d_h}$, $f : \mathbb{R}^{2d_h} \to \mathbb{R}$, defining

$$H_y = \sum_{i=n+1}^{N} h(y_i) \qquad H_{x,c_n} = \sum_{i=n+1}^{N} h(x_{c_i}) \,, \tag{7}$$

and approximating

$$R(c_{n+1:N}, \mathbf{x}, \mathbf{y} | c_1 \dots c_n) \simeq e^{f(H_{x,c_n}, H_y)} . \tag{8}$$

The subindex c_n in H_{x,c_n} indicates a value that cannot be taken by the c_i's in the sum in (7). Inserting (8) into (6) gives

$$q_\theta(c_n | c_{1:n-1}, \mathbf{x}, \mathbf{y}) = \frac{p(x_{c_n}, y_n) e^{f(H_{x,c_n}, H_y)}}{\sum_{c'_n} p(x_{c'_n}, y_n) e^{f(H_{x,c'_n}, H_y)}} \tag{9}$$

which is our proposed approximation for (6), with θ representing the parameters of the neural networks for h, f. The neural architecture is schematized in Figure 2. The pairwise density $p(y_n, x_{c_n})$ can be either known in advance, or represented by a parametrized function to be learned (in the latter case we assume we have samples from it). We call this approach the Neural Permutation Process (NPP).

3. Objective Function

In order to learn the parameters θ of the neural networks h, f (and possibly $p(x_{c_n}, y_n)$), we use stochastic gradient descent to minimize the expected KL divergence,

$$\mathbb{E}_{p(N)p(\mathbf{x},\mathbf{y})} D_{\mathrm{KL}}(p(c|\mathbf{x},\mathbf{y}) \| q_\theta(c|\mathbf{x},\mathbf{y})) = -\mathbb{E}_{p(N)} \mathbb{E}_{p(c_{1:N},\mathbf{x},\mathbf{y})} \left[\sum_{n=2}^{N} \log q_\theta(c_n | c_{1:n-1}, \mathbf{x}, \mathbf{y}) \right] + \text{const.}$$

Samples from $p(N)p(c_{1:N}, \mathbf{x}, \mathbf{y})$ are obtained from the generative model (1). If we can take an unlimited number of samples, we can potentially train a neural network to approximate $p(c_n | c_{1:n-1}, \mathbf{x})$ arbitrarily accurately.

4. Examples

In Figure 1 we show results for the following two examples. Both cases illustrate how a probabilistic approach captures the ambiguities in the observations.

Noisy pairs in 2D: the generative model is

$$x_{c_i} \sim N(0, 3) \qquad y_i \sim N(x_{c_i}, 0.6\mathbf{1}_2) \quad i = 1 \dots N. \tag{10}$$

MNIST digits: the generative model is

$$y_i, x_{c_i} \sim \text{Unif [Pairs of MNIST digits with same label]} \quad i = 1 \dots N. \tag{11}$$

An additional symmetry. Note finally that if $p(y_n, x_{c_n}) = p(x_{c_n}, y_n)$ (as is the case in these examples), the additional symmetry $f(H_{x,c_n}, H_y) = f(H_y, H_{x,c_n})$ can be captured by introducing a new function g and defining $f(H_{x,c_n}, H_y) = f(g(H_{x,c_n}) + g(H_y))$. Interestingly, as shown in Figure 2, we find that a higher likelihood is obtained instead by $f(H_{x,c_n}, H_y) = f(H_{x,c_n} \odot H_y)$, where $\odot$ indicates componentwise multiplication. To our knowledge, this type of encoding has not been studied in the literature, and we plan to explore it further in the future.

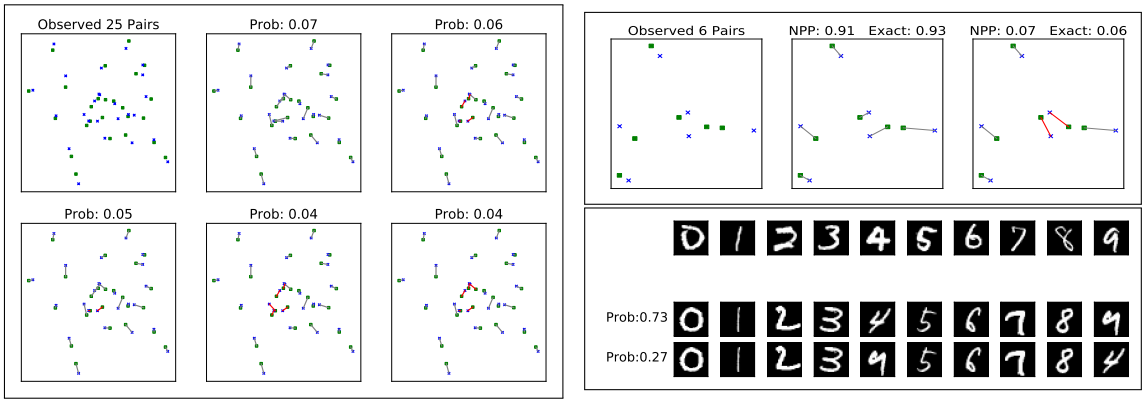

Figure 1: **Neural Permutation Process.** *Left: Noisy pairs in 2D.* $N = 25$ pairs from the model (10), and five different samples from the posterior. Red links indicate departures from the highest probability matchings. *Upper right:* $N = 6$ pairs from the same model. In this case we can compute the exact probabilities, which track the NPP approximate values well. *Lower right:* Two posterior samples from from the model (11), given ten pairs of distinct MNIST digits. Note that the samples capture well the ambiguity between digits '4' and '9'.

5. Conclusion

Our results on simple datasets validate this approach to posterior inference over latent permutations. More complex generative models with latent permutations can be approached using similar tools, a research direction we are presently exploring.

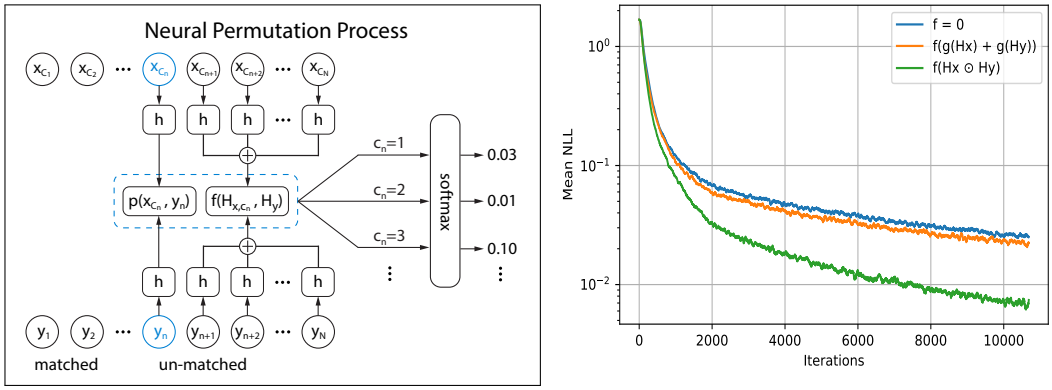

Figure 2: **Left: The NPP neural architecture. Right: Pairwise symmetric models.** The curves show mean training negative log-likelihood/iteration in the MNIST example. $f = 0$ is a baseline model, were we ignore the unassigned points in (9). The other two curves correspond to encoding the symmetry $p(y_n, x_{c_n}) = p(x_{c_n}, y_n)$ as $f(g(H_{x,c_n}) + g(H_y))$ or as $f(H_{x,c_n} \odot H_y)$.

.

References

Persi Diaconis. The Markov chain Monte Carlo revolution. *Bulletin of the American Mathematical Society*, 46(2):179–205, 2009.

Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

Shupeng Gui, Xiangliang Zhang, Pan Zhong, Shuang Qiu, Mingrui Wu, Jieping Ye, Zhengdao Wang, and Ji Liu. Pine: Universal deep embedding for graph nodes via partial permutation invariant set functions. *arXiv preprint arXiv:1909.12903*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. *arXiv preprint arXiv:1610.09900*, 2016.

Scott W Linderman, Gonzalo E Mena, Hal Cooper, Liam Paninski, and John P Cunningham. Reparameterizing the Birkhoff polytope for variational permutation inference. In *AISTATS*, 2018.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Byt3oJ-0W.

Brooks Paige and Frank Wood. Inference networks for sequential Monte Carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049, 2016.

Ari Pakman and Liam Paninski. Amortized Bayesian Inference for Clustering Models. *arXiv:1811.09747, BNP@NeurIPS 2018 Workshop*, 2018.

Nikhil Parthasarathy, Eleanor Batty, William Falcon, Thomas Rutten, Mohit Rajpal, E.J. Chichilnisky, and Liam Paninski. Neural Networks for Efficient Bayesian Decoding of Natural Images from Retinal Neurons. In *Advances in Neural Information Processing Systems 30*, pages 6434–6445. 2017.

Daniel Ritchie, Paul Horsfall, and Noah D Goodman. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016.

Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. Learning stochastic inverses. In *Advances in neural information processing systems*, pages 3048–3056, 2013.

Ruoxi Sun and Liam Paninski. Scalable approximate Bayesian inference for particle tracking data. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in neural information processing systems*, 2017.

Appendix A. Neural architectures in the examples

To train the networks in the examples, we used stochastic gradient descent with ADAM (Kingma and Ba, 2015), with learning rate 10^{-4}. The number of samples in each mini-batch were: 1 for $p(N)$, 1 for $p(c_{1:N})$, 64 for $p(\mathbf{x}, \mathbf{y}|c_{1:N})$. The architecture of the functions in each case were:

Permutations 2D

- h: MLP [2-64-64-64-256] with ReLUs
- f: MLP [512-64-64-64-1] with ReLUs

Permutations MNIST

- h: 2 layers of [convolutional + maxpool + ReLU] + MLP [256-256] with ReLUs
- f: same as above