Discrete Neural Processes

Ari Pakman, Yueqi Wang, Catalin Mitelut, JinHyung Lee, Liam Paninski

Columbia University

Abstract

Many data generating processes involve latent random variables over discrete combinatorial spaces whose size grows factorially with the dataset. In these settings, existing posterior inference methods can be inaccurate and/or very slow. In this work we develop methods for efficient amortized approximate Bayesian inference over discrete combinatorial spaces, with applications to probabilistic clustering (such as Dirichlet process mixture models), random communities (such as stochastic block models) and random permutations. The approach exploits the exchangeability of the generative models and is based on mapping distributed, permutation-invariant representations of discrete arrangements into conditional probabilities. The resulting algorithms parallelize easily, yield iid samples from the approximate posteriors along with a probability estimate of each sample (a quantity generally unavailable using Markov Chain Monte Carlo) and can easily be applied to both conjugate and non-conjugate models, as training only requires samples from the generative model. As a scientific application, we present a novel approach to spike sorting for high-density multielectrode probes.

1 Introduction

Discrete latent random variables appear in a wide variety of statistical models. When these variables have a combinatorial nature (e.g. permutations, graphs, partitions) the state space grows factorially with the data size, making inference challenging.

Popular inference methods in these models fall into a few broad classes. First, we can attempt to compute a maximum a posteriori (MAP) point estimate. However, exploring the full posterior is crucial whenever there is irreducible uncertainty about the latent structure (or when many separate local optima exist), as is often the case in these models. Second, Markov Chain Monte Carlo (MCMC) methods for exploring the posterior \cite{1–4} are asymptotically accurate but time-consuming, with convergence that is difficult to assess. Models whose likelihood and prior are non-conjugate are particularly challenging, since in general in these cases the model parameters cannot be marginalized and must be kept as part of the state of the Markov chain. Finally, variational methods \cite{5–9} are typically much faster but do not come with accuracy guarantees.

In this work we propose a novel technique to perform approximate posterior inference in combinatorial spaces. While the details differ in each generative model, the common motif is to use neural networks to express posterior distributions of latent discrete variables in terms of fixed-dimensional, distributed data representations that respect the permutation symmetries imposed by the discrete variables.

The method can be applied to non-conjugate models, and is amortized in the sense that, after investing computational time in training a neural network with labeled samples from a particular generative model, we can obtain independent, parallelizable, approximate posterior samples of the discrete variables for any new set of observations of arbitrary size, with no need for expensive MCMC steps. Note that clustering and community detection are usually addressed as unsupervised learning tasks. Instead, we approach them as supervised learning of a posterior distribution from labeled samples.
We present our approach in three different settings: in Section 2 we study random clustering; in Section 3 random community graph models; and in Section 4 random permutations. In each case, we present experimental results to illustrate the method. In Section 5 we discuss quantitative evaluations of the new methods and in Section 6 we discuss related works. In Section 7 we present a neuroscientific application of this method to spike sorting for high-density multielectrode probes, and we close in Section 8 by discussing potential directions for future work.

2 Clustering

Probabilistic models for clustering [10] introduce random variables \( c_i \) denoting the cluster number to which the data point \( x_i \) is assigned, and assume a generating process of the form

\[
\begin{align*}
\alpha_1, \alpha_2 &\sim p(\alpha) \\
N &\sim p(N) \\
c_1 \ldots c_N &\sim p(c_1, \ldots, c_N | \alpha_1) \\
\mu_1 \ldots \mu_K | c_{1:N} &\sim p(\mu_1, \ldots, \mu_K | \alpha_2) \\
x_i &\sim p(x_i | \mu_{c_i}) \quad i = 1 \ldots N
\end{align*}
\]

Here \( \alpha_1, \alpha_2 \) are hyperparameters. The number of clusters \( K \) is a random variable, indicating the number of distinct values among the sampled \( c_i \)'s, and \( \mu_k \) denotes a parameter vector controlling the distribution of the \( k \)-th cluster (e.g., \( \mu_k \) could include both the mean and covariance of a Gaussian mixture component). We assume that the priors \( p(c_{1:N} | \alpha_1) \) and \( p(\mu_{1:K} | \alpha_2) \) are exchangeable,

\[
p(c_1, \ldots, c_N | \alpha_1) = p(c_{\sigma_1}, \ldots, c_{\sigma_N} | \alpha_1),
\]

where \( \{ \sigma_i \} \) is an arbitrary permutation of the indices, and similarly for \( p(\mu_{1:K} | \alpha_2) \). Examples of this setting include Mixtures of Finite Mixtures [11] and many Bayesian nonparametric models, such as Dirichlet process mixture models (DPMM); see [12] for a recent overview.

Given \( N \) data points \( \mathbf{x} = \{ x_i \} \), we are interested in iid sampling the \( c_i \)'s, using a decomposition

\[
p(c_{1:N} | \mathbf{x}) = p(c_1 | \mathbf{x})p(c_2 | c_1, \mathbf{x}) \ldots p(c_N | c_{1:N-1}, \mathbf{x}). \tag{1}
\]

Note that \( p(c_1 = 1 | \mathbf{x}) = 1 \), since the first data point is always assigned to the first cluster. While we might also be interested in the hidden variables \( \mu_k \), the reason to focus on the discrete variables \( c_i \)'s is that given samples from them, it is generally relatively easy to obtain posterior samples from \( p(\alpha_1 | c_{1:N}) \) and \( p(\mu_k, \alpha_2 | \mathbf{x}, c_{1:N}) \).

We would like to model all the factors in (1) in a unified way, with a generic factor given by

\[
p(c_n | c_{1:n-1}, \mathbf{x}) = \frac{p(c_1 \ldots c_n, \mathbf{x})}{\sum_{c'_n} p(c_1 \ldots c'_n, \mathbf{x})}. \tag{2}
\]

Here we assumed that there are \( K \) unique values in \( c_{1:n-1} \), and therefore \( c_n \) can take \( K + 1 \) values, corresponding to \( x_n \) joining any of the \( K \) existing clusters, or forming its own new cluster.

We are interested in approximating (2):

\[
p(c_n | c_{1:n-1}, \mathbf{x}) \approx p_\theta(c_n | c_{1:n-1}, \mathbf{x}), \tag{3}
\]

where \( p_\theta \) is parameterized by a flexible model such as a neural network that takes as inputs \( (c_{1:n-1}, \mathbf{x}) \), then extracts features and combines them nonlinearly to output a probability distribution on \( c_n \). Critically, we will design the network to enforce the highly symmetric structure of the lhs of (3).

To make this symmetric structure more transparent, and in light of the expression (2), let us consider the joint distribution of the assignments of the first \( n \) data points,

\[
p(c_1 \ldots c_n, \mathbf{x}). \tag{4}
\]

A neural representation of this quantity should respect the permutation symmetries imposed on the \( x_i \)'s by the values of \( c_{1:n} \). Therefore, our first task is to build symmetry invariant representations of the observations \( \mathbf{x} \). The general problem of constructing such symmetry-invariant encodings was discussed recently in [13]; to adapt this approach to our context, we consider three distinct permutation symmetries:
where we have introduced a new real-valued function \( p \) potentially train a neural network to approximate \( \phi \) conjugate. If we can take an unlimited number of samples from the generative model, we can minimize the expected negative log-likelihood, \( \hat{\theta} \) In order to learn the parameters of the neural networks, we use stochastic gradient descent to minimize the expected negative log-likelihood,

\[
\mathbb{E}_{p(N)}\mathbb{E}_{p(c_{1:N}, x)} \left[ \sum_{n=2}^{N} \log p_\theta(c_n|c_{1:n-1}, x) \right].
\]

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

**Permutations within a cluster:** \( \Phi \) is invariant under permutations of \( x_i \)'s in the same cluster. For each of the \( K \) clusters that have been sampled so far, we define the encoding

\[
H_k = \sum_{i: c_i = k} h(x_i) \quad k = 1 \ldots K, \quad h : \mathbb{R}^d \rightarrow \mathbb{R}^{dh},
\]

which is clearly invariant under permutations of \( x_i \)'s in the same cluster. As we show in Sup. Material A, when \( p(x|\mu) \) belongs to an exponential family with sufficient statistics \( t(x) \), it is sufficient to set \( h(x) = (1, t(x)) \). Otherwise \( h \) is a function we learn from data.

**Permutations between clusters:** \( \Phi \) is invariant under permutations of the cluster labels. In terms of the within-cluster invariants \( H_k \), this symmetry can be captured by

\[
G = \sum_{k=1}^{K} g(H_k), \quad g : \mathbb{R}^{dh} \rightarrow \mathbb{R}^{dv}.
\]

**Permutations of the unassigned data points:** \( \Phi \) is also invariant under permutations of the \( N - n \) unassigned data points. This can be captured by

\[
Q = \sum_{i=n+1}^{N} q(x_i), \quad q : \mathbb{R}^d \rightarrow \mathbb{R}^{dv}.
\]

Note that \( G \) and \( Q \) provide fixed-dimensional, symmetry-invariant representations of the assigned and non-assigned data points, respectively, for any number of \( N \) data points and \( K \) clusters. Encodings of this form were shown in [13] to lead to arbitrarily accurate approximations of symmetric functions.

Now, each of the \( K + 1 \) possible values for \( c_n \) corresponds to \( h(x_n) \) appearing in one particular \( H_k \) in (5), and yields a separate vector \( G_k \) in (6). In terms of the \( G_k \)'s and \( Q \), we propose to model (2) as

\[
p_\theta(c_n = k|c_{1:n-1}, x) = \frac{e^{f(G_k, Q)}}{\sum_{k'=1}^{K+1} e^{f(G_{k'}, Q)}} \quad k = 1 \ldots K + 1,
\]

where we have introduced a new real-valued function \( f \).

In eq. (8), \( \theta \) denotes the parameters in the functions \( h, g, q \) and \( f \); we represent these functions with neural networks (unless \( h \) is already known, as in the exponential family setting). By storing and updating \( G \) and \( Q \) for successive values of \( n \), the computational cost of a full i.i.d. sample of \( c_{1:N} \) is \( O(NK) \), the same as a single Gibbs sweep. See Algorithm 1 in the Sup. Material for details; we term this approach the Neural Clustering Process (NCP).

In order to learn the parameters \( \theta \) of the neural networks, we use stochastic gradient descent to minimize the expected negative log-likelihood,
Figure 2: Clustering of MNIST data. We trained a generative clustering model on the full MNIST training dataset (see text for details), then sampled $N = 20$ observations from the MNIST test set (top row) and applied the NCP to these observations. The five rows below the observations show five samples of $c_{1-20}$ from the NCP posterior of these 20 images, with their corresponding probabilities, each capturing some ambiguity suggested by the form of particular digits.

2.1 Examples

Here we illustrate the method by presenting two examples. In the Sup. Material we provide details of the network architectures used in all the examples of the paper.

Clustering in 2D Gaussian models: We consider a DPMM clustering model for 2D points. The generative model is

$$
\begin{align*}
N & \sim \text{Uniform}[5, 100] \\
c_{1:N} & \sim \text{CRP}(\alpha) \\
\mu_k & \sim N(0, \sigma_\mu^2 1_2) \quad k = 1 \ldots K \\
x_i & \sim N(\mu_{c_i}, \sigma_{x_i}^2 1_2) \quad i = 1 \ldots N
\end{align*}
$$

where CRP stands for the Chinese Restaurant Process, with concentration parameter $\alpha = 0.7$, $\sigma_\mu = 10$, and $\sigma_{x_i} = 1$. Figure 1 shows that the NCP captures the posterior uncertainty inherent in clustering this data.

Clustering of MNIST digits: We consider next a DPMM over the empirical distribution of digits from the MNIST dataset. The generative model is

$$
\begin{align*}
N & \sim \text{Uniform}[5, 100] \\
\text{CRP}_{10}(\alpha) & \\
l_k & \sim \text{Unif}[0, 9] \quad \text{without replacement.} \\
x_i & \sim \text{Unif[MNIST digits with label } l_{c_i}] \quad i = 1 \ldots N
\end{align*}
$$

where CRP$_{10}$ is a Chinese Restaurant Process truncated to up to 10 clusters, with $\alpha = 0.7$, $d_x = 28 \times 28$. Figure 2 shows that posterior samples from the estimated model correctly capture the shape ambiguity of some of the digits. Note that in this case the generative model has no analytical expression (and therefore is non-conjugate), but this presents no problem; a generative model that we can sample from is all we need for training.

3 Communities

Our next setting has a similar prior as above over cluster labels, but the observation model is more challenging:

$$
\begin{align*}
\alpha, N & \sim p(\alpha), p(N) \\
c_1 \ldots c_N & \sim p(c_1, \ldots, c_N | \alpha) \\
\phi_{k_1, k_2} & \sim p(\phi | \beta) \quad k_1 \leq k_2 \\
x_{i,j} & \sim \text{Bernoulli}(\phi_{c_i, c_j}) \quad i \leq j \quad i, j = 1 \ldots N
\end{align*}
$$

where $k_1, k_2 = 1 \ldots K$. The prior $p(c_{1:n} | \alpha)$ can be any of the priors used for clustering above, and the observations $x_{i,j}$ represent now the presence or absence of an edge in a graph of $N$ vertices. We focus on the symmetric graph case here, so $\phi_{k_1, k_2} = \phi_{k_2, k_1}$ and $x_{i,j} = x_{j,i}$. We use a Beta model
For \( p(\phi|\beta) \), but other choices are possible. These models include stochastic block models \([14, 15]\) and the single-type Infinite Relational Model \([16, 17]\).

In principle posterior inference in this case can proceed similarly to the clustering case, by considering \( N \) particles, each given by a row of the adjacency matrix \( x_i = (x_{i1}, \ldots, x_{iN}) \). But we should be careful when encoding these particles. Consider the situation when the values of \( c_{1:n} \) have been assigned. Encoding with a generic function \( h(x_i) \) would ignore the permutation symmetries present among the components of \( x_i \), i.e., the columns of the matrix \( x_{i,j} \), as a result of the \( c_{1:n} \) assignments. These symmetries are the same three symmetries discussed above for clustering models. Moreover, a fixed function \( h(x_i) \) would not be able to accommodate the fact that the length of \( x_i \) changes with the size \( N \) of the dataset. In Sup. Material B we present the details of an invariant encoding that respects all these requirements. We call our approach Neural Block Process (NBP). Figure 3 shows that posterior samples from this model again capture the ambiguities inherent in clustering the observed graph data.

\[ c_i \in \{1, \ldots, N\}, \quad c_i \neq c_j \text{ for } i \neq j \]

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 \( x = \{x_i\} \), \( y = \{y_i\} \), we are interested in sampling the posterior of the permutations \( c_{1:N} \), using a factorization similar to (1). As we show in Sup. Material C, each factor can be represented as

\[ p_\theta(c_n|c_{1:n-1}, x, y) = \frac{p(x_{c_n}, y_n) e^{R(x_{c_n}, y_n)}}{\sum_{c'_n} p(x_{c'_n}, y_n) e^{R(x_{c'_n}, y_n)}}, \]

where \( x_{c_n} = \{x_{c_i}\}_{i=n+1}^N \), \( y_n = \{y_i\}_{i=n+1}^N \) correspond to all the pairs not yet matched. The function \( R(x_{c_n}, y_n) \) is invariant under separate permutations of the components of \( x_{c_n} \) and \( y_n \). Moreover, if the \( x_i \)'s and \( y_i \)'s belong to the same space and \( p(x_{c_n}, y_n) = p(y_n, x_{c_n}) \), then \( R(x_{c_n}, y_n) = R(y_n, x_{c_n}) \) and \( e^{R(x_{c_n}, y_n)} \) estimates, up to a multiplicative constant, the permanent of the symmetric matrix with entries \( p(y_i, x_{c_j}) \) for \( i, j \geq n + 1 \).

Note that (11) does not depend on \( \{x_{c_i}\}_{i=1}^{n-1} \) or \( \{y_i\}_{i=1}^{n-1} \). The density \( p(x_{c_n}, y_n) \) can be either known in advance, or a parametrized function to be learned. We call this approach the Neural Permutation Process (NPP). In Figure 4 we present two examples:

**Noisy pairs in 2D**: the generative model is

\[ x_{c_i} \sim N(0, 3) \quad y_i \sim N(x_{c_i}, 0.61) \quad i = 1 \ldots N. \]
Observed 25 Pairs
Prob: 0.07
Prob: 0.06
Prob: 0.05
Prob: 0.04
Prob: 0.04

Observed 6 Pairs
NPP: 0.91   Exact: 0.93
NPP: 0.07   Exact: 0.06

Figure 4: Neural Permutation Process. Left: Noisy pairs in 2D. \( N = 25 \) pairs from the model (12), 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 (13), given ten pairs of distinct MNIST digits. Note that the samples capture well the ambiguity between digits ‘4’ and ‘9’.

MNIST digits: the generative model is
\[
y_i, x_{c_i} \sim \text{Unif}[\text{Pairs of MNIST digits with same label}] \quad i = 1 \ldots N.
\] (13)

Both cases illustrate how a probabilistic approach captures the ambiguities in the observations.

5 Quantitative evaluations and diagnostics

The above examples provide strong qualitative evidence that our approximations to the true posterior distributions in these models are capturing the uncertainty inherent in the observed data. But we would like to go further and ask quantitatively how well our approximations match the exact posterior. Unfortunately, for sample sizes much larger than \( N = O(10) \) it is impossible to compute the exact posterior in these models. Nonetheless, there are several quantitative metrics we can examine to check the accuracy of the model output.

Global symmetry from exchangeability: Our results relied on \( p(c_{1:N} | \alpha_1) \) being exchangeable, which in turn implies exchangeability of the joint posterior (11). But this is not explicit in the rhs of (11), where a particular order is chosen for the expansion. If our model learns the conditional probabilities correctly, this symmetry should be (approximately) satisfied, and this can be monitored during training, as we show in the Sup. Material.

Estimated vs. Analytical Probabilities: Some conditional probabilities can be computed analytically and compared with the estimates output by the network. We present such a comparison for the NPP in Figure 4 upper-right, and for the NCP in Figure 5 upper-right. In both cases, the estimated probabilities are in close agreement with their exact values.

Geweke’s Tests: A popular family of tests that check the correctness of MCMC implementations (18) can also be applied in our case: verify the (approximate) identity between the prior \( p(c_{1:N}) \) and
\[
p_\theta(c_{1:N}) \equiv \int dx p_\theta(c_{1:N} | x) p(x),
\] (14)

where \( p(x) \) is the marginal from the generative model. Figure 5 shows such a comparison for the 2D Gaussian DPM from Section 2.1 showing excellent agreement.

Comparison with MCMC: Our method has two main advantages over MCMC approaches. Firstly, it gives a probability estimate for each sample, in general unavailable in MCMC. Secondly, an extremely higher efficiency from the parallelization of iid samples. For example, in the Gaussian 2D example in eq (10), in the clock time a naive collapsed Gibbs sampler produces one correlated sample, our method produces more than 100 iid approximate samples.
6 Related Works

Many works have studied posterior inference in the models discussed. Our work differs from previous approaches in its use of neural networks to explicitly approximate the posterior over combinatorial discrete spaces in order to obtain iid approximate samples.

MAP: The work [19] surveys deep learning techniques for combinatorial optimization. Functions over sets: Permutation-invariant neural architectures have been explored recently in [20–23]. The representation of a set via a sum (or mean) of encoding vectors was also used in [13, 24–28]. Amortized inference: Related approaches were explored in Bayesian networks [29], sequential Monte Carlo [30], probabilistic programming [31, 32] and particle tracking [33].

Clustering: The works [34–36] review neural network-based clustering, and [37] studies biologically inspired online clustering. Clustering based on information-theoretic criteria was studied in [38–40]. Amortized clustering for a fixed-size Gaussian mixture was studied in [23]. For a clustering technique motivated by spike sorting, see [41]. Communities: The work [42] reviews relational models with Bayesian non-parametric priors. For SBMs, posterior inference is studied in [43–45]. Neural architectures for communities in graphs have been studied in [46] as a classification problem for every node. Permutations: Posterior over random permutations have been studied using MCMC techniques in [3], and variational methods in [9].

7 Application: multichannel spike sorting with NCP

We close with an application to a challenging neuroscientific problem: clustering of electrophysiological waveforms (“spikes”) recorded from multi-electrode arrays (MEAs) — aka spike sorting (Fig. 6 left panel). As MEAs have grown in scale and popularity recently there has been new urgency in solving the spike sorting problem [47–50]. Empirically, many existing spike sorting packages do a good job sorting spikes from high-SNR cells (e.g. peak-to-peak of spike > 8-10 x standard
Figure 6: **Multi-electrode spike sorting with NCP.** *Left:* A detection module isolates observed multi-channel spike waveforms (left) and isolated waveforms are clustered by NCP (right; scale bar represents 5 x standard deviation (SD) of signal). *Right:* Multiple sample cluster configurations from the NCP posterior, each indicating a visually plausible clustering of the data: Example #1: spikes with average amplitude = 3.1 x SD are assigned to either a single cluster or two clusters; Example #2: assignment for spikes with average ptp = 3.5 x SD (scale bar = 5 x SD).

deviation of the signal), but there remains significant opportunity to improve performance in the lower-SNR regime, where there are many more neurons but increased uncertainty about cluster identity. Fully-Bayesian clustering approaches have been proposed to properly handle this uncertainty [51][52], but these approaches have not yet been popular in practice, largely because previous efforts require expensive Gibbs sampling and therefore do not scale well to modern large datasets. Thus the NCP approach proposed above — which explicitly aims to provide scalable Bayesian inference for challenging clustering problems — is well-suited to this task. In addition, neuroscientists typically sort many statistically similar datasets, meaning the cost of training the model can be highly amortized, because we can train the NCP network once and then reuse it on datasets from the same setup.

To apply the NCP approach here we need to start by specifying the generative model. Somewhat surprisingly, a very simple generative model already leads to promising performance. As in [53], on a training set of real data we simply collect (1) a set of mean spike shapes (obtained by running an existing spike sorting package, followed by very light manual curation to discard any obvious “garbage” templates) and (2) the noise covariance estimated in snippets of data where no spikes were detected; then our generative model simply adds Gaussian noise with the covariance from (2) to real spike shapes sampled from (1). In Sup. Material E we provide a detailed description of the training data preparation and the neural architecture used.

On the real data test set, NCP uncovered dozens of neurons not identified by a state-of-the-art sorting algorithm [47] (see Sup. Material E for details). Critically, NCP is able to separate spike clusters with amplitude as low as 3 × the standard deviation of the noise into plausible units that are not mere scaled version of each other but have distinct shapes on different channels (see Figure 6, right panel). These preliminary results show substantial promise for incorporating NCP into a production-scale spike sorting pipeline.

Finally, we describe in Sup. Material F a generalization of NCP to handle data in which the per-cluster parameters (e.g., the cluster means) are nonstationary in time, as a step towards addressing the important problem of spike sorting in the presence of electrode drift [54–56].

8 **Outlook**

Our method can potentially be applied to other discrete models [57], with the symmetries dictating the data encodings. Also, non-exchangeable priors exist for permutations [58][59] and clusters [60][65]. Recurrent neural networks may give useful encodings in these cases. Another direction is to use these methods for unsupervised learning. We plan to explore these ideas in future work.
Acknowledgements

We thank Scott Linderman, Ruoxi Sun, and Aaron Schein for helpful conversations, and E.J. Chichilnisky for sharing the data used in the spike sorting application. Work supported by the Simons Foundation, the DARPA NESD program, ONR N00014-17-1-2843, NIH/NIBIB R01 EB22913, NSF NeuroNex Award DBI-1707398 and The Gatsby Charitable Foundation.

References

[1] Radford M Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

[2] Sonia Jain and Radford M Neal. A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model. *Journal of computational and Graphical Statistics*, 13(1):158–182, 2004.

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

[4] Aaron F McDaid, Thomas Brendan Murphy, Nial Friel, and Neil J Hurley. Improved bayesian inference for the stochastic block model with application to large networks. *Computational Statistics & Data Analysis*, 60:12–31, 2013.

[5] David M. Blei and Michael I. Jordan. Variational Methods for the Dirichlet Process. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML ’04, 2004.

[6] Kenichi Kurihara, Max Welling, and Yee Whye Teh. Collapsed Variational Dirichlet Process Mixture Models. In *IJCAI*, volume 7, pages 2796–2801, 2007.

[7] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9(Sep):1981–2014, 2008.

[8] Michael Hughes, Dae Il Kim, and Erik Sudderth. Reliable and scalable variational inference for the hierarchical Dirichlet process. In *Artificial Intelligence and Statistics*, pages 370–378, 2015.

[9] 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.

[10] Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*, volume 84. Marcel Dekker, 1988.

[11] Jeffrey W Miller and Matthew T Harrison. Mixture models with a prior on the number of components. *Journal of the American Statistical Association*, 113(521):340–356, 2018.

[12] Abel Rodriguez and Peter Mueller. NONPARAMETRIC BAYESIAN INFERENCE. *NSF-CBMS Regional Conference Series in Probability and Statistics*, 9:i–110, 2013.

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

[14] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[15] Krzysztof Nowicki and Tom A B Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American statistical association*, 96(455):1077–1087, 2001.

[16] Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, volume 3, page 5, 2006.
[17] Zhao Xu, Volker Tresp, Kai Yu, and Hans-Peter Kriegel. Learning infinite hidden relational models. *Uncertainty in Artificial Intelligence (UAI2006)*, 2006.

[18] John Geweke. Getting it right: Joint distribution tests of posterior simulators. *Journal of the American Statistical Association*, 99(467):799–804, 2004.

[19] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d’hORIZON. *arXiv preprint arXiv:1811.06128*, 2018.

[20] Siamak Ravanbakhsh, Jeff Schneider, and Barnabás Póczos. Equivariance through parameter-sharing. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[21] Iryna Korshunova, Jonas Degrave, Ferenc Huszar, Yarin Gal, Arthur Gretton, and Joni Dambre. Bruno: A deep recurrent model for exchangeable data. In *Advances in Neural Information Processing Systems 31*, 2018.

[22] Benjamin Bloem-Reddy and Yee Whye Teh. Probabilistic symmetry and invariant neural networks. *arXiv preprint arXiv:1901.06082*, 2019.

[23] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer. *arXiv preprint arXiv:1810.00825*, 2018.

[24] Nicholas Guttenberg, Nathaniel Virgo, Olaf Witkowski, Hidetoshi Aoki, and Ryota Kanai. Permutation-equivariant neural networks applied to dynamics prediction. *arXiv preprint arXiv:1612.04530*, 2016.

[25] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016.

[26] Harrison Edwards and Amos Storkey. Towards a neural statistician. *ICLR*, 2017.

[27] Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Eslami. Conditional neural processes. In *International Conference on Machine Learning*, 2018.

[28] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. In *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

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

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

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

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

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

[34] K-L Du. Clustering: A neural network approach. *Neural networks*, 23(1):89–107, 2010.

[35] Elie Aljalgout, Vladimir Golkov, Yawar Siddiqui, and Daniel Cremers. Clustering with Deep Learning: Taxonomy and New Methods. *arXiv preprint arXiv:1801.07648*, 2018.

[36] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514, 2018.

[37] Cengiz Pehlevan, Alexander Genkin, and Dmitri B. Chklovskii. A clustering neural network model of insect olfaction. *bioRxiv*, 2018.
[38] Noam Slonim and Naftali Tishby. Document clustering using word clusters via the information bottleneck method. In Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, pages 208–215. ACM, 2000.

[39] Noam Slonim, Gurinder Singh Atwal, Gašper Tkačik, and William Bialek. Information-based clustering. Proceedings of the National Academy of Sciences, 102(51):18297–18302, 2005.

[40] Lev Faivishevsky and Jacob Goldberger. Nonparametric information theoretic clustering algorithm. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), pages 351–358, 2010.

[41] Jeremy F Magland and Alex H Barnett. Unimodal clustering using isotonic regression: Iso-split. arXiv preprint arXiv:1508.04841, 2015.

[42] Mikkel N Schmidt and Morten Morup. Nonparametric bayesian modeling of complex networks: An introduction. IEEE Signal Processing Magazine, 30(3):110–128, 2013.

[43] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. Physical Review E, 84(6):066106, 2011.

[44] Emmanuel Abbe. Community Detection and Stochastic Block Models. Foundations and Trends® in Communications and Information Theory, 14(1-2):1–162, 2018.

[45] S. L. van der Pas and A. W. van der Vaart. Bayesian community detection. Bayesian Analysis, 13(3):767–796, 2018.

[46] Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised Community Detection with Line Graph Neural Networks. ICLR, 2019.

[47] Marius Pachitariu, Nicholas Steinmetz, Shabnam Kadir, Matteo Carandini, and Kenneth D Harris. Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels. BioRxiv, page 061481, 2016.

[48] Jason E Chung, Jeremy F Magland, Alex H Barnett, Vanessa M Tolosa, Angela C Tooker, Kye Y Lee, Kedar G Shah, Sarah H Felix, Loren M Frank, and Leslie F Greengard. A fully automated approach to spike sorting. Neuron, 95(6):1381–1394, 2017.

[49] James J. Jun, Catalin Mitelut, Chongxi Lai, Sergey L. Gratiy, Costas A. Anastassiou, and Timothy D. Harris. Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. bioRxiv, 2017.

[50] David Carlson and Lawrence Carin. Continuing progress of spike sorting in the era of big data. Current opinion in neurobiology, 55:90–96, 2019.

[51] Frank Wood and Michael J Black. A nonparametric bayesian alternative to spike sorting. Journal of neuroscience methods, 173(1):1–12, 2008.

[52] David E Carlson, Joshua T Vogelstein, Qisong Wu, Wenzhao Lian, Mingyuan Zhou, Colin R Stoetzer, Daryl Kipke, Douglas Weber, David B Dunson, and Lawrence Carin. Multichannel electrophysiological spike sorting via joint dictionary learning and mixture modeling. IEEE Transactions on Biomedical Engineering, 61(1):41–54, 2013.

[53] Jin Hyung Lee, David E Carlson, Hooshmand Shokri Razaghi, Weichi Yao, Georges A Goetz, Espen Hagen, Eleanor Batty, EJ Chichilnisky, Gaute T Einevoll, and Liam Paninski. Yass: Yet another spike sorter. In Advances in Neural Information Processing Systems, pages 4002–4012, 2017.

[54] Ana Calabrese and Liam Paninski. Kalman filter mixture model for spike sorting of non-stationary data. Journal of neuroscience methods, 196(1):159–169, 2011.

[55] Kilosort2 - https://github.com/mouseland/kilosort2.

[56] Kevin Q Shan, Evgeniui V Lubenov, and Athanassios G Siapas. Model-based spike sorting with a mixture of drifting t-distributions. Journal of neuroscience methods, 288:82–98, 2017.
[57] Peter Orbanz and Daniel M Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):437–461, 2015.

[58] Douglas E Critchlow, Michael A Fligner, and Joseph S Verducci. Probability models on rankings. *Journal of mathematical psychology*, 35(3):294–318, 1991.

[59] Guy Lebanon and John Lafferty. Cranking: Combining rankings using conditional probability models on permutations. In *ICML*, volume 2, pages 363–370. Citeseer, 2002.

[60] Steven N MacEachern. Dependent Dirichlet processes. *Unpublished manuscript, Department of Statistics, The Ohio State University*, pages 1–40, 2000.

[61] Hanna Wallach, Shane Jensen, Lee Dicker, and Katherine Heller. An alternative prior process for nonparametric Bayesian clustering. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 892–899, 2010.

[62] David M Blei and Peter I Frazier. Distance dependent Chinese restaurant processes. *Journal of Machine Learning Research*, 12(Aug):2461–2488, 2011.

[63] Nicholas J Foti and Sinead A Williamson. A survey of non-exchangeable priors for Bayesian nonparametric models. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):359–371, 2015.

[64] Giuseppe Di Benedetto, François Caron, and Yee Whye Teh. Non-exchangeable random partition models for microclustering. *arXiv preprint arXiv:1711.07287*, 2017.

[65] Jun Lu, Meng Li, and David Dunson. Reducing over-clustering via the powered Chinese restaurant process. *arXiv preprint arXiv:1802.05592*, 2018.

[66] E. J. Chichilnisky and Rachel S. Kalmar. Functional asymmetries in on and off ganglion cells of primate retina. *Journal of Neuroscience*, 22(7):2737–2747, 2002.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

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

[69] Alex Graves. Sequence transduction with recurrent neural networks. *CoRR*, abs/1211.3711, 2012.

[70] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[71] Michael C Hughes and Erik Sudderth. Memoized online variational inference for dirichlet process mixture models. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1133–1141. Curran Associates, Inc., 2013.

[72] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.
A Neural Clustering Process for Exponential Families

The likelihood for an exponential family is given by
\[
p(x|\mu) = e^{\mu \cdot t(x) - \psi(\mu) + m(x_i)}
\]
where \( t(x) \) is a vector of sufficient statistics, and we defined
\[
h(x) = (1, t(x))
\]
\[
\lambda = (-\psi(\mu), \mu)
\]
Let us denote by \( K \) and \( K' \geq K \) the total number of distinct values in \( c_{1:n} \) and \( c_{1:N} \), respectively.

Consider the joint distribution
\[
p(c_{1:N}, x, \mu) = p(c_{1:N}) p(\mu) \prod_{k=1}^{K'} e^{\lambda_k \sum_{i: c_i = k} h(x_i)} \prod_{i=1}^{N} m(x_i)
\]
from which we obtain the marginal distributions
\[
p(c_{1:n}) = \sum_{c_{n+1} \cdots c_N} p(c_{1:N}, x)
\]
\[
= \sum_{c_{n+1} \cdots c_N} \int d\mu p(c_{1:N}) p(\mu) \prod_{k=1}^{K'} e^{\lambda_k \sum_{i: c_i = k} h(x_i)} \prod_{i=1}^{N} m(x_i)
\]
\[
= F(H_1, \ldots, H_K, h(x_{n+1}), \ldots, h(x_N)) \prod_{i=1}^{N} m(x_i)
\]
where we defined
\[
H_k = \sum_{i \leq n, c_i = k} h(x_i) \quad k = 1 \ldots K
\]
and \( H_k = 0 \) for \( k > K \). All the dependence of \( F \) on \( x_{1:n} \) is encoded in the \( H_k \)'s, and \( F \) is symmetric under separate permutations of the \( H_k \)'s and the \( h(x_i) \)'s for \( i > n \). Based on these symmetries we can approximate \( F \) as
\[
F \approx e^{f(G,Q)}
\]
modulo adding to \( f \) any function symmetric on all \( x_i \)'s, where
\[
G = \sum_{k=1}^{K} g(H_k)
\]
\[
Q = \sum_{i=n+1}^{N} q(x_i)
\]
In the conditional probability we are interested in,
\[
p(c_n|c_{1:n-1}, x) = \frac{p(c_{1:n}, x)}{\sum_{c_n} p(c_{1:n}, x)},
\]
the product of the \( m(x_i) \)'s in (22) cancels. Similarly, adding to \( f \) a function symmetric on all \( x_i \)'s leaves invariant our proposed approximation
\[
p_\theta(c_n = k|c_{1:n-1}, x) = \frac{e^{f(G_k, Q)}}{\sum_{k'=1}^{K+1} e^{f(G_{k'}, Q)}} \quad k = 1 \ldots K + 1.
\]
Algorithm 1 \(O(NK)\) Neural Clustering Process Sampling

1: \(h_i \leftarrow h(x_i)\) \(\quad i = 1 \ldots N\) \[\text{[Notation]}\]
2: \(Q \leftarrow \sum_{i=2}^{N} h_i\) \[\text{[Initialize unassigned set]}\]
3: \(H_1 \leftarrow h_1\) \[\text{[Create first cluster with } x_1\text{]}\]
4: \(G \leftarrow g(H_1)\)
5: \(K \leftarrow 1, c_1 \leftarrow 1\)
6: for \(n \leftarrow 2 \ldots N\) do
7: \(Q \leftarrow Q - h_n\) \[\text{[Remove } x_n\text{ from unassigned set]}\]
8: \(H_{K+1} \leftarrow 0\) \[\text{[We define } g(0) = 0\text{]}\]
9: for \(k \leftarrow 1 \ldots K + 1\) do
10: \(G \leftarrow G + g(H_k + h_n) - g(H_k)\) \[\text{[Add } x_n\text{]}\]
11: \(p_k \leftarrow e^f(G, Q, h_n)\)
12: \(G \leftarrow G - g(H_k + h_n) + g(H_k)\) \[\text{[Remove } x_n\text{]}\]
13: end for
14: \(p_k \leftarrow p_k / \sum_{k'=1}^{K+1} p_{k'}\) \[\text{[Normalize probabilities]}\]
15: \(c_n \sim p_k\) \[\text{[Sample assignment for } x_n\text{]}\]
16: if \(c_n = K + 1\) then
17: \(K \leftarrow K + 1\)
18: end if
19: \(G \leftarrow G - g(H_{c_n}) + g(H_{c_n} + h_n)\) \[\text{[Add point } x_n\text{]}\]
20: \(H_{c_n} \leftarrow H_{c_n} + h_n\)
21: end for
22: Return \(c_1 \ldots c_N\)

B Details of the Neural Block Process

Let us recall the generative model in this case,

\[
\begin{align*}
\alpha, N &\sim p(\alpha), p(N) \\
c_1 \ldots c_N &\sim p(c_1, \ldots, c_N | \alpha) \\
\phi_{k_1, k_2} &\sim \text{Beta}(\alpha, \beta) \quad k_1 \leq k_2 \\
x_{i,j} &\sim \text{Bernoulli}(\phi_{c_i, c_j}) \quad i \leq j, \quad i, j = 1 \ldots N
\end{align*}
\]

(29)

where \(k_1, k_2 = 1 \ldots K\). The prior \(p(c_{1:n} | \alpha)\) can be any exchangeable priors for clustering, and the observations \(x_{i,j}\) represent the presence or absence of an edge in a graph of \(N\) vertices. We set \(\phi_{k_1, k_2} = \phi_{k_2, k_1}\), and \(x_{i,j} = x_{j,i}\), and assume for notational convenience that \(x_{i,j} \in \{+1, -1\}\).

B.1 Encoding each row of the adjacency matrix

In principle posterior inference in this case can proceed similarly to the clustering case, by considering \(N\) particles, each given by a row of the adjacency matrix \(x_i = (x_{i,1}, \ldots, x_{i,N})\). But we should be careful when encoding these particles. Consider the situation when the values of \(c_{1:n}\) have been assigned. Encoding with a generic function \(b(x_i)\) would ignore the permutation symmetries present among the components of \(x_i\), i.e., the columns of the matrix \(x_{i,j}\), as a result of the \(c_{1:n}\) assignments. These symmetries are the same three symmetries discussed above for clustering models. Moreover, a fixed function \(b(x_i)\) would not be able to accommodate the fact that the length of \(x_i\) changes with the size \(N\) of the dataset.

Suppose that there are \(K\) clusters among the \(c_{1:n}\), each with \(s_k\) elements. In order to simplify the notation, let us assume that the \(N - n\) unassigned points all belong to an additional \((K + 1)\)-th cluster with \(s_{K+1} = N - n\), so we assume \(c_{n+1:N} = K + 1\), and we have \(\sum_{k=1}^{K+1} s_k = N\) and \(s_k = \sum_{j=1}^{N} \delta(c_j = k)\).
Now, in each row $x_i$, the number $s_k$ of elements in the $k$-th cluster can be split as

$$s_k = s_{i,k}^+ + s_{i,k}^-$$

where the encoding function is $s_{i,k}$

$$s_{i,k}^+ = \sum_{j=1}^{N} \delta(c_j = k)\delta(x_{i,j} = +1)$$

and note that both $s_{i,k}^+$ and $s_{i,k}^-$ are invariant under the symmetry of permuting the indices within cluster $k$.

**Example:** $N = 5$ and $x_1 = (1, +1, -1, +1, +1)$. If four assignments were made $c_1 = c_2 = 1$, $c_3 = c_4 = 2$, then $K = 2$ and $c_5 = 3$, and from $x_1$ we get $s_{1,1}^+ = 2, s_{1,1}^- = 0, s_{1,2}^+ = 1, s_{1,2}^- = 1, s_{1,3}^+ = 1, s_{1,3}^- = 0$. If we permute the columns 3 and 4, both from cluster $k = 2$, we get $x_1 = (+1, +1, +1, -1, +1)$, but all the $s_{i,1}$'s stay invariant.

Additional invariants can be obtained combining $s_{j,k}^+$ and $s_{j,k}^-$ across all rows $x_j$'s with $c_j = c_i$, such as

$$m_{c_i,k}^+ = \frac{1}{s_{c_i}} \sum_{j: c_j = c_i} s_{j,k}^+$$

and similarly $m_{c_i,k}^-$ and $v_{c_i,k}^-$. Note that these invariants are the same for all rows $x_j$ with $c_j = k$.

The motivation to consider them is that, if the partition corresponding to $c_{1:n}$ is correct, then for $i \leq n$ and $k \leq K$ we have $n_{c_i,k}^+ \simeq m_{c_i,k}^+$ since they are both estimators of the latent Bernoulli parameter $\phi_{c_i,k}$. For the same reason, if the partition is correct and those two estimators of $\phi_{c_i,k}$ are exact, then $v_{c_i,k}^- \simeq 0$. Similarly for $m_{c_i,k}^-$ and $v_{c_i,k}^-$. Then these values provide learning signals to the network that estimates the probability of the assignments $c_{1:n}$ being correct.

Therefore we propose to encode the components of $x_i$ belonging to cluster $k$ as

$$r_{i,k} = (s_{i,k}^+, m_{c_i,k}^+, v_{c_i,k}^+ , s_{i,k}^-, m_{c_i,k}^-, v_{c_i,k}^-) \in \mathbb{R}^6.$$  \hfill (32)

In order to preserve the symmetry of the first $K$ labels under permutations, we combine them as

$$t_i = \sum_{k=1}^K t(r_{i,k}) \in \mathbb{R}^{d_t}$$  \hfill (33)

where the encoding function is $t : \mathbb{R}^6 \rightarrow \mathbb{R}^{d_t}$. The encoding \hfill (32) of the unassigned components $x_{i,n+1:N}$ is kept separate and denoted as $q_i = r_{i,K+1}$.

In summary, each row $x_i$ of the adjacency matrix is represented by the fixed-dimensional pair $(t_i, q_i) \in \mathbb{R}^{d_t+6}$ in a way that respects the symmetries of the assignments $c_{1:n}$; permutations between members of a cluster, permutations of cluster labels and permutations among unassigned columns.

**B.2 Clustering the rows of the adjacency matrix**

We can proceed now as in regular clustering, encoding each cluster of $x_i$'s within $c_{1:n}$ as

$$H_k = \sum_{i: c_i = k} h(t_i, q_i) \in \mathbb{R}^{d_h} \quad k = 1 \ldots K,$$  \hfill (34)

and defining the permutation invariant, fixed-dimensional vectors

$$G = \sum_{k=1}^K g(H_k),$$  \hfill (35)

$$Q = \sum_{i=n+1}^N q(t_i, q_i).$$  \hfill (36)
In terms of these quantities, the conditional probabilities are defined as usual as

\[ p(y|c_n = k|c_{1:n-1}, x) = \frac{\exp(G_k, Q)}{\sum_{k'=1}^{K+1} \exp(G_{k'}, Q)} \]

for \( k = 1 \ldots K + 1 \), with \( h_n = h(t_n, q_n) \) and with \( G_k \) being the value of \( G \) for the different configurations. Compared to the regular clustering case, here we need to learn the additional function \( t \). We call our approach Neural Block Process (NBP).

C Details of the Neural Permutation Process

Let us recall the generating model in this case,

\[ N \sim p(N) \quad c_{1:N} \sim p(c_{1:N}) \quad x_{c_i}, y_i \sim p(x_{c_i}, y_i) \]

Here \( p(c_{1:N}) \) is a uniform distribution over permutations (and therefore exchangeable), with the random variable

\[ c_i \in \{1, \ldots, N\}, \quad c_i \neq c_j \text{ for } i \neq j \]

denoting that \( x_{c_i} \) is paired with \( y_i \).

Given two sets of \( N \) data points \( x = \{x_i\}, y = \{y_i\} \), we are interested in iid sampling the posterior of the \( c_i \)'s, using a decomposition

\[ p(c_{1:N}|x, y) = p(c_1|x, y)p(c_2|x, y)p(c_3|x, y) \ldots p(c_N|x_{c_1:N-1}, x, y); \]

note now that \( p(c_N|c_{1:N-1}, x, y) = 1 \), since the last point \( y_\text{N} \) is always matched with the last unmatched point among the \( x_i \)'s.

A generic factor in (38) is

\[ p(c_n|c_{1:n-1}, x, y) = \frac{p(c_1 \ldots c_n, x, y)}{\sum_{c_n'} p(c_1 \ldots c_n', x, y)} \]

where \( c_n \) takes values in \( \{k_i\}_{i=1}^{N-n+1} \). Consider first

\[ p(c_1, \ldots, c_n, x, y) = (N!)^{-1} p(x_{c_1}, y_1) \ldots p(x_{c_n}, y_n) \sum_{\{c_{n+1} \ldots c_N\}} \prod_{i=n+1}^{N} p(x_{c_i}, y_i) \]

where the factor \( (N!)^{-1} \) comes from the uniform prior \( p(c_{1:N}) \), and the sum above has \( (N-n)! \) terms. Inserting this expression into (39) gives

\[ p(c_n|c_{1:n-1}, x, y) = \frac{p(x_{c_n}, y_n) \sum_{\{c_{n+1} \ldots c_N\}} \prod_{i=n+1}^{N} p(x_{c_i}, y_i)}{\sum_{c_n'} p(x_{c_n'}, y_n) \sum_{\{c_{n+1} \ldots c_N\}} \prod_{i=n+1}^{N} p(x_{c_i}, y_i)} \]

Note that (41) 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 \( \{c_i\}_{i=n}^{N} \). We introduce now a function \( R \) in order to represent the sums above as

\[ e^{R(x_{c_n}, y_n)} \approx \sum_{\{c_{n+1} \ldots c_N\}} \prod_{i=n+1}^{N} p(x_{c_i}, y_i) \]

where \( x_{c_n} = \{x_{c_i}\}_{i=n+1}^{N}, y_n = \{y_i\}_{i=n+1}^{N} \) correspond to all the pairs not yet matched, and the function \( R(x_{c_n}, y_n) \) is invariant under separate permutations of the components of \( x_{c_n} \) and \( y_{c_n} \). Note that the rhs of (42) is the permanent of a matrix with components \( p(x_{c_i}, y_i) \). The symmetries of \( R(x_{c_n}, y_n) \) can be captured by defining

\[ G_y = \sum_{i=n+1}^{N} g(y_i), \]

\[ G_x = \sum_{i=n+1}^{N} g(x_{c_i}), \]

16
and

\[ R(x_{cn}, y_n) = R(G_x, G_y) \, . \]  

(45)

The expression \( p(y_n, x_{cn}) \) can be either known in advance, or represented by a parametrized function to be learned. Inserting (42) into (41) leads to

\[ p_\theta(c_n | c_1; n-1, x, y) = \frac{p(x_{cn}, y_n) e^{R(x_{cn}, y_n)}}{\sum_{c_n'} p(x_{c'n}, y_n) e^{R(x_{c'n}, y_n)}} \, . \]  

(46)

which is our proposed form for the factors of the Neural Permutation Process. Note that the identification (42) holds modulo the addition to \( R \) of a function symmetric on the components of \( y_n \), since such an addition cancels in (46).

Note finally that in case \( p(y_n, x_{cn}) = p(x_{cn}, y_n) \), which occurs, for example, when \( y_n \) is obtained from \( x_{cn} \) by adding Gaussian noise, there is an additional symmetry \( R(x_{cn}, y_n) = R(y_n, x_{cn}) \). In such a case, the invariance of \( R \) mentioned above is reduced to an additive constant. In our examples, we capture this last symmetry by defining

\[ s_1 = G_x + G_y, \]  

(47)

\[ s_2 = G_x \times G_y, \]  

(48)

\[ s_3 = (G_x - G_y)^2, \]  

(49)

where all the above operations are defined componentwise, and define

\[ R(G_x, G_y) = R(s_1, s_2, s_3) \, . \]  

(50)

D Monitoring global permutation invariance

As mentioned in Section 5, we must verify the symmetry of the posterior likelihood under global permutations of all the data points. We show such a check in Figure 7.

Figure 7: Global permutation invariance. Training curves for the NCP model of 2D Gaussians in Section 2. Each minibatch was evaluated for 8 random permutations of the order of the points in the dataset. Above: Mean of the NLL over the permutations. Below: NLL standard deviation/NLL mean. Note that the ratio is of order \( 10^{-2} \).
E Details of Multichannel Spike Sorting with NCP

E.1 Dataset

Raw data used in constructing synthetic, real, and hybrid test datasets for the spike sorting application was sourced from primate retinal recordings as described in [66]. Briefly, primate retina patches (2–4 mm in diameter) were placed on 512 channel extracellular microelectrodes arrays (MEAs) and electrical activity was recorded at 20,000Hz (in the current study only 49 channel sections of the 512 channel array were used). The MEA array consisted of a symmetric 2D hexagonal pattern with inter-channel distance of 60 µm. The preparation was stimulated with a white noise stimulus which enabled the computation of the receptive field (RF) of clustered neurons using the mean of the white-noise frame present at each spike in a cluster.

E.2 Data preprocessing and spike detection

Our spike detection module follows a method similar to [53]. Voltage time series on each electrode are band-pass filtered (300Hz-6000Hz) and then passed through a convolutional neural network, which takes as input a multichannel voltage signal \( x \in \mathbb{R}^{C \times R} \) from \( C \) channels (here \( C = 7 \) to reflect the centre channel of a spike plus its 6 neighboring channels) and \( R \) time steps. The output of the network is a binary time series \( y \in \{0, 1\}^R \) where “1”s indicate the the presence of an isolated or collided waveform was detected in the center channel of the input and “0”s representing noise or misaligned waveforms (e.g. spikes that peaked on the non-centre channel or that peaked on the centre-channel but were not temporally centered). See [53] for network training details.

After detection, each spike is assigned to the channel where it has the maximum peak-to-peak amplitude in its multi-channel waveform. We refer to this selected channel as the “center channel” for each spike. This procedure effectively partitions the spike train dataset into smaller center-channel-based datasets, i.e. a channel-0 dataset only contains spikes with the largest amplitude in channel 0.

For each spike, although its waveform is simultaneously recorded from many electrode channels, the most informative features often come from electrodes close to its center channel due to the biophysics of the somatic action potential. Setting the maximum distance of included features to be 70 µm results in using waveforms from the center channel and its 6 neighbors for each spike (channels on the borders of the electrode array are padded with zeros for the missing neighbor channels). Thus, the selected features are the spike waveforms from 7 channels, where each waveform contains 32 time steps surrounding the peak, making the input data for clustering a 7 x 32 array.

E.3 Synthetic training data generation

To train NCP for neural spike clustering, we produced synthetic training datasets using a generative model of noisy spike waveforms. The generative model aims to mimic the distribution of real spikes and is based on the following mixture of finite mixtures (MFM) [11] generative process:

\[
N \sim \text{Unif}[200, 500] \\
K \sim 1 + \text{Poisson}(\lambda) \\
\pi_1 \ldots \pi_K \sim \text{Dirichlet}(\alpha_1, \ldots, \alpha_K) \\
c_1 \ldots c_N \sim \text{Cat}(\pi_1, \ldots, \pi_K) \\
\mu_k \sim p(\mu), \quad k = 1 \ldots K \\
x_i \sim p(x_i|\mu_{c_i}, \Sigma_s \otimes \Sigma_t), \quad i = 1 \ldots N
\]

Here, \( N \) is the number of spikes. The number of clusters \( K \) is sampled from a shifted Poisson distribution. We choose \( \lambda = 2 \) so that each center-channel-based synthetic dataset has on average 3 clusters, similar to the mean in real datasets. \( \pi_1;K \) represents the proportion of each cluster and is sampled from a Dirichlet distribution with \( \alpha_1 = \ldots = \alpha_K = 1 \).

The spike templates \( \mu_k \in \mathbb{R}^{7 \times 32} \) are sampled uniformly from a reservoir of 957 templates obtained from previous spike sorting results of extracellular recordings (note: these training templates
Figure 8: **Synthetic training data example for the spike sorting application.** A synthetic training dataset containing 3 clusters and 500 spike waveforms.

were different from the templates and spikes used in any of the experiments in Section E.). The temporal location of each template was jittered to reflect the inherent randomness in the sampling times of the electrophysiological signals. To do this, we upsampled the templates 5-fold, added random time shifts (from a uniform $[-1, 1]$ distribution) and downsampled the templates.

Next, each spike waveform $x_\text{i}$ is sampled in (56) by adding to $\mu_{c_\text{i}}$ Gaussian noise with covariance given by the Kronecker product of spatial and temporal correlation matrices, estimated from segments of recording data where no spikes were detected. This method creates spatially and temporally correlated noise patterns similar to real data. Examples of single samples from our synthetic datasets are shown in Figure 8.

### E.4 Neural architectures for NCP spike sorting

For the functions $h$ and $q$ that encode the spike waveform for NCP, a 1-D convolutional neural network along the time axis was used, with each electrode channel treated as a feature dimension. Specifically, we used a 1-D residual network (ResNet) with 4 residual blocks, each having 32, 64, 128, 256 convolutional feature maps (kernel size = 3), respectively. The latter 3 residual blocks have a stride of 2 and the last block is followed by an averaged pooling layer and a final linear layer. The outputs of the 1-D ResNet encoder are the $h$ and $q$ vectors of NCP, i.e. $h_\text{i} = \text{ResNetEncoder}(x_\text{i})$, and similarly for $q_\text{i}$. We used $d_h = d_q = 256$. The rest of the NCP neural network ($g$ and $f$) is identical to the 2D Gaussian NCP.

With labeled synthetic data generated as described above, we trained the spike sorting NCP for 20000 iterations with a batch size of 32 to optimize the NLL loss. The Adam optimizer with a learning rate of 0.0001 was used. The learning rate was decreased by half at 10000 and 17000 iterations.

### E.5 Clustering real spikes using NCP

At inference time, we fed the featurized 7 x 32 array of real spike waveforms to NCP, and performed parallel posterior sampling of cluster labels. A parallel sampling size of 1500 was used initially. However, as the number of spikes increase, sampling becomes expensive. Thus, we alternatively employed beam search to efficiently find high-likelihood clustering results. A beam size of 150 was used in the results because it finds higher-likelihood clusters in much less inference time.

After the clustering step, for each single neuron cluster, a template was obtained as the average (multi-channel) shape of the spike waveforms in that cluster.

### E.6 Experimental Results

In this section we present the results of three different experiments, in which we compared NCP spike sorting against two other methods:

1. **vGMFM:** A Gaussian Mixture of Finite Mixtures, with inference performed using the variational method described in. Due to the high dimensionality of the data ($7 \times 32$),
we reduce the data to the first 5 PCA components on each channel. Also, independence across channels is assumed.

2. Kilosort: This is the spike sorting pipeline described in [47].

Experiment 1: Synthetic Data

In this experiment we compared NCP vs. vGMFM on 20 synthetic datasets each with 500, 1000, and 2000 samples with known cluster labels, generated as described in Section E.3. Since the true labels are known, we measured the quality of the clusters inferred using the Adjusted Mutual Information (AMI) [72], and summarized the comparison results in Figure 9. The AMI of NCP is on average 11% higher than that of vGMFM, showing that NCP performs better than vGMFM on synthetic datasets.

Experiment 2: Real Data

In this experiment we compared NCP against Kilosort on real data from a 49-channel, 20-minute retina recording where white noise stimulus was used. For this purpose, a simple spike sorting pipeline was built with NCP. We first detected and partitioned spikes from the raw recording into 49 center-channel-based datasets as described in Section E.2. Within each partitioned dataset, 2000 spikes were randomly sampled from all spikes detected in that channel and clustered using NCP, as illustrated in Figure 10. We considered clusters with less than 20 samples as coming from collisions (simultaneous spikes from multiple cells, leading to outliers) and thus discarded them. After clustering the subsampled data, we then assigned all detected spikes to one of the NCP clusters based on L2 distance between each spike and the cluster centers. Next, spike templates were computed using all 49 channels as the mean waveforms of all spikes assigned to that cluster. A template was removed if the maximum amplitude channel did not match the center channel of the clustered spikes; this serves to de-duplicate units recovered on a neighboring channel where the template size was higher. This gave us a set of multi-channel spike templates, and the assignments of detected spikes to one of the templates.

Since the real data does not come with ground truth cluster assignments, we used receptive fields (computed as described in [E.1]) to evaluate the quality of spike templates. A clearly demarcated receptive field usually indicates that the spike template belongs to a real neuron and that a sufficient number of spikes from the neuron are correctly recovered. We then compared the receptive fields and spike templates from the NCP pipeline with that from Kilosort. Pairs of templates were matched using L-infinity distance and pairs of receptive fields were matched using cosine distance. Unit-by-unit comparisons of randomly selected 55 pairs are shown in Figure 11 and a figure of all pairs attached as supplemental material. The NCP pipeline found 103 units with clear receptive fields, among which 48 were not found in Kilosort. Kilosort found 72 clear units and 17 of them were not found by NCP. In summary, this simple implementation of NCP performs at least as well as Kilosort on this real dataset, and finds a few more spike templates with clear receptive fields.
**Figure 10: Clustering spikes on real data.** 2000 spike waveforms from real data are clustered by NCP and vGMFM. **Upper left:** NCP spike clusters. Each column shows the spikes assigned to one cluster, plotted as overlaying traces and their average. Each row is one channel on the multielectrode array. **Upper right:** t-SNE visualization of the spike clusters. Clusters with less than 20 spikes are colored gray in the t-SNE plot. Note that the colored clusters are well-separated in the t-SNE projection. (To be clear, t-SNE is just used for visualization purposes here, and was not used as part of the clustering process.) **Bottom left and Bottom right:** vGMFM clusters of the same dataset. Note that the separation of the vGMFM clusters is less clear.

**Experiment 3: Hybrid Data**

In our final experiment, we compared NCP against both vGMFM and Kilosort on a hybrid recording with partial ground truth as in [47] to simulate clustering in a realistic setting where hundreds of neurons are present along side “ground-truth” neurons. 20 ground-truth spike templates were manually selected from a 20 minute, 49-channel training retinal recording and inserted at different timepoints into another 49-channel data set. Overall, this approach allows for testing the performance of clustering on a realistic recording where spikes collide and background noise does not necessarily have a Gaussian distribution. The three methods were evaluated based on the number of ground-truth injected templates that were recovered. The pipeline for vGMFM was built similarly to the NCP pipeline (see [E.6]). As summarized in Figure [12] NCP recovered 13 of the 20 injected templates, outperforming both Kilosort and vGMFM, which recovered 8 and 6 injected templates, respectively.

**E.7 Conclusions**

Our results show that using NCP for spike sorting matches or outperforms a current state of the art algorithm. On synthetic spike waveforms with ground truth clusters, NCP yields better clustering than vGMFM in terms of AMI score. On a real multi-electrode array dataset, we extracted many additional neurons with clear receptive fields from NCP-based clusters when compared to a leading sorting algorithm (i.e. Kilosort [47]). Finally, on hybrid data, NCP is able to recover more injected templates compared to both Kilosort and vGMFM. Together, NCP spike sorting shows promising improvement over existing methods.
Figure 11: Spike sorting on real data. Receptive fields of 55 randomly selected pairs of units recovered from Kilosort and NCP spike sorting. (Red boxes indicate units found by NCP; blue boxes by Kilosort.) Both approaches find the spikes with the biggest peak-to-peak (PTP) size. For smaller-PTP units often one sorting method finds a cell that the other sorter misses. NCP and KS find a comparable number of units with receptive fields here, with NCP finding a few more than KS; see text for details.
Figure 12: Spike sorting on hybrid data. 20 units are added into a real data stream and the pipelines are compared by counting how many of these ground-truth added units are recovered. Above: NCP, Kilosort, vGMFM find 13, 8, and 6 units, respectively. Below: Peak-to-peak signal size and firing rate of each injected unit. (Smaller units with smaller firing rates are generally more challenging.)

Importantly, there is further room to improve NCP for spike sorting. First, although our simple generative model of synthetic training data already leads to good performance, we could make the synthetic data more realistic by learning to correctly label collisions (i.e. secondary spikes close to the primary spike) by assigning them to the correct cluster or removing them. Second, our current ResNet spike encoder only takes as input the center channel and its six neighbors. Adding features from more distant channels where additional information is present could help distinguish neurons that have similar spike waveforms locally but differ on distant channels. Finally, an interesting direction for the future is to include NCP as part of a faster pipeline where the easy, big spikes are handled by some standard method and NCP focuses on the small spikes. We hope to pursue this direction in future work.
Figure 13: Neural Particle Tracking. Left: Time trajectories of 5 2D particles. Note that particles can appear or disappear at arbitrary times. Middle and right: Two posterior samples. Note that since only one particle is observed at each time, a particle not observed for some time leads to a possible ambiguity on the number of particles. (Best seen in color.)

F Particle tracking

Inspired by the problem of electrode drift [54–56], let us consider now a generative model given by

\[ c_t \sim p(c_t|c_1, \ldots, c_{t-1}) \quad t = 1, \ldots, T \]  
\[ \mu_{k,t} \sim p(\mu_{k,t}|\mu_{k,t-1}) \quad k = 1 \ldots K \quad t = 1, \ldots, T \]  
\[ x_t \sim p(x_t|\mu_{c_t,t}) \quad t = 1, \ldots, T \]  

In this model, a cluster corresponds to the points along the time trajectory of a particle, and (58) represents the time evolution of the cluster parameters. The cluster labels \( c_t \) indicate which particle is observed at time \( t \), and note that particles can in principle appear or disappear at any time.

To take the time evolution into account, we let particles influence one another with a weight that depends on their time distance. For this, let us introduce a time-decay constant \( b > 0 \), and generalize the NCP equations to

\[ H_{k,t} = \sum_{t' = 1: c_{t'} = k}^{t} e^{-b|t-t'|} h(x_{t'}) \quad k = 1 \ldots K, \]  
\[ G_t = \sum_{k = 1}^{K} g(H_{k,t}) , \]  
\[ Q_t = \sum_{t' = t+1}^{T} e^{-b|t-t'|} q(x_{t'}). \]  

The conditional assignment probability for \( c_t \) is now

\[ p_{\theta}(c_t = k|c_{1:t-1}, x) = \frac{e^{f(G_{k,t}, Q_t)}}{\sum_{k'=1}^{K+1} e^{f(G_{k',t}, Q_t)}} \]  

for \( k = 1 \ldots K + 1 \). The time-decay constant \( b \) is learnt along with all the other parameters. We can also consider replacing \( e^{-b|t-t'|} \) with a general distance function \( e^{-d(|t-t'|)} \). Figure 13 illustrates this model in a simple 2D example. We call this approach Neural Particle Tracking.

G Neural architectures in the examples

To train the networks in the examples, we used stochastic gradient descent with ADAM [68], 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 \( p(x|c_{1:N}) \). The architecture of the functions in each case were:
Clusters: 2D Gaussians
- $q$: MLP [2-64-64-256] with ReLUs
- $g$: MLP [3-128-128-128-256] with ReLUs
- $f$: MLP [512-128-128-128-1] with ReLUs

Clusters: MNIST
- $h$: 2 layers of [convolutional + maxpool + ReLU] + MLP [256-256] with ReLUs
- $q$: same as $h$
- $g$: MLP [256-128-128-128-256] with ReLUs
- $f$: same as above

Communities: IRL
- $t$: MLP [6-64-64-256] with ReLUs
- $h$: MLP [262-64-64-256] with ReLUs
- $q$: MLP [262-64-64-256] with ReLUs
- $g$: MLP [256-64-64-256] with ReLUs
- $f$: MLP [512-64-64-64-1] with ReLUs

Permutations: 2D
- $g$: MLP [2-64-64-256] with ReLUs
- $R$: MLP [768-64-64-64-1] with ReLUs

Permutations: MNIST
- $g$: same as $h$ for MNIST clusters.
- $R$: same as above