Differentiable Random Partition Models

Anonymous Author(s)
Affiliation
Address
`email`

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

A Preliminaries

A.1 Hypergeometric Distribution

This part is largely based on Sutter et al. [2023].

Suppose we have an urn with marbles in different colors. Let $K \in \mathbb{N}$ be the number of different classes or groups (e.g. marble colors in the urn), $\boldsymbol{m} = [m_1, \dots, m_K] \in \mathbb{N}^K$ describe the number of elements per class (e.g. marbles per color), $N = \sum_{k=1}^{K} m_K$ be the total number of elements (e.g. all marbles in the urn) and $n \in \{0, \dots, N\}$ be the number of elements (e.g. marbles) to draw. Then, the multivariate hypergeometric distribution describes the probability of drawing $\boldsymbol{n} = [n_1, \dots, n_K] \in \mathbb{N}^K$ marbles by sampling without replacement such that $\sum_{k=1}^{K} n_k = n$, where n_k is the number of drawn marbles of class k.

In the literature, two different versions of the noncentral hypergeometric distribution exist, Fisher's [Fisher, 1935] and Wallenius' [Wallenius, 1963, Chesson, 1976] distribution. Sutter et al. [2023] restrict themselves to Fisher's noncentral hypergeometric distribution due to limitations of the latter [Fog, 2008]. Hence, we will also talk solely about Fisher's noncentral hypergeometric distribution.

Definition A.1 (Multivariate Fisher's Noncentral Hypergeometric Distribution [Fisher, 1935])**.** *A random vector $\boldsymbol{X}$ follows Fisher's noncentral multivariate distribution, if its joint probability mass function is given by*

$$P(\boldsymbol{N} = \boldsymbol{n}; \boldsymbol{\omega}) = p(\boldsymbol{n}; \boldsymbol{\omega}) = \frac{1}{P_0} \prod_{k=1}^{K} \binom{m_k}{n_k} \omega_k^{n_k} \tag{12}$$

$$\textit{where} \quad P_0 = \sum_{(\eta_1, \dots, \eta_K) \in \mathcal{S}} \prod_{k=1}^{K} \binom{m_k}{\eta_k} \omega_k^{\eta_k} \tag{13}$$

The support S of the PMF is given by $S = \{\boldsymbol{n} \in \mathbb{N}^K : \forall k \quad n_k \leq m_k, \sum_{k=1}^{K} n_k = n\}$ and $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

The class importance $\boldsymbol{\omega}$ is a crucial modeling parameter in applying the noncentral hypergeometric distribution (see [Chesson, 1976]).

A.1.1 Differentiable MVHG

Their reparameterizable sampling for the differentiable MVHG consists of three parts:

1. Reformulate the multivariate distribution as a sequence of interdependent and conditional univariate hypergeometric distributions.
2. Calculate the probability mass function of the respective univariate distributions.
3. Sample from the conditional distributions utilizing the Gumbel-Softmax trick.

Following the chain rule of probability, the MVHG distribution allows for sequential sampling over classes k. Every step includes a merging operation, which leads to biased samples compared to groundtruth non-differentiable sampling with equal class weights $\boldsymbol{\omega}$. Given that we intend to use the differentiable MVHG in settings where we want to learn the unknown class weights, we do not expect a negative effect from this sampling procedure. For details on how to merge the MVHG into a sequence of unimodal distributions, we refer to Sutter et al. [2023].

The probability mass function calculation is based on unnormalized log-weights, which are interpreted as unnormalized log-weights of a categorical distribution. The interpretation of the class-conditional unimodal hypergeometric distributions as categorical distributions allows applying the Gumbel-Softmax trick [Jang et al., 2016, Maddison et al., 2017]. Following the use of the Gumbel-Softmax trick, the class-conditional version of the hypergeometric distribution is differentiable and reparameterizable. Hence, the MVHG has been made differentiable and reparameterizable as well. Again, for details we refer to the original paper [Sutter et al., 2023].

A.2 Distribution over Random Orderings

Yellott [1977] show that the distribution over permutation matrices $p(\pi; \boldsymbol{s})$ follows a Plackett-Luce (PL) distribution [Plackett, 1975, Luce, 1959], if and only of the perturbed scores $\tilde{\boldsymbol{s}}$ are sampled independently from Gumbel distributions with identical scales. For each item i, sample $g_i \sim \text{Gumbel}(0, \beta)$ independently with zero mean and and fixed scale β. Let $\tilde{\boldsymbol{s}}$ be the vector of Gumbel perturbed log-weights such that $\tilde{s}_i = \beta \log s_i + g_i$. Hence,

$$q(\tilde{s}_1 \geq \cdots \geq \tilde{s}_n) = \frac{s_1}{Z} \cdot \frac{s_2}{Z - s_1} \cdot \cdots \cdot \frac{s_n}{Z - \sum_{i=1}^{n-1} s_i} \tag{14}$$

We refer to Yellott [1977] or Grover et al. [2019] for the proof. However, Grover et al. [2019] provide only an adapted proof sketch from Yellott [1977]. The probability of sampling element i first is given by its score s_i divided by the sum of all weights in the set

$$q(\tilde{s}_i) = \frac{s_i}{Z} \tag{15}$$

For $z_i = \log s_i$, the right hand side of Equation (15) is equal to the softmax distribution $\text{softmax}(z_i) = \exp(z_i)/\sum_j \exp(z_j)$ as already described in [Xie and Ermon, 2019]. Hence, Equation (15) directly leads to the Gumbel-Softmax trick [Jang et al., 2016, Maddison et al., 2017].

A.2.1 Differentiable Sorting

In the main text of the paper we rely on a differentiable function $f_\pi(\tilde{\boldsymbol{s}})$, which sorts the resampled version of the scores $\boldsymbol{s}$

$$\pi = f_\pi(\tilde{\boldsymbol{s}}) = \text{sort}(\tilde{\boldsymbol{s}}) \tag{16}$$

Here, we summarise the findings from Grover et al. [2019] on how to construct such a differentiable sorting operator. As already mentioned in Section 2, there are multiple works on the topic [Prillo and Eisenschlos, 2020, Petersen et al., 2021, Mena et al., 2018], but we restrict ourselves to the work of Grover et al. [2019] as we see the differentiable generation of permutation matrices as a tool in our pipeline.

Corollary A.2 (Permutation Matrix [Grover et al., 2019])**.** *Let $\boldsymbol{s} = [s_1, \ldots, s_n]^T$ be a real-valued vector of length n. Let $A_{\boldsymbol{s}}$ denote the matrix of absolute pairwise differences of the elements of $\boldsymbol{s}$ such that $A_{\boldsymbol{s}}[i, j] = |s_i - s_j|$. The permutation matrix π corresponding to sort($\boldsymbol{s}$) is given by:*

$$\pi = \begin{cases} 1 & \text{if } j = \arg\max[(n + 1 - 2i)\boldsymbol{s} - A_{\boldsymbol{s}}\mathbb{1}] \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where $\mathbb{1}$ denotes the column vector of all ones.

As we know, the $\arg\max$ operator is non-differentiable which prohibits the direct use of Corollary A.2 for gradient computation. Hence, Grover et al. [2019] propose to replace the $\arg\max$ operator with softmax to obtain a continuous relaxation $\pi(\tau)$ similar to the GS trick [Jang et al., 2016, Maddison et al., 2017]. In particular, the ith row of $\pi(\tau)$ is given by:

$$\pi(\tau)[i, :] = \text{softmax}[(n + 1 - 2i)\boldsymbol{s} - A_{\boldsymbol{s}}\mathbb{1}/\tau] \tag{18}$$

where $\tau > 0$ is a temperature parameter. We adapted this section from Grover et al. [2019] and we also refer to their original work for more details on how to generate differentiable permutation matrices.

In this, work we remove the temperature parameter τ to reduce clutter in the notation. Hence, we only write π instead of $\pi(\tau)$, although it is still needed for the generation of the matrix π. For details on how we select the temperature parameter τ in our experiments, we refer to Appendix C.

B Detailed Derivation of the Differentiable Two-Stage Random Partition Model

B.1 Two-Stage Partition Model

We want to partition n elements $[n] = \{1, \ldots, n\}$ into K subsets $\{\mathcal{S}_1, \ldots, \mathcal{S}_K\}$ where K is *a priori* unknown.

Definition B.1 (Partition)**.** *A partition ρ of a set of elements $[n] = \{1, \ldots, n\}$ is a collection of subsets $(\mathcal{S}_1, \ldots, \mathcal{S}_K)$ such that*

$$\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_K = [n] \quad \textit{and} \quad \forall i \neq j : \; \mathcal{S}_i \cap \mathcal{S}_j = \emptyset \tag{19}$$

Put differently, every element i has to be assigned to precisely one subset $\mathcal{S}_k$. We denote the size of the k-th subset $\mathcal{S}_k$ as $n_k = |\mathcal{S}_k|$. Alternatively, we describe a partition ρ as an assignment matrix $Y = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K]^T \in \{0,1\}^{K \times n}$. Every row $\boldsymbol{y}_k \in \{0,1\}^{1 \times n}$ is a multi-hot vector, where $\boldsymbol{y}_{ki} = 1$ assigns element i to subset $\mathcal{S}_k$.

In this work, we propose a new two-stage procedure to learn partitions. The proposed formulation separately infers the number of elements per subset n_k and the assignment of elements to subsets $\mathcal{S}_k$ by inducing an order on the n elements and filling $\mathcal{S}_1, ..., \mathcal{S}_K$ sequentially in this order. See Figure 1 for an example.

Definition B.2 (Two-stage partition model)**.** *Let $\boldsymbol{n} = [n_1, \ldots, n_K] \in \mathbb{N}_0^K$ be the subset sizes in ρ, with $\mathbb{N}_0$ the set of natural numbers including 0 and $\sum_{k=1}^{K} n_k = n$, where n is the total number of elements. Let $\pi \in \{0,1\}^{n \times n}$ be a permutation matrix that defines an order over the n elements. We define the two-stage partition model of n elements into K subsets as an assignment matrix $Y = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K]^T \in \{0,1\}^{K \times n}$ with*

$$\boldsymbol{y}_k = \sum_{i=\nu_k+1}^{\nu_k+n_k} \boldsymbol{\pi}_i, \qquad \textit{where} \quad \nu_k = \sum_{\iota=1}^{k-1} n_\iota \tag{20}$$

such that $Y = [\{\boldsymbol{y}_k \mid n_k > 0\}_{k=1}^{K}]^T$.

Note that in contrast to previous work on partition models [Mansour and Schork, 2016], we allow $\mathcal{S}_k$ to be the empty set $\emptyset$. Hence, K defines the maximum number of possible subsets, not the effective number of non-empty subsets.

To model the order of the elements, we use a permutation matrix $\pi = [\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_n]^T \in \{0,1\}^{n \times n}$ which is a square matrix where every row and column sums to 1. This doubly-stochastic property of all permutation matrices π [Marcus, 1960] thus ensures that the columns of Y remain one-hot vectors. At the same time, its rows correspond to n_k-hot vectors $\boldsymbol{y}_k$ in Definition B.2 and therefore serve as subset assignment vectors.

Corollary B.3. *A two-stage partition model Y, which follows Definition B.2, is a valid partition satisfying Definition B.1.*

Proof. By definition, every row $\boldsymbol{\pi}_i$ and column $\boldsymbol{\pi}_j$ of π is a one-hot vector, hence every $\sum_{i=\nu_k+1}^{\nu_k+n_k} \boldsymbol{\pi}_i$ results in different, non-overlapping n_k-hot encodings, ensuring $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset \;\; \forall \;\; i, j$ and $i \neq j$. Further, since n_k-hot encodings have exactly n_k entries with 1, we have $\sum_{i=\nu_k+1}^{\nu_k+n_k} \sum_{j=1}^{n} \boldsymbol{\pi}_{ij} = n_k$. Hence, since $\sum_{k=1}^{K} n_k = n$, every element i is assigned to a $\boldsymbol{y}_k$, ensuring $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_K = [n]$. □

B.2 Two-Stage Random Partition Models

An RPM $p(Y)$ defines a probability distribution over partitions Y. In this section, we derive how to extend the two-stage procedure from Definition B.2 to the probabilistic setting to create a two-stage RPM. To derive the two-stage RPM's probability distribution $p(Y)$, we need to model distributions over $\boldsymbol{n}$ and π. We choose the MVHG distribution $p(\boldsymbol{n}; \boldsymbol{\omega})$ and the PL distribution $p(\pi; \boldsymbol{s})$ (see Section 3).

We calculate the probability $p(Y; \boldsymbol{\omega}, \boldsymbol{s})$ sequentially over the probabilities of subsets $p_{\boldsymbol{y}_k} := p(\boldsymbol{y}_k \mid \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s})$. $p_{\boldsymbol{y}_k}$ itself depends on the probability over subset permutations $p_{\bar{\pi}_k} := p(\bar{\pi} \mid n_k, \boldsymbol{y}_{<k}; \boldsymbol{s})$, where a subset permutation matrix $\bar{\pi}$ represents an ordering over n_k out of n elements.

Definition B.4 (Subset permutation matrix $\bar{\pi}$)**.** *A subset permutation matrix $\bar{\pi} \in \{0,1\}^{n_k \times n}$, where $n_k \leq n$, must fulfill*

$$\forall i \leq n_k : \; \sum_{j=1}^{n} \bar{\pi}_{ij} = 1 \quad \textit{and} \quad \forall j \leq n : \; \sum_{i=1}^{n_k} \bar{\pi}_{ij} \leq 1.$$

We describe the probability distribution over subset permutation matrices $p_{\bar{\pi}_k}$ using Definition B.4 and Equation (3).

Lemma B.5 (Probability over subset permutations $p_{\bar{\pi}_k}$)**.** *The probability $p_{\bar{\pi}_k}$ of any subset permutation matrix $\bar{\pi} = [\bar{\boldsymbol{\pi}}_1, \dots, \bar{\boldsymbol{\pi}}_{n_k}]^T \in \{0,1\}^{n_k \times n}$ is given by*

$$p_{\bar{\pi}_k} := p(\bar{\pi} \mid n_k, \boldsymbol{y}_{<k}; \boldsymbol{s}) = \prod_{i=1}^{n_k} \frac{(\bar{\pi}\boldsymbol{s})_i}{Z_k - \sum_{j=1}^{i-1} (\bar{\pi}\boldsymbol{s})_j} \tag{21}$$

where $\boldsymbol{y}_{<k} = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_{k-1}\}$, $Z_k = Z - \sum_{j \in \mathcal{S}_{<k}} s_j$ and $\mathcal{S}_{<k} = \bigcup_{j=1}^{k-1} \mathcal{S}_j$.

Proof. We provide the proof for $p_{\bar{\pi}_1}$, but it is equivalent for all other subsets. Without loss of generality, we assume that there are n_1 elements in $\mathcal{S}_1$. Following Equation (3), the probability of a permutation matrix $p(\pi; \boldsymbol{s})$ is given by

$$p(\pi; \boldsymbol{s}) = \frac{(\pi\boldsymbol{s})_1}{Z} \frac{(\pi\boldsymbol{s})_2}{Z - (\pi\boldsymbol{s})_1} \cdots \frac{(\pi\boldsymbol{s})_n}{Z - \sum_{j=1}^{n-1} (\pi\boldsymbol{s})_j} \tag{22}$$

At the moment, we are only interested in the ordering of the first n_1 elements. The probability of the first n_1 is given by marginalizing over the remaining $n - n_1$ elements:

$$p(\bar{\pi} \mid n_1; \boldsymbol{\omega}) = \sum_{\pi \in \Pi_1} p(\pi \mid \boldsymbol{s}) \tag{23}$$

where Π_1 is the set of permutation matrices such that the top n_1 rows select the elements in a specific ordering $\bar{\pi} \in \{0,1\}^{n_1 \times n}$, i.e. $\Pi_1 = \{\pi : [\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_{n_1}]^T = \bar{\pi}\}$. It follows

$$\begin{aligned}
p(\bar{\pi} \mid n_1; \boldsymbol{\omega}) &= \sum_{\pi \in \Pi_1} p(\pi \mid \boldsymbol{s}) && (24)\\
&= \sum_{\pi \in \Pi_1} \prod_{i=1}^{n} \frac{(\pi\boldsymbol{s})_i}{Z - \sum_{j=1}^{i-1} (\pi\boldsymbol{s})_j} && (25)\\
&= \prod_{i=1}^{n_1} \frac{(\bar{\pi}\boldsymbol{s})_i}{Z - \sum_{j=1}^{i-1} (\bar{\pi}\boldsymbol{s})_j} \sum_{\pi \in \Pi_1} \prod_{i=1}^{n-n_1} \frac{(\pi\boldsymbol{s})_{n_1+i}}{Z - \sum_{j=1}^{n_1} (\bar{\pi}\boldsymbol{s})_j - \sum_{j=1}^{i-1} (\bar{\pi}\boldsymbol{s})_j} && (26)\\
&= \prod_{i=1}^{n_1} \frac{(\bar{\pi}\boldsymbol{s})_i}{Z - \sum_{j=1}^{i-1} (\bar{\pi}\boldsymbol{s})_j} \sum_{\pi \in \Pi_1} \prod_{i=1}^{n-n_1} \frac{(\pi\boldsymbol{s})_{n_1+i}}{Z_1 - \sum_{j=1}^{i-1} (\bar{\pi}\boldsymbol{s})_j} && (27)
\end{aligned}$$

where $Z_1 = Z - \sum_{j=1}^{n_1} (\bar{\pi}\boldsymbol{s})_j$. It follows

$$p(\bar{\pi} \mid n_1; \boldsymbol{\omega}) = \prod_{i=1}^{n_1} \frac{(\bar{\pi}\boldsymbol{s})_i}{Z - \sum_{j=1}^{i-1} (\bar{\pi}\boldsymbol{s})_j} \tag{28}$$

□

Lemma B.5 describes the probability of drawing the elements $i \in \mathcal{S}_k$ in the order described by the subset permutation matrix $\bar{\pi}$ given that the elements in $\mathcal{S}_{<k}$ are already determined. Note that in a slight abuse of notation, we use $p(\bar{\pi} \mid n_k, \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s})$ as the probability of a subset permutation $\bar{\pi}$ given that there are n_k elements in $\mathcal{S}_k$ and thus $\bar{\pi} \in \{0,1\}^{n_k \times n}$. Additionally, we condition on the subsets $\boldsymbol{y}_{<k}$ and n_k, the size of subset $\mathcal{S}_k$. In contrast to the distribution over permutations matrices $p(\pi; \boldsymbol{s})$ in Equation (3), we take the product over n_k terms and have a different normalization constant Z_k. Although we induce an ordering over all elements i in Definition B.2, the probability $p_{\boldsymbol{y}_k}$ is invariant to intra-subset orderings of elements $i \in \mathcal{S}_k$.

Lemma B.6 (Probability distribution $p_{\boldsymbol{y}_k}$)**.** *The probability distribution over subset assignments $p_{\boldsymbol{y}_k}$ is given by*

$$p_{\boldsymbol{y}_k} := p(\boldsymbol{y}_k \mid \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s}) = p(n_k \mid n_{<k}; \boldsymbol{\omega}) \sum_{\bar{\pi} \in \Pi_{\boldsymbol{y}_k}} p(\bar{\pi} \mid n_k, \boldsymbol{y}_{<k}; \boldsymbol{s})$$

where $\Pi_{\boldsymbol{y}_k} = \{\bar{\pi} \in \{0,1\}^{n_k \times n} : \boldsymbol{y}_k = \sum_{i=1}^{n_k} \bar{\boldsymbol{\pi}}_i\}$ and $p(\bar{\pi} \mid n_k, \boldsymbol{y}_{<k}; \boldsymbol{s})$ as in Lemma B.5.

Proof. We can proof the statement of Lemma B.6 as follows:

$$
\begin{aligned}
p_{\boldsymbol{y}_k} &= p(\boldsymbol{y}_k \mid \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s}) \\
&= \sum_{n'_k} p(\boldsymbol{y}_k, n'_k \mid \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s}) && (29) \\
&= \sum_{n'_k} p(n'_k \mid \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s}) p(\boldsymbol{y}_k \mid n'_k, \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s}) && (30) \\
&= \sum_{n'_k} p(n'_k \mid n_{<k}; \boldsymbol{\omega}, \boldsymbol{s}) p(\boldsymbol{y}_k \mid n'_k, \boldsymbol{y}_{<k}; \boldsymbol{s}) && (31) \\
&= p(n_k \mid n_{<k}; \boldsymbol{\omega}, \boldsymbol{s}) p(\boldsymbol{y}_k \mid n_k, \boldsymbol{y}_{<k}; \boldsymbol{s}) && (32) \\
&= p(n_k \mid n_{<k}; \boldsymbol{\omega}) \sum_{\bar{\pi} \in \Pi_{\boldsymbol{y}_k}} p(\bar{\pi} \mid n_k, \boldsymbol{y}_{<k}; \boldsymbol{s}) && (33)
\end{aligned}
$$

Equation (29) holds by marginalization, where n'_k denotes the random variable that stands for the size of subset $\mathcal{S}_k$. By Bayes' rule, we can then derive Equation (30). The next derivations stem from the fact that we can compute $n_{<k}$ if $\boldsymbol{y}_{<k}$ is given, as the assignments $\boldsymbol{y}_{<k}$ hold information on the size of subsets $\mathcal{S}_{<k}$. More explicitly, $n_i = \sum_{j=1}^{n} y_{ij}$. Further, $\boldsymbol{y}_k$ is independent of $\boldsymbol{\omega}$ if the size n'_k of subset $\mathcal{S}_k$ is given, leading to Equation (31). We further observe that $p(\boldsymbol{y}_k \mid n'_k, \boldsymbol{y}_{<k}; \boldsymbol{s})$ is only non-zero, if $n'_k = \sum_{i=1}^{n} y_{ki} = n_k$. Dropping all zero terms from the sum in Equation (31) thus results in Equation (32). Finally, by Definition B.2, we know that $\boldsymbol{y}_k = \sum_{i=\nu_k+1}^{\nu_k+n_k} \boldsymbol{\pi}_i$, where $\nu_k = \sum_{\iota=1}^{k-1} n_\iota$ and $\pi \in \{0,1\}^{n \times n}$ a permutation matrix. Hence, in order to get $\boldsymbol{y}_k$ given $\boldsymbol{y}_{<k}$, we need to marginalize over all permutations of the elements of $\boldsymbol{y}_k$ given that the elements in $\boldsymbol{y}_{<k}$ are already ordered, which corresponds exactly to marginalizing over all subset permutation matrices $\bar{\pi}$ such that $\boldsymbol{y}_k = \sum_{i=1}^{n_k} \bar{\boldsymbol{\pi}}_i$, resulting in Equation (33). □

In Lemma B.6, we describe the set of all subset permutations $\bar{\pi}$ of elements $i \in \mathcal{S}_k$ by $\Pi_{\boldsymbol{y}_k}$. Put differently, we make $p(\boldsymbol{y}_k \mid \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s})$ invariant to the ordering of elements $i \in \mathcal{S}_k$ by marginalizing over the probabilities of subset permutations $p_{\bar{\pi}_k}$ [Xie and Ermon, 2019].

Using Lemmas B.5 and B.6, we propose the two-stage random partition $p(Y; \boldsymbol{\omega}, \boldsymbol{s})$. Since $Y = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K]^T$, we calculate $p(Y; \boldsymbol{\omega}, \boldsymbol{s})$, the PMF of the two-stage RPM, sequentially using Lemmas B.5 and B.6, where we leverage the PL distribution for permutation matrices $p(\pi; \boldsymbol{s})$ to describe the probability distribution over subsets $p(\boldsymbol{y}_k \mid \boldsymbol{y}_{<k}; \boldsymbol{\omega}, \boldsymbol{s})$.

Proposition 4.1 (Two-Stage Random Partition Model). Given a probability distribution over subset sizes $p(\boldsymbol{n}; \boldsymbol{\omega})$ with $\boldsymbol{n} \in \mathbb{N}_0^K$ and distribution parameters $\boldsymbol{\omega} \in \mathbb{R}_+^K$ and a PL probability distribution over random orderings $p(\pi; \boldsymbol{s})$ with $\pi \in \{0,1\}^{n \times n}$ and distribution parameters $\boldsymbol{s} \in \mathbb{R}_+^n$, the probability mass function $p(Y; \boldsymbol{\omega}, \boldsymbol{s})$ of the two-stage RPM is given by

$$
p(Y; \boldsymbol{\omega}, \boldsymbol{s}) = p(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K; \boldsymbol{\omega}, \boldsymbol{s}) = p(\boldsymbol{n}; \boldsymbol{\omega}) \sum_{\pi \in \Pi_Y} p(\pi; \boldsymbol{s}) \tag{34}
$$

where $\Pi_Y = \{\pi : \boldsymbol{y}_k = \sum_{i=\nu_k+1}^{\nu_k+n_k} \boldsymbol{\pi}_i, k = 1, \ldots, K\}$, and $\boldsymbol{y}_k$ and ν_k as in Definition B.2.

Proof. Using Lemmas B.5 and B.6, we write

$$
\begin{aligned}
p(Y) =& p(\boldsymbol{y}_1, \dots, \boldsymbol{y}_K; \boldsymbol{\omega}, \boldsymbol{s}) = p(\boldsymbol{y}_1; \boldsymbol{\omega}, \boldsymbol{s}) \cdots p(\boldsymbol{y}_K \mid \{\boldsymbol{y}_j\}_{j<K}; \boldsymbol{\omega}, \boldsymbol{s}) \\
=& \left(p(n_1; \boldsymbol{\omega}) \sum_{\bar{\pi}_1 \in \Pi_{\boldsymbol{y}_1}} p(\bar{\pi}_1 \mid n_1; \boldsymbol{s}) \right) \\
& \cdots \left(p(n_K \mid \{n_j\}_{j<K}; \boldsymbol{\omega}) \sum_{\bar{\pi}_K \in \Pi_{\boldsymbol{y}_K}} p(\bar{\pi}_K \mid \{n_j\}_{j \le K}; \boldsymbol{s}) \right) && (35) \\
=& p(n_1; \boldsymbol{\omega}) \cdots p(n_K \mid \{n_K\}_{j<K}; \boldsymbol{\omega}) \\
& \cdot \left(\sum_{\bar{\pi}_1 \in \Pi_{\boldsymbol{y}_1}} p(\bar{\pi}_1 \mid n_1; \boldsymbol{s}) \cdots \sum_{\bar{\pi}_K \in \Pi_{\boldsymbol{y}_K}} p(\bar{\pi}_K \mid \{n_j\}_{j \le K}; \boldsymbol{s}) \right) && (36) \\
=& p(\boldsymbol{n}; \boldsymbol{\omega}) \left(\sum_{\bar{\pi}_1 \in \Pi_{\boldsymbol{y}_1}} \cdots \sum_{\bar{\pi}_K \in \Pi_{\boldsymbol{y}_K}} p(\bar{\pi}_1 \mid n_1; \boldsymbol{s}) \cdots p(\bar{\pi}_K \mid \{n_j\}_{j \le K}; \boldsymbol{s}) \right) && (37) \\
=& p(\boldsymbol{n}; \boldsymbol{\omega}) \sum_{\pi \in \Pi_Y} p(\pi \mid \boldsymbol{n}; \boldsymbol{s}) && (38) \\
=& p(\boldsymbol{n}; \boldsymbol{\omega}) \sum_{\pi \in \Pi_Y} p(\pi; \boldsymbol{s}) && (39)
\end{aligned}
$$

□

B.3 Approximating the Probability Mass Function

Lemma 4.2. $p(Y; \boldsymbol{\omega}, \boldsymbol{s})$ can be upper and lower bounded as follows

$$
\forall \pi \in \Pi_Y : \ p(\boldsymbol{n}; \boldsymbol{\omega}) p(\pi; \boldsymbol{s}) \ \le \ p(Y; \boldsymbol{\omega}, \boldsymbol{s}) \ \le \ |\Pi_Y| p(\boldsymbol{n}; \boldsymbol{\omega}) \max_{\tilde{\pi}} p(\tilde{\pi}; \boldsymbol{s}) \tag{40}
$$

Proof. Since $p(\pi; \boldsymbol{s})$ is a probability we know that $\forall \pi \in \{0,1\}^{n \times n} \quad p(\pi; \boldsymbol{s}) \ge 0$. Thus, it follows directly that:

$$
\forall \pi \in \Pi_Y : \quad p(Y; \boldsymbol{\omega}, \boldsymbol{s}) = p(\boldsymbol{n}; \boldsymbol{\omega}) \sum_{\pi' \in \Pi_Y} p(\pi'; \boldsymbol{s}) \ge p(\boldsymbol{n}; \boldsymbol{\omega}) p(\pi; \boldsymbol{s}),
$$

proving the lower bound of Lemma 4.2.

On the other hand, can prove the upper bound in Lemma 4.2 by:

$$
\begin{aligned}
p(Y; \boldsymbol{\omega}, \boldsymbol{s}) = & p(\boldsymbol{n}; \boldsymbol{\omega}) \sum_{\pi' \in \Pi_Y} p(\pi'; \boldsymbol{s}) \\
\le & p(\boldsymbol{n}; \boldsymbol{\omega}) \sum_{\pi' \in \Pi_Y} \max_{\pi \in \Pi_Y} p(\pi; \boldsymbol{s}) \\
= & p(\boldsymbol{n}; \boldsymbol{\omega}) \max_{\pi \in \Pi_Y} p(\pi; \boldsymbol{s}) \sum_{\pi' \in \Pi_Y} 1 \\
= & |\Pi_Y| \cdot p(\boldsymbol{n}; \boldsymbol{\omega}) \max_{\pi \in \Pi_Y} p(\pi; \boldsymbol{s}) \\
\le & |\Pi_Y| \cdot p(\boldsymbol{n}; \boldsymbol{\omega}) \max_{\pi} p(\pi; \boldsymbol{s})
\end{aligned}
$$

We can compute the maximum probability $\max_\pi p(\pi; \boldsymbol{s})$ with the probability of the permutation matrix $f_\pi(\boldsymbol{s})$, which sorts the unperturbed scores in decreasing order. □

B.4 The Differentiable Random Partition Model

We propose the DRPM $p(Y; \boldsymbol{\omega}, \boldsymbol{s})$, a differentiable and reparameterizable two-stage RPM.

Lemma 4.3 (DRPM). A two-stage RPM is differentiable and reparameterizable if the distribution over subset sizes $p(\boldsymbol{n}; \boldsymbol{\omega})$ and the distribution over orderings $p(\pi; \boldsymbol{s})$ are differentiable and reparameterizable.

Proof. To prove that our two-stage RPM is differentiable we need to prove that we can compute gradients for the bounds in Lemma 4.2 and to provide a reparameterization scheme for the two-stage approach in Definition B.2.

Gradients for the bounds: Since we assume that $p(\boldsymbol{n}; \boldsymbol{\omega})$ and $p(\pi; \boldsymbol{s})$ are differentiable and reparameterizable, we only need to show that we can compute $|\Pi_Y|$ and $\max_{\tilde{\pi}} p(\tilde{\pi}; \boldsymbol{s})$ in a differentiable manner to prove that the bounds in Lemma 4.2 are differentiable. By definition (see Section 4.1),

$$|\Pi_Y| = \prod_{k=1}^{K} |\Pi_{\boldsymbol{y}_k}| = \prod_{k=1}^{K} n_k!.$$

Hence, $|\Pi_Y|$ can be computed given a reparametrized version n_k, which is provided by the reparametrization trick for the MVHG $p(\boldsymbol{n}; \boldsymbol{\omega})$. Further, from Equation (14) we immediately see that the most probable permutation is given by the order induced by sorting the original, unperturbed scores $\boldsymbol{s}$ from highest to lowest. This implies that $\max_{\tilde{\pi}} p(\tilde{\pi}; \boldsymbol{s}) = p(\pi_{\boldsymbol{s}}; \boldsymbol{s})$, which we can compute due to $p(\pi_{\boldsymbol{s}}; \boldsymbol{s})$ being differentiable according to our assumptions.

Reparametrization of the two-stage approach: Given reparametrized versions of $\boldsymbol{n}$ and π, we compute a partition as follows:

$$\boldsymbol{y}_k = \sum_{i=\nu_k+1}^{\nu_k+n_k} \boldsymbol{\pi}_i, \quad \text{where} \quad \nu_k = \sum_{\iota=1}^{k-1} n_\iota \tag{41}$$

The challenge here is that we need to be able to backpropagate through n_k, which appears as an index in the sum. Let $\boldsymbol{\alpha}_k = \{0,1\}^n$, such that

$$(\boldsymbol{\alpha}_k)_i = \begin{cases} 1 & \text{if } \nu_k < i \leq \nu_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

Given such $\boldsymbol{\alpha}_k$, we can rewrite Equation (41) with

$$\boldsymbol{y}_k = \sum_{i=1}^{n} (\boldsymbol{\alpha}_k)_i \boldsymbol{\pi}_i. \tag{42}$$

While this solves the problem of propagating through sum indices, it is not clear how to compute $\boldsymbol{\alpha}_k$ in a differentiable manner. Similar to other works on continuous relaxations [Jang et al., 2016, Maddison et al., 2017], we can compute a relaxation of $\boldsymbol{\alpha}_k$ by introducing a temperature τ. Let us introduce auxiliary function $f : \mathbb{N} \to [0,1]^n$, that maps an integer x to a vector with entries

$$f_i(x; \tau) = \sigma\left(\frac{x - i + \epsilon}{\tau}\right),$$

such that $f_i(x;\tau) \approx 0$ if $\frac{x-i}{\tau} < 0$ and $f_i(x;\tau) \approx 1$ if $\frac{x-i}{\tau} \geq 0$. Note that $\sigma(\cdot)$ is the standard sigmoid function and $\epsilon << 1$ is a small positive constant to break the tie at $\sigma(0)$. We then compute an approximation of $\boldsymbol{\alpha}_k$ with

$$\tilde{\boldsymbol{\alpha}}_k(\tau) = f(\nu_k; \tau) - f(\nu_{k-1}; \tau),$$

$\tilde{\boldsymbol{\alpha}}_k(\tau) \in [0,1]^n$. Then, for $\tau \to 0$ we have $\tilde{\boldsymbol{\alpha}}_k(\tau) \to \boldsymbol{\alpha}_k$. In practice, we cannot set $\tau = 0$ since this would amount to a division by 0. Instead, we can apply the straight-through estimator [Bengio et al., 2013] to the auxiliary function $f(x;\tau)$ in order to get $\tilde{\boldsymbol{\alpha}}_k \in \{0,1\}^n$ and use it to compute Equation (42). □

Note that in our experiments, we use the MVHG relaxation of Sutter et al. [2023] and can thus leverage that they return one-hot encodings for n_k. This allows a different path for computing $\boldsymbol{\alpha}_k$ which circumvents introducing yet another temperature parameter altogether. We refer to our code in the supplement for more details.

Table 3: Total GPU hours per experiment. We report the cumulative training and testing hours to generate the results shown in the main part of this manuscript. We relied on our internal cluster infrastructure equipped with RTX2080Ti GPUs. Hence, we report the number of compute hours for this GPU-type.

Experiment	Computation Time (h)
Clustering (Section 5.1)	100
Partitioning of Generative Factors (Section 5.2)	480
MTL (Section 5.3)	100

C Experiments

In the following, we describe each of our experiments in more detail and provide additional ablations. All our experiments were run on RTX2080Ti GPUs. Each run took 6h-8h (Variational Clustering), 4h-6h (Generative Factor Partitioning), or $\sim$ 1h (Multitask Learning) respectively. We report the training and test time per model. Please note that we can only report the numbers to generate the final results but not the development time.

C.1 Variational Clustering with Random Partition Models

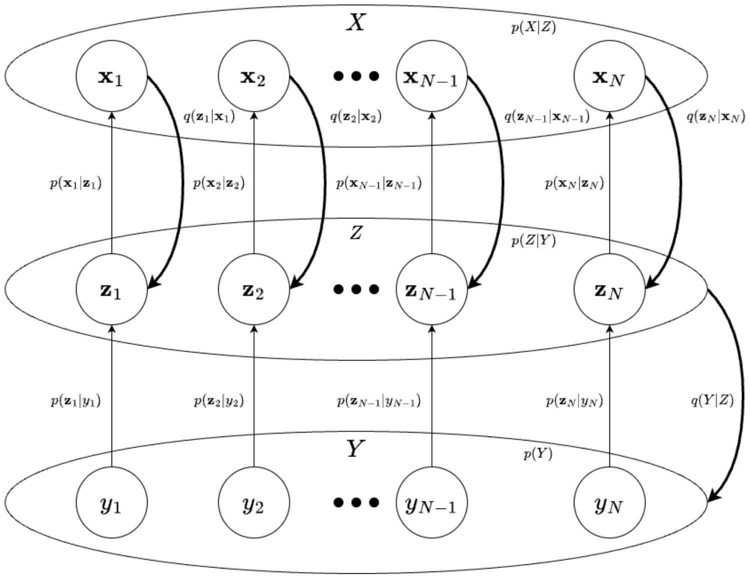

X
$p(X|Z)$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_{N-1}$
$\mathbf{x}_N$
$q(\mathbf{z}_1|\mathbf{x}_1)$
$q(\mathbf{z}_2|\mathbf{x}_2)$
$q(\mathbf{z}_{N-1}|\mathbf{x}_{N-1})$
$q(\mathbf{z}_N|\mathbf{x}_N)$
$p(\mathbf{x}_1|\mathbf{z}_1)$
$p(\mathbf{x}_2|\mathbf{z}_2)$
$p(\mathbf{x}_{N-1}|\mathbf{z}_{N-1})$
$p(\mathbf{x}_N|\mathbf{z}_N)$
Z
$p(Z|Y)$
$\mathbf{z}_1$
$\mathbf{z}_2$
$\mathbf{z}_{N-1}$
$\mathbf{z}_N$
$p(\mathbf{z}_1|y_1)$
$p(\mathbf{z}_2|y_2)$
$p(\mathbf{z}_{N-1}|y_{N-1})$
$p(\mathbf{z}_N|y_N)$
$q(Y|Z)$
Y
$p(Y)$
y_1
y_2
y_{N-1}
y_N

Figure 5: Generative model of the DRPM clustering model. Generative paths are marked with thin arrows, whereas inference is in bold.

C.1.1 Loss Function

As mentioned in Section 5.1, for a given dataset X with N samples, let Z and Y contain the respective latent vectors and cluster assignments for each sample in X. The generative process can then be summarized as follows: First, we sample the cluster assignments Y from an RPM, i.e., $Y \sim P(Y; \boldsymbol{\omega}, \boldsymbol{s})$. Given Y, we can sample the latent variables Z, where for each $\boldsymbol{y}$ we have $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{y}}, \boldsymbol{\sigma}_{\boldsymbol{y}}^{\boldsymbol{T}} \mathbb{I}_l)$, $\boldsymbol{z} \in \mathbb{R}^l$. Finally, we sample X by passing each $\boldsymbol{z}$ through a decoder like in vanilla VAEs. Using Bayes rule and Jensen's inequality, we can then derive the following evidence lower bound (ELBO):

$$\begin{aligned}\log(p(X)) &= \log\left(\int\sum_{Y} p(X,Y,Z)dZ\right)\\ &\geq \mathbb{E}_{q(Z,Y|X)}\left[\log\left(\frac{p(X|Z)p(Z|Y)p(Y)}{q(Z,Y|X)}\right)\right]\\ &:= \mathcal{L}_{ELBO}(X)\end{aligned}$$

We then assume that we can factorize the approximate posterior as follows:

$$q(Z,Y|X) = q(Y|X)\prod_{\boldsymbol{x}\in X} q(\boldsymbol{z}|\boldsymbol{x})$$

Note that while we do assume conditional independence between $\boldsymbol{z}$ given its corresponding $\boldsymbol{x}$, we model $q(Y|X)$ with the DRPM and do not have to assume conditional independence between different cluster assignments. This allows us to leverage dependencies between samples from the dataset. Hence, we can rewrite the ELBO as follows:

$$\begin{aligned}\mathcal{L}_{ELBO}(X) =&\mathbb{E}_{q(Z|X)}\left[\log(p(X|Z))\right]\\ &- \mathbb{E}_{q(Y|X)}\left[KL[q(Z|X)||p(Z|Y)]\right]\\ &- KL[q(Y|X)||p(Y)]\\ =&\sum_{\boldsymbol{x}\in X}\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right]\\ &- \sum_{\boldsymbol{x}\in X}\mathbb{E}_{q(Y|X)}\left[KL[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|Y)]\right]\\ &- KL[q(Y|X)||p(Y)]\end{aligned}$$

See Figure 5 for an illustration of the generative process and the assumed inference model. Since computing $P(Y)$ and $q(Y|X)$ is intractable, we further apply Lemma 4.2 to approximate the KL-Divergence term in $\mathcal{L}_{ELBO}$, leading to the following lower bound:

$$\begin{aligned}\mathcal{L}_{ELBO} \geq& \sum_{\boldsymbol{x}\in X}\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] &(43)\\ &- \sum_{\boldsymbol{x}\in X}\mathbb{E}_{q(Y|X)}\left[KL[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|Y)]\right] &(44)\\ &- \mathbb{E}_{q(Y|X)}\left[\log\frac{|\Pi_Y|\cdot q(\boldsymbol{n};\boldsymbol{\omega}(X))}{p(\boldsymbol{n};\boldsymbol{\omega})p(\pi_Y;\boldsymbol{s})}\right] &(45)\\ &- \log\left(\max_{\tilde{\pi}} q(\tilde{\pi};\boldsymbol{s}(X))\right), &(46)\end{aligned}$$

where π_Y is the permutation that lead to Y during the two-stage resampling process. Further, we want to control the regularization strength of the KL divergences similar to the β-VAE [Higgins et al., 2016]. Since the different terms have different regularizing effects, we rewrite Equations (45) and (46) and weight the individual terms as follows, leading to our final loss:

$$\begin{aligned}\mathcal{L} :=& -\sum_{\boldsymbol{x}\in X}\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] &(47)\\ &+ \beta\cdot\sum_{\boldsymbol{x}\in X}\mathbb{E}_{q(Y|X)}\left[KL[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|Y)]\right] &(48)\\ &+ \gamma\cdot\mathbb{E}_{q(Y|X)}\left[\log\left(\frac{|\Pi_Y|\cdot q(\boldsymbol{n};\boldsymbol{\omega}(X))}{p(\boldsymbol{n};\boldsymbol{\omega})}\right)\right] &(49)\\ &+ \delta\cdot\mathbb{E}_{q(Y|X)}\left[\log\left(\frac{\max_{\tilde{\pi}} q(\tilde{\pi};\boldsymbol{s}(X))}{p(\pi_Y;\boldsymbol{s})}\right)\right] &(50)\end{aligned}$$

C.1.2 Architecture

The model for our clustering experiments is a relatively simple, fully-connected autoencoder with a structure as seen in Figure 6. We have a fully connected encoder E with three layers mapping the input

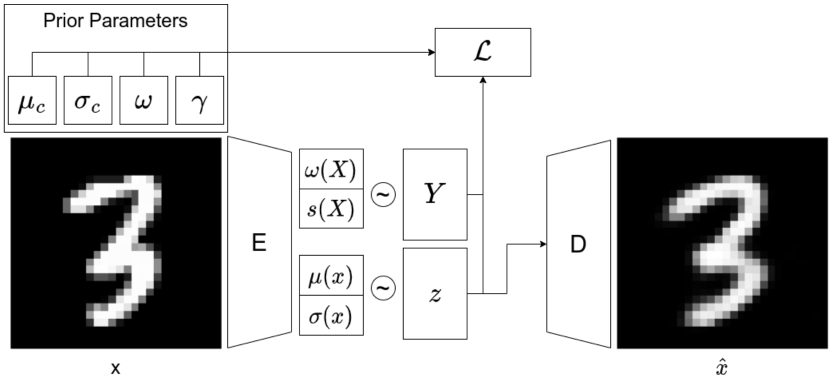


Figure 6: Autoencoder architecture of the DRPM-VC model.

to 500, 500, and 2000 neurons, respectively. We then compute each parameter by passing the encoder output through a linear layer and mapping to the respective parameter dimension in the last layer. In our experiments, we use a latent dimension size of $l = 10$ for MNIST and $l = 20$ for FMNIST, such that $\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x}) \in \mathbb{R}^l$. To understand the architecture choice for the DRPM parameters, let us first take a closer look at Equation (48). For each sample $\boldsymbol{x}$, this term minimizes the expected KL divergence between its approximate posterior $q(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \text{diag}(\boldsymbol{\sigma}(\boldsymbol{x})))$ and the prior at index $\boldsymbol{y}$ given by the partition Y sampled from the DRPM $q(Y|X; \boldsymbol{s}, \boldsymbol{\omega})$, i.e., $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{y}}, \text{diag}(\boldsymbol{\sigma}_{\boldsymbol{y}}))$. Ideally, the most likely partition should assign the approximate posterior to the prior that minimizes this KL divergence. We can compute such $\boldsymbol{s}(X)$ and $\boldsymbol{\omega}(X)$ given the parameters of the approximate posterior and priors as follows:

$$\forall \boldsymbol{x}_i \in X : s_i(\boldsymbol{x}_i) = u \cdot \left(K - \arg\min_k \left(KL[\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}_i), \text{diag}(\boldsymbol{\sigma}(\boldsymbol{x}_i)))||\mathcal{N}(\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\sigma}_k))]\right)\right)$$

$$\boldsymbol{\omega}(X) = \frac{1}{|X|} \sum_{\boldsymbol{x} \in X}^{N} \left\{ \frac{\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\sigma}_k))}{\sum_{k'=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \text{diag}(\boldsymbol{\sigma}_{k'}))} \right\}_{k=1}^{K},$$

where u is a scaling constant that controls the probability of sampling the most likely partition. Note that $\boldsymbol{\omega}$ and $\boldsymbol{s}$ minimize Equation (48) if defined this way when given the distribution parameters of the approximate posterior and the priors. The only thing that is left unclear is how much u should scale the scores $\boldsymbol{s}$. Ultimately, we leave u as a learnable parameter but detach the rest of the computation of $\boldsymbol{s}$ and $\boldsymbol{\omega}$ from the computational graph to improve stability during training. Finally, once we resample $z \sim \mathcal{N}(\mu(\boldsymbol{x}), \sigma(\boldsymbol{x}))$, we pass it through a fully connected decoder D with four layers mapping z to 2000, 500, and 500 neurons in the first three layers and then finally back to the input dimension in the last layer to end up with the reconstructed sample $\hat{\boldsymbol{x}}$.

C.1.3 Training

As in vanilla VAEs, we can estimate the reconstruction term in Equation (47) with MCMC by applying the reparametrization trick [Kingma and Welling, 2014] to $q(\boldsymbol{z}|\boldsymbol{x})$ to sample M samples $\boldsymbol{z}^{(i)} \sim q(\boldsymbol{z}|\boldsymbol{x})$ and compute their reconstruction error to estimate Equation (47). Similarly, we can sample from $q(Y|X)$ L times to estimate the terms in Equations (48) to (50), such that we minimize

$$\begin{aligned}\tilde{\mathcal{L}} \coloneqq &- \sum_{\boldsymbol{x}\in X} \frac{1}{M} \sum_{i=1}^{M} \log p(\boldsymbol{x}|\boldsymbol{z}^{(i)}) \\ &+ \frac{\beta}{L} \cdot \sum_{\boldsymbol{x}\in X} \sum_{i=1}^{L} KL[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|Y^{(i)})] \\ &+ \frac{\gamma}{L} \cdot \sum_{i=1}^{L} \log \left(\frac{|\Pi_{Y^{(i)}}| \cdot q(\boldsymbol{n}^{(i)}; \boldsymbol{\omega}(X))}{p(\boldsymbol{n}^{(i)}; \boldsymbol{\omega})} \right) \\ &+ \frac{\delta}{L} \cdot \sum_{i=1}^{L} \log \left(\frac{\max_{\tilde{\pi}} q(\tilde{\pi}; \boldsymbol{s}(X))}{p(\pi_{Y^{(i)}}; \boldsymbol{s})} \right)\end{aligned}$$

In our experiments, we set $M = 1$ and $L = 100$ since the MVHG and PL distributions are not concentrated around their mean very well, and more Monte Carlo samples thus lead to better approximations of the expectation terms. We further set $\beta = 1$ for MNIST and $\beta = 0.1$ for FMNIST, and otherwise $\gamma = 1$, and $\delta = 0.01$ for all experiments.

To resample $\boldsymbol{n}$ and π we need to apply temperature annealing [Grover et al., 2019, Sutter et al., 2023]. To do this, we applied the exponential schedule that was originally proposed together with the Gumbel-Softmax trick [Jang et al., 2016, Maddison et al., 2017], i.e., $\tau = \max(\tau_{final}, exp(-rt))$, where t is the current training step and r is the annealing rate. For our experiments, we choose $r = \frac{\log(\tau_{final}) - \log(\tau_{init})}{100000}$ in order to annealing over 100000 training step. Like Jang et al. [2016], we set $\tau_{init} = 1$ and $\tau_{final} = 0.5$.

Similar to Jiang et al. [2016], we quickly realized that proper initialization of the cluster parameters and network weights is crucial for variational clustering. In our experiments, we pretrained the autoencoder structure by adapting the contrastive loss of [Li et al., 2022], as they demonstrated that their representations manage to retain clusters in low-dimensional space. Further, we also added a reconstruction loss to initialize the decoder properly. To initialize the prior parameters, we fit a GMM to the pretrained embeddings of the training set and took the resulting Gaussian parameters to initialize our priors. Note that we used the same initialization across all baselines. See Appendix C.1.4 for an ablation where we pretrain with only a reconstruction loss similar to what was proposed with the VADE baseline.

To optimize the DRPM-VC in our experiments, we used the AdamW [Loshchilov and Hutter, 2019] optimizer with a learning rate of 0.0001 with a batch size of 256 for 1024 epochs. During initial experiments with the DRPM-VC, we realized that the pretrained weights of the encoder would often lose the learned structure in the first couple of training epochs. We suspect this to be an artifact of instabilities induced by temperature annealing. To deal with these problems, we decided to freeze the first three layers of the encoder when training the DRPM-VC, giving us much better results. See Appendix C.1.5 for an ablation where we applied the same optimization procedure to VADE.

Finally, when training the VADE baseline and the DRPM-VC on FMNIST, we often observe a local optimum where the prior distributions collapse and become identical. We can solve this problem by refitting the GMM in the latent space every 10 epochs and by using the resulting parameters to reinitialize the prior distributions.

C.1.4 Reconstruction Pretraining

While the results of our variational clustering method depend a lot on the specific pretraining, we want to demonstrate that improvements over the baselines do not depend on the chosen pretraining method. To that end, we repeat our experiments but initialize the weights of our model with an autoencoder that has been trained to minimize the mean squared error between the input and the reconstruction. This initialization procedure was originally proposed in [Jiang et al., 2016]. We present the results of this ablation in Table 4. Simply minimizing the reconstruction error does not necessarily retain cluster structures in the latent space. Thus, it does not come as a surprise that overall results get about 10% to 20% worse across most metrics, especially for MNIST, while results on FMNIST only slightly decrease. However, we still beat the baselines across most metrics, suggesting

Table 4: We compare the clustering performance of the DRPM-VC on test sets of MNIST and FMNIST between GMM in latent space (Latent GMM) and Variational Deep Embedding (VADE) initializing weights using an autoencoder trained on a reconstruction objective. We measure performance in terms of the Normalized Mutual Information (NMI), Adjusted Rand Index (ARI), and cluster accuracy (ACC) over five seeds and put the best model in bold.

	MNIST			FMNIST		
	NMI	ARI	ACC	NMI	ARI	ACC
Latent GMM	0.75±0.00	0.66±0.01	**0.75**±0.01	0.56±0.02	0.41±0.03	0.57±0.02
VADE	**0.77**±0.02	0.62±0.04	0.69±0.04	0.53±0.07	0.35±0.08	0.47±0.09
DRPM-VC	0.74±0.00	**0.67±0.01**	**0.75**±0.02	**0.59**±0.01	**0.47**±0.02	**0.62**±0.01

Table 5: We compare the clustering performance of the DRPM-VC on test sets of MNIST and FMNIST between GMM in latent space (Latent GMM), and Variational Deep Embedding (VADE) when freezing the encoder. We measure performance in terms of the Normalized Mutual Information (NMI), Adjusted Rand Index (ARI), and cluster accuracy (ACC) over five seeds and put the best model in bold.

	MNIST			FMNIST		
	NMI	ARI	ACC	NMI	ARI	ACC
Latent GMM	0.86±0.02	0.83±0.06	0.88±0.07	0.60±0.00	0.47±0.01	0.62±0.01
VADE	**0.90**±0.02	**0.88**±0.06	0.92±0.06	**0.64**±0.01	0.47±0.01	0.59±0.03
DRPM-VC	0.89±0.01	**0.88**±0.03	**0.94**±0.02	**0.64**±0.00	**0.51**±0.01	**0.65**±0.00

that modeling the implicit dependencies between cluster assignments helps to improve variational clustering performance.

C.1.5 Baselines with fixed Encoder

For the experiments in the main text, we wanted to implement the VADE baseline similar to the original method proposed in Jiang et al. [2016]. This means, in contrast to our method, we used their optimization procedure, i.e., Adam with a learning rate of 0.002 with a decay of 0.95 every 10 steps, and did not freeze the encoder as we do for the DRPM-VC. To ensure our results do not stem from this minor discrepancy, we perform an ablation experiment on VADE using the same optimizer and learning rate as with the DRPM-VC and freeze the encoder backbone. The results of this additional experiment can be found in Appendix C.1.5. As can be seen, VADE results do improve when adjusting the optimization procedure in this way. However, we still match or improve upon the results of VADE in most metrics, especially in ARI and ACC, suggesting purer clusters compared to VADE. We suspect this is because we assign samples to fixed clusters when sampling from the DRPM, whereas VADE performs soft assignments by marginalizing over a categorical distribution.

C.1.6 Additional Partition Samples

In Section 5.1, we have seen a sample of a partition of the DRPM-VC trained on FMNIST. We provide additional samples for both MNIST and FMNIST at the end of the appendix in Figures 12 and 13. We can see that for both datasets, the DRPM-VC learns coherent representations of each cluster that easily allow us to generate new samples from each class.

C.1.7 Samples per cluster

In addition to sampling partitions and then generating samples according to the sampled cluster assignments, we can also directly sample from each of the learned priors. We show some examples of this for both MNIST and FMNIST at the end of the appendix in Figures 14 and 15. We can again see that the DRPM-VC learns accurate cluster representations since each of the samples seems to

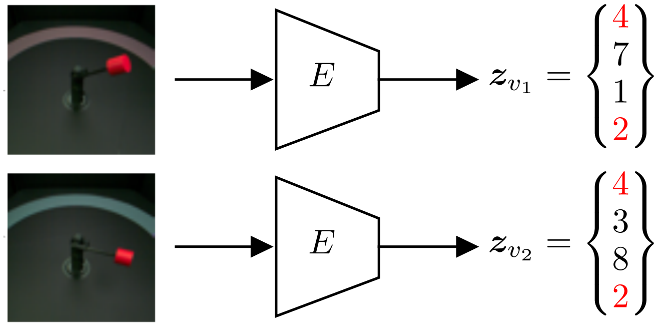


Figure 7: Motivation for the Partitioning of Generative Factors under weak supervision. The knowledge about the data collection process provides a weak supervision signal. We have access to a dataset of pairs of images of the same robot arm with a subset of shared generative factors (in red). We want to learn the shared and independent generative factors in addition to learning from the data. The images of the robot arms are taken from Locatello et al. [2020] but originate from the mpi3d toy dataset (see `https://github.com/rr-learning/disentanglement_dataset`). The image is from Sutter et al. [2023] and their ICLR 2023 presentation video (see `https://iclr.cc/virtual/2023/poster/10707`).

correspond to one of the classes in the datasets. Further, the clusters also seem to capture the diversity in each cluster, as we see a lot of variety across the generated samples.

C.2 Variational Partitioning of Generative Factors

We assume that we have access to multiple instances or views of the same event, where only a subset of generative factors changes between views. The knowledge about the data collection process provides a form of weak supervision. For example, we have two images of a robot arm as depicted here on the left side (see [Gondal et al., 2019]), which we would describe using high-level concepts such as color, position or rotation degree. From the data collection process, we know that a subset of these generative factors is shared between the two views We do not know how many generative factors there are in total nor how many of them are shared. More precisely, looking at the robot arm, we do not know that the views share two latent factors, depicted in red, out of a total of four factors. Please note that we chose four generative in Figure 7 only for illustrative reason as there are seven generative factors in the *mpi3d* toy dataset. Hence, the goal of learning under weak supervision is not only to infer good representations, but also inferring the number of shared and independent generative factors. Learning what is shared and what is independent lets us reason about the group structure without requiring explicit knowledge in the form of expensive labeling. Additionally, leveraging weak supervision and, hence, the underlying group structure holds promise for learning more generalizable and disentangled representations (see [e.g., Locatello et al., 2020]).

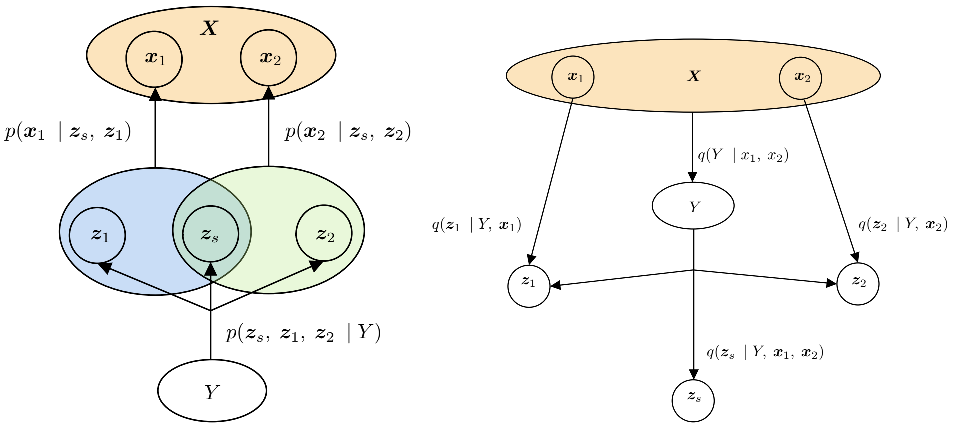
C.2.1 Generative Model

We assume the following generative model for DRPM-VAE

$$p(\boldsymbol{X}) = \int_{\boldsymbol{z}} p(\boldsymbol{X}, \boldsymbol{z}) d\boldsymbol{z} \tag{51}$$

$$= \int_{\boldsymbol{z}} p(\boldsymbol{X} \mid \boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} \tag{52}$$

where $\boldsymbol{z} = \{\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2\}$. The two frames share an unknown number n_s of generative latent factors $\boldsymbol{z}_s$, and an unknown number, n_1 and n_2, of independent factors $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$. The RPM infers n_k and


$\boldsymbol{X}$
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$p(\boldsymbol{x}_1 \mid \boldsymbol{z}_s, \boldsymbol{z}_1)$
$p(\boldsymbol{x}_2 \mid \boldsymbol{z}_s, \boldsymbol{z}_2)$
$\boldsymbol{z}_1$
$\boldsymbol{z}_s$
$\boldsymbol{z}_2$
$p(\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2 \mid Y)$
Y
$\boldsymbol{x}_1$
$\boldsymbol{X}$
$\boldsymbol{x}_2$
$q(Y \mid x_1, x_2)$
Y
$q(\boldsymbol{z}_1 \mid Y, \boldsymbol{x}_1)$
$q(\boldsymbol{z}_2 \mid Y, \boldsymbol{x}_2)$
$\boldsymbol{z}_1$
$\boldsymbol{z}_2$
$q(\boldsymbol{z}_s \mid Y, \boldsymbol{x}_1, \boldsymbol{x}_2)$
$\boldsymbol{z}_s$

(a) Generative Model (b) Inference Model

Figure 8: Graphical Models for DRPM-VAE models in the weakly-supervised experiment.

$\boldsymbol{z}_k$ using Y. Hence, the generative model extends to

$$\begin{aligned} p(\boldsymbol{X}) &= \int_{\boldsymbol{z}} p(\boldsymbol{X} \mid \boldsymbol{z}) \sum_{Y} p(\boldsymbol{z} \mid Y) p(Y) d\boldsymbol{z} \\ &= \int_{\boldsymbol{z}} p(\boldsymbol{x}_1, \boldsymbol{x}_2 \mid \boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2) \sum_{Y} p(\boldsymbol{z} \mid Y) p(Y) d\boldsymbol{z} \\ &= \int_{\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2} p(\boldsymbol{x}_1 \mid \boldsymbol{z}_s, \boldsymbol{z}_1) p(\boldsymbol{x}_2 \mid \boldsymbol{z}_s, \boldsymbol{z}_2) \sum_{Y} p(\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2 \mid Y) p(Y) d\boldsymbol{z}_s d\boldsymbol{z}_1 d\boldsymbol{z}_2 \end{aligned} \tag{53}$$

Figure 8 shows the generative and inference models assumptions in a graphical model.

C.2.2 DRPM ELBO

We derive the following ELBO using the posterior approximation $q(\boldsymbol{z}, Y \mid \boldsymbol{X})$

$$\mathcal{L}_{ELBO}(\boldsymbol{X}) = \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log p(\boldsymbol{X} \mid \boldsymbol{z}, Y) - \log \frac{q(\boldsymbol{z}, Y \mid \boldsymbol{X})}{p(\boldsymbol{z}, Y)} \right] \tag{54}$$

$$= \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log p(\boldsymbol{X} \mid \boldsymbol{z}) - \log \frac{q(\boldsymbol{z} \mid Y, \boldsymbol{X}) q(Y \mid \boldsymbol{X})}{p(\boldsymbol{z}) p(Y)} \right] \tag{55}$$

$$= \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log p(\boldsymbol{x}_1, \boldsymbol{x}_2 \mid \boldsymbol{z}) - \log \frac{q(\boldsymbol{z} \mid Y, \boldsymbol{X})}{p(\boldsymbol{z})} - \log \frac{q(Y \mid \boldsymbol{X})}{p(Y)} \right] \tag{56}$$

$$\begin{aligned} &= \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log p(\boldsymbol{x}_1 \mid \boldsymbol{z}_s, \boldsymbol{z}_1) \right] - \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log p(\boldsymbol{x}_2 \mid \boldsymbol{z}_s, \boldsymbol{z}_2) \right] \\ &\quad - \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log \frac{q(\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2 \mid Y, \boldsymbol{X})}{p(\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2)} \right] - \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log \frac{q(Y \mid \boldsymbol{X})}{p(Y)} \right] \end{aligned} \tag{57}$$

Following Lemma 4.2, we are able to optimize DRPM-VAE using the following ELBO $\mathcal{L}_{ELBO}(\boldsymbol{X})$:

$$\mathcal{L}_{ELBO} \geq \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log p(\boldsymbol{x}_1 \mid \boldsymbol{z}_s, \boldsymbol{z}_1) \right] - \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log p(\boldsymbol{x}_2 \mid \boldsymbol{z}_s, \boldsymbol{z}_2) \right] \tag{58}$$

$$- \mathbb{E}_{q(\boldsymbol{z},Y|\boldsymbol{X})} \left[\log \frac{q(\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2 \mid Y, \boldsymbol{X})}{p(\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2)} \right] \tag{59}$$

$$- \mathbb{E}_{q(Y|\boldsymbol{X})} \left[\log \left(\frac{|\Pi_Y| \cdot q(\boldsymbol{n} \mid \boldsymbol{X}; \boldsymbol{\omega})}{p(\boldsymbol{n}; \boldsymbol{\omega}_p) p(\pi_Y; \boldsymbol{s}_p)} \right) \right] \tag{60}$$

$$- \log \left(\max_{\tilde{\pi}} q(\tilde{\pi} \mid \boldsymbol{X}; \boldsymbol{s}) \right), \tag{61}$$

where π_Y is the permutation that lead to Y during the two-stage resampling process. Further, we want to control the regularization strength of the KL divergences similar to the β-VAE [Higgins et al.,

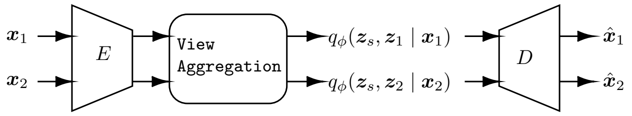


Figure 9: Setup for the weakly-supervised experiment. The three methods differ only in the `View Aggregation` module.

2016]. The ELBO $\mathcal{L}(\boldsymbol{X})$ to be optimized can be written as

$$\begin{aligned}
\mathcal{L}_{ELBO} = \; & \mathbb{E}_{q(\boldsymbol{z},Y \mid \boldsymbol{X})}\left[\log p(\boldsymbol{x}_1 \mid \boldsymbol{z}_s, \boldsymbol{z}_1)\right] + \mathbb{E}_{q(\boldsymbol{z},Y \mid \boldsymbol{X})}\left[\log p(\boldsymbol{x}_2 \mid \boldsymbol{z}_s, \boldsymbol{z}_2)\right] && (62)\\
& - \beta \cdot \mathbb{E}_{q(\boldsymbol{z},Y \mid \boldsymbol{X})}\left[\log \frac{q(\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2 \mid Y, \boldsymbol{X})}{p(\boldsymbol{z}_s, \boldsymbol{z}_1, \boldsymbol{z}_2)}\right] && (63)\\
& - \gamma \cdot \mathbb{E}_{q(Y \mid \boldsymbol{X})}\left[\log\left(\frac{|\Pi_Y| \cdot q(\boldsymbol{n}; \boldsymbol{\omega}(\boldsymbol{X}))}{p(\boldsymbol{n}; \boldsymbol{\omega}_p)}\right)\right] && (64)\\
& - \delta \cdot \mathbb{E}_{q(Y \mid \boldsymbol{X})}\left[\log\left(\frac{\max_{\tilde{\pi}} q(\tilde{\pi}; \boldsymbol{s}(\boldsymbol{X}))}{p(\pi_Y; \boldsymbol{s}_p)}\right)\right] && (65)
\end{aligned}$$

where $\boldsymbol{s}(\boldsymbol{X})$ and $\boldsymbol{\omega}(\boldsymbol{X})$ denote distribution parameters, which are inferred from $\boldsymbol{X}$ (similar to the Gaussian parameters in the vanilla VAE).

As in vanilla VAEs, we can estimate the reconstruction term in Equation (58) with MCMC by applying the reparametrization trick [Kingma and Welling, 2014] to $q(\boldsymbol{z} \mid Y, \boldsymbol{X})$ to sample L samples $\boldsymbol{z}^{(l)} \sim q(\boldsymbol{z} \mid Y, \boldsymbol{X})$ and compute their reconstruction error to estimate Equation (58). Similarly, we can sample from $q(Y \mid \boldsymbol{X})$ L times. We use $L = 1$ to estimate all expectations in $\mathcal{L}_{ELBO}$.

C.2.3 Implementation and Hyperparameters

In this experiment, we use the `disentanglement_lib` from Locatello et al. [2020]. We use the same architectures proposed in the original paper for all methods we compare to. The baseline algorithms, LabelVAE [Bouchacourt et al., 2018, Hosoya, 2018] and AdaVAE [Locatello et al., 2020] are already implemented in `disentanglement_lib`. For details on the implementation of these methods we refer to the original paper from Locatello et al. [2020]. HGVAE is implemented in Sutter et al. [2023]. We did not change any hyperparameters or network details. All experiments were performed using $\beta = 1$ as this is the best performing β (according to Locatello et al. [2020]. For DRPMVAE we chose $\gamma = 0.25$ for all runs. All models are trained on 5 different random seeds and the reported results are averaged over the 5 seeds. We report mean performance with standard deviations.

We adapted Figure 9 from Sutter et al. [2023]. It shows the baseline architecture, which is used for all methods. As already stated in the main part of the paper, the methods only differ in the `View Aggregation` module, which determines the shared and independent latent factors. Given a subset S of shared latent factors, we have

$$\begin{aligned}
q_\phi(z_i \mid \boldsymbol{x}_j) &= avg(q_\phi(z_i \mid \boldsymbol{x}_1), q_\phi(z_i \mid \boldsymbol{x}_2)) && \forall \;\; i \in S && (66)\\
q_\phi(z_i \mid \boldsymbol{x}_j) &= q_\phi(z_i \mid \boldsymbol{x}_j) && \text{else} && (67)
\end{aligned}$$

where avg is the averaging function of choice [Locatello et al., 2020, Sutter et al., 2023] and $j \in \{1, 2\}$. The methods used (i. e. Label-VAE, Ada-VAE, HG-VAE, DRPM-VAE) differ in how to select the subset S.

For DRPM-VAE, we infer $\boldsymbol{\omega}$ from the pairwise KL-divergences KL_{pw} between the latent vectors of the two views.

$$KL_{pw}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{1}{2} KL[q(\boldsymbol{z}_1 \mid \boldsymbol{x}_1) || q(\boldsymbol{z}_2 \mid \boldsymbol{x}_2)] + \frac{1}{2} KL[q(\boldsymbol{z}_2 \mid \boldsymbol{x}_2) || q(\boldsymbol{z}_1 \mid \boldsymbol{x}_1)] \qquad (68)$$

where $q(\boldsymbol{z}_j \mid \boldsymbol{x}_j)$ are the encoder outputs of the respective images. We do not average or sum across dimensions in the computation of $KL_{pw}(\cdot)$ such that the $KL_{pw}(\cdot)$ is d-dimensional, where d is the latent space size. The encoder E in Figure 9 maps to $\boldsymbol{\mu}(\boldsymbol{x}_j)$ and $\boldsymbol{\sigma}(\boldsymbol{x}_j)$ of a Gaussian distribution. Hence, we can compute the KL divergences above in closed form. Afterwards, we feed the pairwise KL divergence KL_{pw} to a single fully-connected layer, which maps from d to K values

$$\log \boldsymbol{\omega} = FC(KL_{pw}(\boldsymbol{x}_1, \boldsymbol{x}_2)) \qquad (69)$$

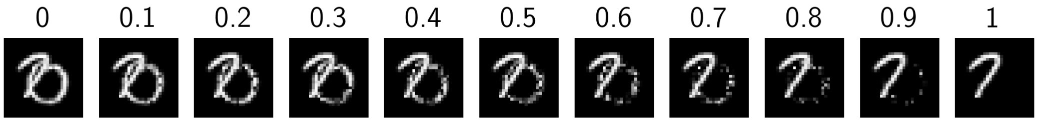


Figure 10: Samples from the noisyMultiMNIST dataset with increasing noise ratio in the right task.

where $d = 10$ and $K = 2$ in this experiment. d is the total number of latent dimensions and K is the number of groups in the latent space. To infer the scores $\boldsymbol{s}(\boldsymbol{X})$ we again rely on the pairwise KL divergence KL_{pw}. Instead of using another fully-connected layer, we directly use the log-values of the pairwise KL divergence

$$\log \boldsymbol{s} = \log KL_{pw}(\boldsymbol{x}_1, \boldsymbol{x}_2) \tag{70}$$

Similar to the original works, we also anneal the temperature parameter for $p(\boldsymbol{n}; \boldsymbol{\omega})$ and $p(\pi; \boldsymbol{s})$ [Grover et al., 2019, Sutter et al., 2023]. We use the same annealing function as in the clustering experiment (see Appendix C.1). We anneal the temperature τ from 1.0 to 0.5 over the complete training time.

C.3 Multitask Learning

C.3.1 MultiMNIST Dataset

The different tasks in multitask learning often vary in difficulty. To measure the effect of discrepancies in task difficulties on DRPM-MTL, we introduce the noisyMultiMNIST dataset.

The noisyMultiMNIST dataset modifies the MultiMNIST dataset [Sabour et al., 2017] as follows. In the right image, we set each pixel value to zero with probability $\alpha \in [0, 1]$. This is done before merging the left and right image in order to only affect the difficulty of the right task. Note that for $\alpha = 0$ noisyMultiMNIST is equivalent to MultiMNIST and for $\alpha = 1$ the right task can no longer be solved. This allows us to control the difficulty of the right task, without changing the difficulty of the left. A few examples are shown in Figure 10.

C.3.2 Implementation & Architecture

The multitask loss function for the *MultiMNIST* dataset is

$$\mathbb{L} = w_L \mathbb{L}_L + w_R \mathbb{L}_R \tag{71}$$

where w_L and w_L are the loss weights, and $\mathbb{L}_L$ and $\mathbb{L}_R$ are the individual loss terms for the respective tasks L and R. In our experiments, we set the task weights to be equal for all dataset versions, i.e. $w_L = w_R = 0.5$. We use these loss weights for the DRPM-MTL and ULS method. For the ULS method, it is by definition and to see the influence of a mismatch in loss weights. The DRPM-MTL method on the other hand does not need additional weighting of loss terms. The task losses are defined as cross-entropy losses

$$\mathbb{L}_t = -\sum_{c=1}^{C_t} \boldsymbol{gt}_c \log \boldsymbol{p}_c = -\boldsymbol{gt}^T \log \boldsymbol{p} \tag{72}$$

where $C_L = C_R = 10$ for MultiMNIST, $\boldsymbol{gt}$ is a one-hot encoded label vector and $\boldsymbol{p}$ is a categorical vector of estimated class assignments probabilities, i.e. $\sum_c \boldsymbol{p}_c = 1$.

The predictions for the individual tasks $\boldsymbol{p}_t$ are given as

$$\boldsymbol{p}_t = h_{\theta_t}(\boldsymbol{z}), \quad \text{where} \tag{73}$$

$$\boldsymbol{z} = \text{enc}_\theta(\boldsymbol{x}) \tag{74}$$

for a sample $\boldsymbol{x} \in \boldsymbol{X}$ (see also Figure 11). We use an adaptation of the LeNet-5 architecture LeCun et al. [1998] to the multitask learning problem [Sener and Koltun, 2018]. Both DRPM-MTL and ULS use the same network $\text{enc}_\theta(\cdot)$ with shared architecture up to some layer for both tasks, after which the network branches into two task-specific sub-networks that perform the classifications. Different

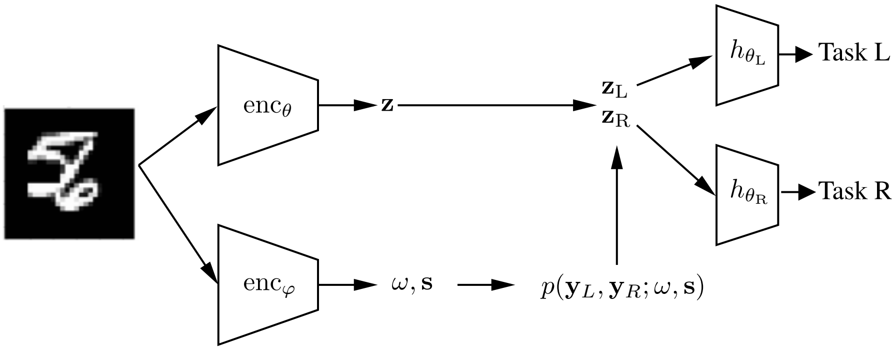

enc_θ
$\mathbf{z}$
$\mathbf{z}_{\mathrm{L}}$
$\mathbf{z}_{\mathrm{R}}$
$h_{\theta_{\mathrm{L}}}$
Task L
$h_{\theta_{\mathrm{R}}}$
Task R
enc_φ
$\omega, \mathbf{s}$
$p(\mathbf{y}_L, \mathbf{y}_R; \omega, \mathbf{s})$

Figure 11: Overview of the multitask learning pipeline of the DRPM-MTL method.

to the ULS method, the task-specific networks in the DRPM-MTL pipeline predict the digit using only a subset of $\boldsymbol{z}$. DRPM-MTL uses the following prediction scheme

$$\boldsymbol{p}_t = h_{\theta_t}(\boldsymbol{z}_t), \quad \text{where} \tag{75}$$

$$\boldsymbol{z}_t = \boldsymbol{z} \odot \boldsymbol{y}_t \tag{76}$$

$$\boldsymbol{y}_t = \mathrm{DRPM}(\boldsymbol{\omega}, \boldsymbol{s})_t = \mathrm{DRPM}(\mathrm{enc}_\varphi(\boldsymbol{x}))_t \tag{77}$$

The DRPM-MTL encoder first predicts a latent representation $\mathbf{z} \leftarrow \mathrm{enc}_\theta(\boldsymbol{x})$, where $\boldsymbol{x}$ is the input image. Using the same encoder architecture but different parameters φ , we predict a partitioning encoding $\boldsymbol{z}' \leftarrow \mathrm{enc}_\varphi(\boldsymbol{x})$. With a single linear layer per DRPM log-parameter $\log \boldsymbol{\omega}$ and $\log \boldsymbol{s}$ are computed. Next we infer the partition masks $\boldsymbol{y}_{\mathrm{L}}, \boldsymbol{y}_{\mathrm{R}} \sim p(\boldsymbol{y}_{\mathrm{L}}, \boldsymbol{y}_{\mathrm{R}}; \boldsymbol{\omega}, \boldsymbol{s})$. We then feed the masked latent representations $\boldsymbol{z}_{\mathrm{L}} \leftarrow \boldsymbol{z} \odot \boldsymbol{y}_{\mathrm{L}}$ and $\boldsymbol{z}_{\mathrm{R}} \leftarrow \boldsymbol{z} \odot \boldsymbol{y}_{\mathrm{R}}$ into the task specific classification networks $h_{\theta_{\mathrm{L}}}(\boldsymbol{z}_{\mathrm{L}})$ and $h_{\theta_{\mathrm{R}}}(\boldsymbol{z}_{\mathrm{R}})$ respectively to obtain the task specific predictions. Since the two tasks in the MultiMNIST dataset are of similar nature, the task-specifc networks $h_{\theta_{\mathrm{L}}}$ and $h_{\theta_{\mathrm{R}}}$ share the same architecture, but have different parameters.

C.3.3 Training

For both the ULS and the DRPM-MTL model, we use the Adam optimizer with learning rate 0.0005 and train them for 200 epochs with a batch size of 256. We again choose an exponential schedule for the temperature τ and anneal it over the training time, as is explained in Appendix C.1.3.

In our ablation we use $\alpha \in \{0, 0.1, 0.2, \ldots, 0.9\}$ and train each model with five different seeds. The reported accuracies and partition sizes are then means over the five seeds with the error bands indicating the variance and standard deviation respectively. We evaluate each model after the epoch with the best average test accuracy.

C.3.4 CelebA for MTL

In addition to the experiment shown in Section 5.3, we show additional results for DRPM-MTL on the CelebA dataset [Liu et al., 2015]. In MTL, each of the 40 attributes of the CelebA dataset serves as an individual task. Hence, using CelebA for MTL results is a 40 task learning problem making the scaling of different task losses more difficult compared to MultiMNIST (see Section 5.3) where we only need to scale two different tasks.

We again use the newly introduced DRPM-MTL method and compare it to the ULS model. We use the same pipeline as for MultiMNIST dataset but with different encoders and hyperparameters (see Appendices C.3.2 and C.3.3). We use the pipeline of Sener and Koltun [2018] with a ResNet-based encoder to map an image to a representation of $d = 64$ dimensions. For architectural details, we refer to Sener and Koltun [2018] and `https://github.com/isl-org/MultiObjectiveOptimization`.

Again, ULS inputs all $d = 64$ dimensions to the task-specific sub-networks whereas DRPM-MTL partitions the intermediate representations into n_T different subsets, which are then fed to the respective task networks. n_T is the number of tasks.

Compared to the MultiMNIST experiment (see Appendix C.3.2), we introduce an additional regularization for the DRPM-MTL method. The additional regularization is based on the upper bound in

Table 6: Results for the MTL experiment on the CelebA dataset. We compare the DRPM-MTL again to the ULS method. We assess the performance of both methods on two sub-experiment of the CelebA experiment. In Table 6a, we form a MTL experiment with 10 different tasks. In Table 6b, we form a MTL experiment with 20 different tasks where the first 10 tasks are the same as in the 10 tasks experiment. We train both methods for 50 methods using a learning rate of 0.0001 and a batch size of 128. The temperature annealing schedule remains the same as in the MultiMNIST experiment. We report the per task classification accuracy in percentages (%) as well as the average task accuracy in the bottow row of both subtables.

(a) 10 Tasks

	ULS	DRPM
T0	92.0±0.5	**92.4±0.5**
T1	**83.8±0.4**	83.7±0.2
T2	80.2±0.5	80.2±0.4
T3	81.9±0.8	**82.2±0.6**
T4	98.5±0.2	98.5±0.1
T5	95.2±0.2	**95.3±0.2**
T6	80.0±1.4	**82.4±0.4**
T7	82.0±0.3	**82.2±0.2**
T8	89.7±0.7	**90.7±0.2**
T9	94.6±0.5	**95.0±0.2**
avg(Tasks)	87.8±0.3	**88.3±0.1**

(b) 20 Tasks

	ULS	DRPM
T0	92.4±0.7	**93.0±0.2**
T1	83.7±0.6	**83.9±0.7**
T2	79.9±0.6	**80.1±0.4**
T3	82.4±0.5	**83.0±0.7**
T4	98.6±0.1	98.6±0.1
T5	95.2±0.1	**95.5±0.0**
T6	82.0±1.3	**84.4±0.4**
T7	82.5±0.1	**82.8±0.2**
T8	**90.1±0.9**	91.0±0.4
T9	94.7±0.2	**95.1±0.1**
T10	95.9±0.1	95.9±0.1
T11	**84.9±0.1**	84.6±0.3
T12	91.0±0.4	**91.6±0.2**
T13	94.7±0.1	**94.9±0.1**
T14	95.4±0.3	**96.0±0.1**
T15	99.2±0.0	99.2±0.1
T16	95.8±0.3	**96.0±0.1**
T17	97.3±0.3	**97.5±0.2**
T18	91.2±0.3	91.2±0.1
T19	87.0±0.3	**87.3±0.2**
avg(Tasks)	90.7±0.2	**91.1±0.1**

Lemma 4.2 and is penalizing size of $|\Pi_Y|$ for a given $\boldsymbol{n}$. Hence, the loss function changes to

$$\mathbb{L} = \frac{1}{n_T}\sum_{t=1}^{n_T}\mathbb{L}_t + \lambda \cdot \mathbb{L}_{\text{reg}} \tag{78}$$

$$\text{where} \quad \mathbb{L}_{\text{reg}} = \log\left(\prod_{t=1}^{n_T} n_t!\right) = \sum_{t=1}^{n_T}\log\Gamma(n_t+1) \tag{79}$$

For both versions of the experiment (i.e. $n_T = 10$ and $n_T = 20$), we set $\lambda = 0.015 \approx \frac{1}{64}$, which is the number of elements we want to partition. The task losses $\mathbb{L}_t$ are simple BCE losses similar to the MultiMNIST experiments but with two classes per task only.

We perform two different experiments based on the CelebA experiment. First, we use form a MTL experiments using the first 10 attributes out of the 40 attributes. Second, we increase the number of different tasks to 20. Because we sort the attributes alphabetically in both cases, the first 10 tasks are shared between the two experiment versions.

Table 6 shows the results of both methods, ULS and DRPM-MTL. We see that the DRPM-MTL scales better to a larger number of tasks compared to the ULS method, highlighting the importance of finding new ways of automatic scaling between tasks. Interestingly, the DRPM-MTL outperforms the ULS method on most tasks for the 20-tasks experiment even though it has only access to $d/n_T = 64/20 = 3.2$ dimensions on average. On the other hand, the ULS method can access the full set of 64 dimensions for every single task.

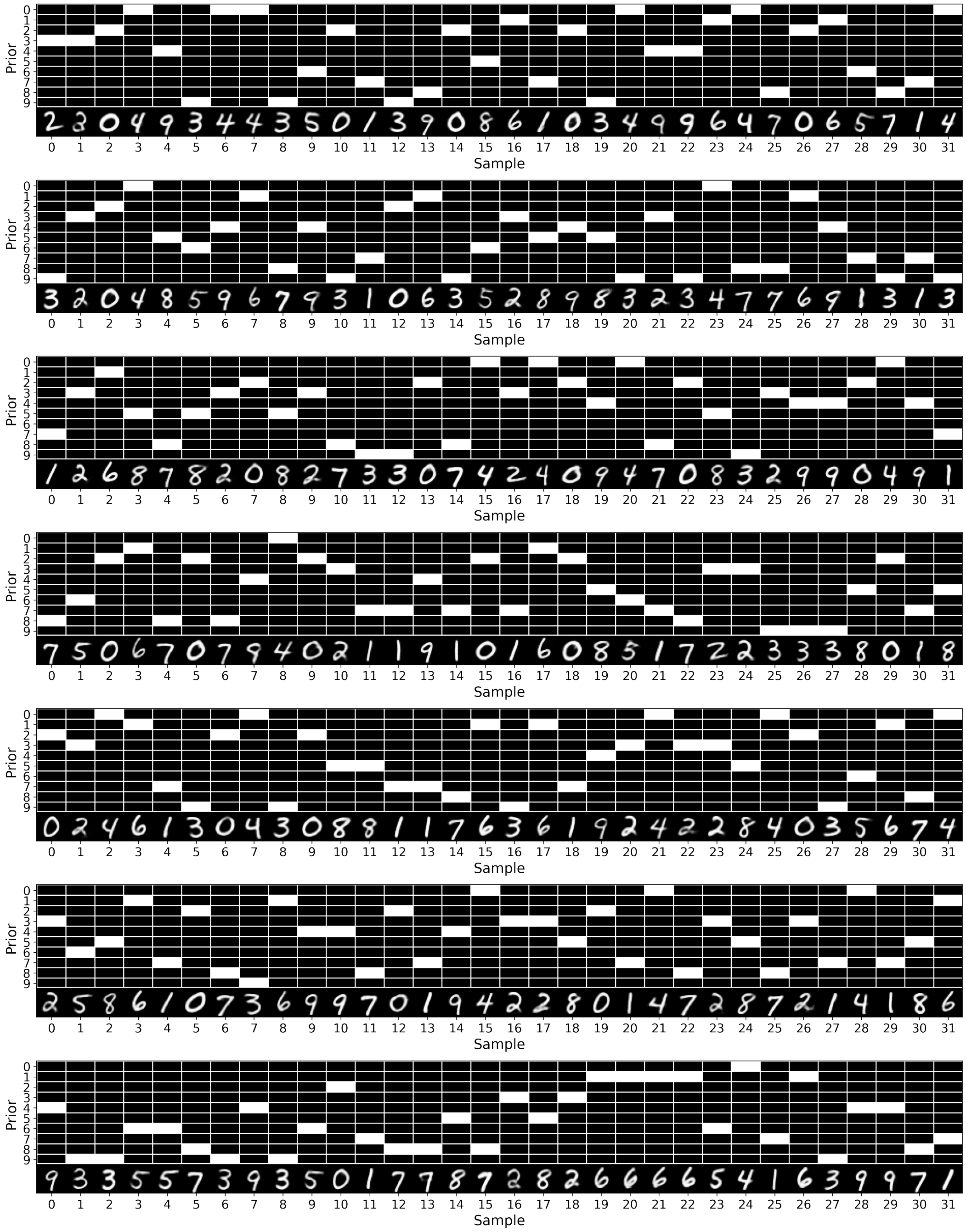


Figure 12: Additional partition samples from the DRPM-VC trained on MNIST. The different sets of each partition match each of the digits very well, even after repeatedly sampling from the model.

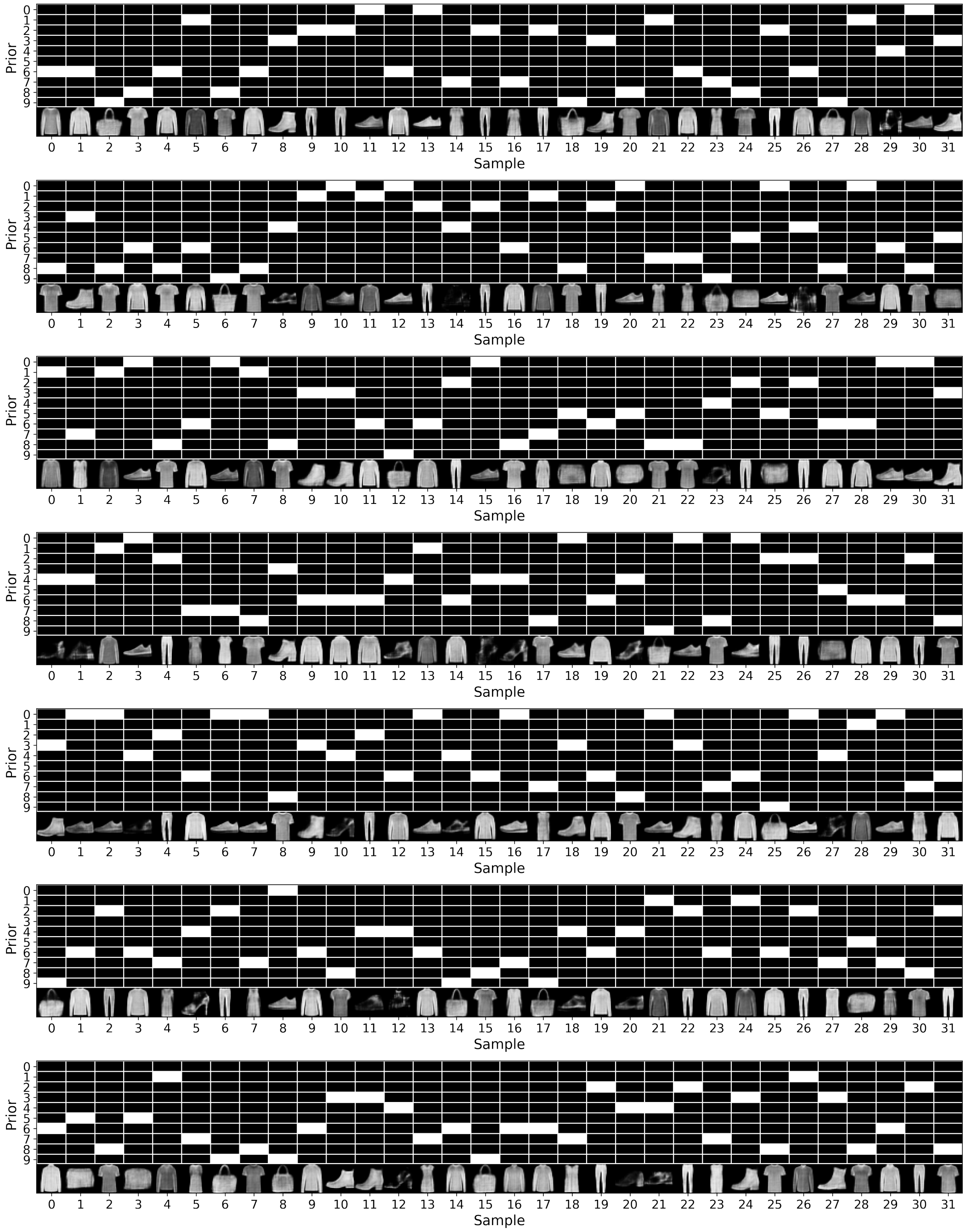


Figure 13: Additional partition samples from the DRPM-VC trained on FMNIST. Most clusters accurately represent one of the clothing categories and generate new samples very well. The only problem is with the handbag class, where the DRPM-VC learns two different clusters for different kinds of handbags (cluster 5 and 6).

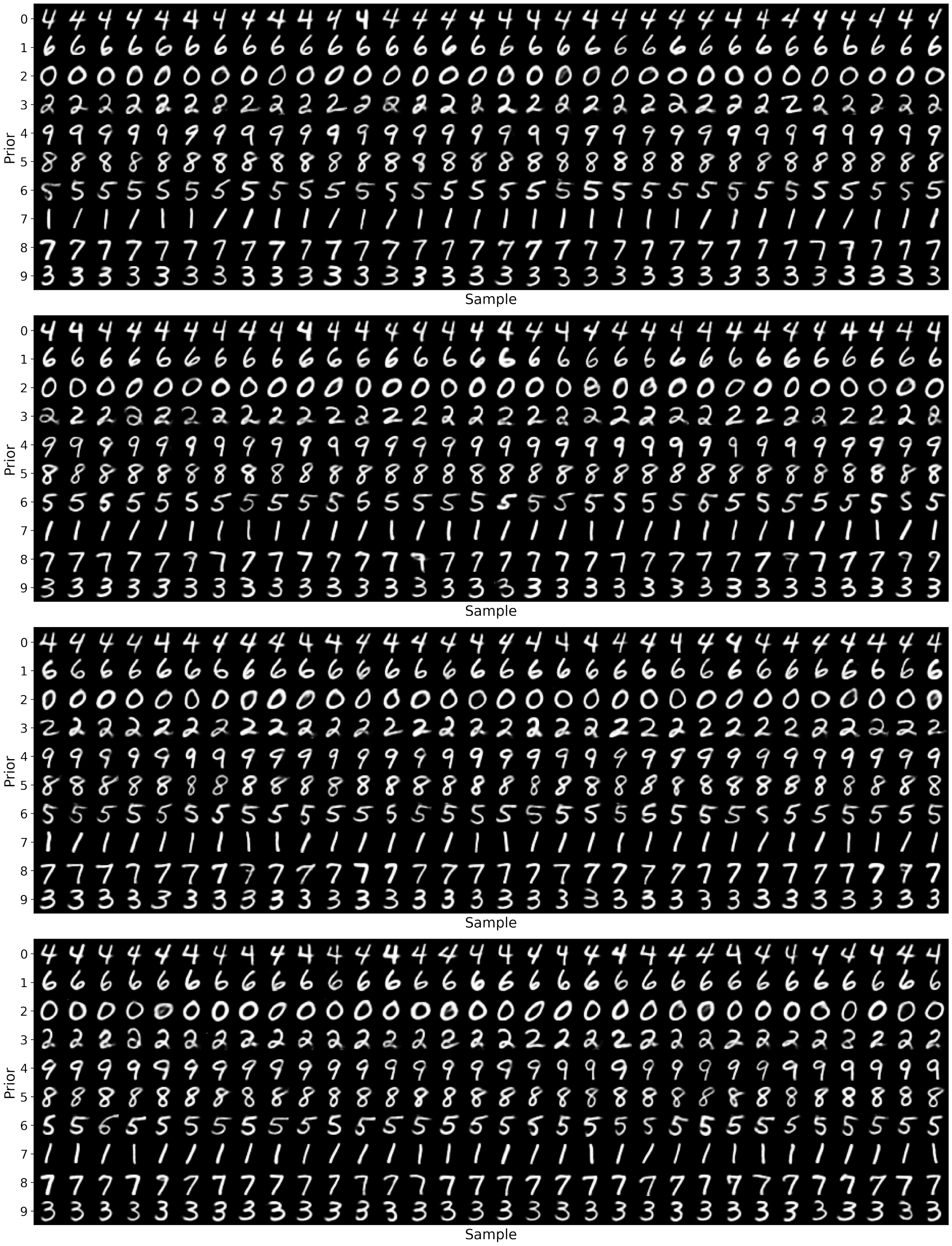


Figure 14: Various samples from each of the generative priors. Each prior learns to represent one of the digits. Further, we see a lot of variation between the different samples, suggesting that the clusters of the DRPM-VC manage to capture some of the diversity present in the dataset.

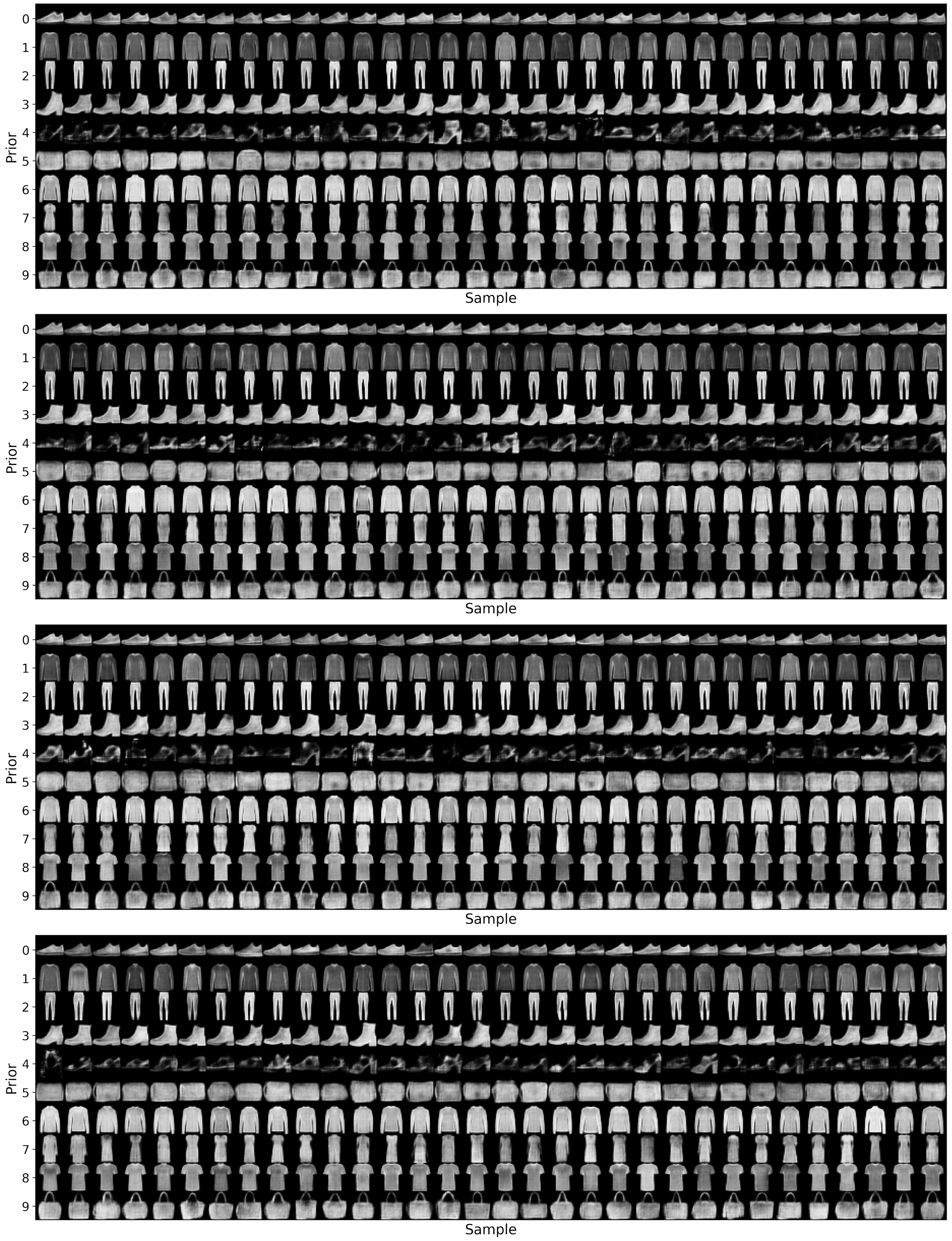


Figure 15: Various samples from each of the generative priors. Each prior learns to represent one of the digits. The DRPM-VC learns nice representations that provide coherent generations of most classes. For high-heels (cluster 4), generating new samples seems difficult due to the heterogeneity within that class.

References

R. P. Adams and R. S. Zemel. Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925*, 2011.

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

C. M. Bishop and M. Svensen. Robust Bayesian Mixture Modelling. *To appear in the Proceedings of ESANN*, page 1, 2004. Publisher: Citeseer.

D. Bouchacourt, R. Tomioka, and S. Nowozin. Multi-level variational autoencoder: Learning disentangled representations from grouped observations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

R. Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Machine learning: Proceedings of the tenth international conference*, pages 41–48, 1993.

R. Caruana and V. R. de Sa. Promoting poor features to supervisors: Some inputs work better as outputs. *Advances in Neural Information Processing Systems*, 9, 1996.

J. Chesson. A non-central multivariate hypergeometric distribution arising from biased sampling with application to selective predation. *Journal of Applied Probability*, 13(4):795–797, 1976. Publisher: Cambridge University Press.

R. De la Cruz-Mesía, F. A. Quintana, and P. Müller. Semiparametric Bayesian Classification with longitudinal Markers. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 56 (2):119–137, Mar. 2007. ISSN 0035-9254. doi: 10.1111/j.1467-9876.2007.00569.x. URL `https://doi.org/10.1111/j.1467-9876.2007.00569.x`.

N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan. Deep unsupervised clustering with Gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

R. A. Fisher. The logic of inductive inference. *Journal of the royal statistical society*, 98(1):39–82, 1935. Publisher: JSTOR.

A. Fog. Calculation methods for Wallenius' noncentral hypergeometric distribution. *Communications in Statistics—Simulation and Computation®*, 37(2):258–273, 2008. Publisher: Taylor & Francis.

M. W. Gondal, M. Wuthrich, D. Miladinovic, F. Locatello, M. Breidt, V. Volchkov, J. Akpo, O. Bachem, B. Schölkopf, and S. Bauer. On the Transfer of Inductive Bias from Simulation to the Real World: a New Disentanglement Dataset. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

R. L. Graham, D. E. Knuth, O. Patashnik, S. L.-C. i. Physics, and undefined 1989. Concrete mathematics: a foundation for computer science. *aip.scitation.org*, 3(5):165, 1989. doi: 10.1063/1.4822863. URL `https://aip.scitation.org/doi/pdf/10.1063/1.4822863`. Publisher: AIP Publishing.

A. Grover, E. Wang, A. Zweig, and S. Ermon. Stochastic Optimization of Sorting Networks via Continuous Relaxations. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1eSS3CcKX`.

J. A. Hartigan. Partition models. *Communications in Statistics - Theory and Methods*, 19(8): 2745–2756, 1990. doi: 10.1080/03610929008830345. Publisher: Taylor & Francis.

I. Higgins, L. Matthey, A. Pal, C. Burgess, and X. Glorot. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016. URL `https://openreview.net/forum?id=Sy2fzU9gl`.

H. Hosoya. A simple probabilistic deep generative model for learning generalizable disentangled representations from grouped data. *CoRR*, abs/1809.0, 2018.

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

Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

M. E. Johnson. *Multivariate statistical simulation: A guide to selecting and generating continuous multivariate distributions*, volume 192. John Wiley \textbackslash& Sons, 1987.

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2014.

V. Kurin, A. D. Palma, I. Kostrikov, S. Whiteson, and M. P. Kumar. In Defense of the Unitary Scalarization for Deep Multi-Task Learning. *CoRR*, abs/2201.04122, 2022. URL `https://arxiv.org/abs/2201.04122`.

Y. LeCun, C. Cortes, and C. J. Burges. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998. Issue: 11.

C. J. Lee and H. Sang. Why the Rich Get Richer? On the Balancedness of Random Partition Models. *arXiv preprint arXiv:2201.12697*, 2022.

Y. Li, M. Yang, D. Peng, T. Li, J. Huang, and X. Peng. Twin Contrastive Learning for Online Clustering. *International Journal of Computer Vision*, 130(9):2205–2221, Sept. 2022. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-022-01639-z. URL `http://arxiv.org/abs/2210.11680`. arXiv:2210.11680 [cs].

S. W. Linderman, G. E. Mena, H. J. Cooper, L. Paninski, and J. P. Cunningham. Reparameterizing the Birkhoff Polytope for Variational Permutation Inference. In A. J. Storkey and F. Pérez-Cruz, editors, *International Conference on Artificial Intelligence and Statistics, {AISTATS} 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, pages 1618–1627. PMLR, 2018. URL `http://proceedings.mlr.press/v84/linderman18a.html`.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep Learning Face Attributes in the Wild, Sept. 2015. URL `http://arxiv.org/abs/1411.7766`. arXiv:1411.7766 [cs].

F. Locatello, B. Poole, G. Rätsch, B. Schölkopf, O. Bachem, and M. Tschannen. Weakly-supervised disentanglement without compromises. In *International Conference on Machine Learning*, pages 6348–6359. PMLR, 2020.

I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization, Jan. 2019. URL `http://arxiv.org/abs/1711.05101`. arXiv:1711.05101 [cs, math].

R. D. Luce. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 1959.

J. MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

C. Maddison, A. Mnih, and Y. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

L. Manduchi, K. Chin-Cheong, H. Michel, S. Wellmann, and J. E. Vogt. Deep Conditional Gaussian Mixture Model for Constrained Clustering. *ArXiv*, abs/2106.0, 2021.

T. Mansour and M. Schork. *Commutation relations, normal ordering, and Stirling numbers*. CRC Press Boca Raton, 2016.

M. Marcus. Some Properties and Applications of Doubly Stochastic Matrices. *The American Mathematical Monthly*, 67(3):215–221, 1960. ISSN 00029890, 19300972. URL `http://www.jstor.org/stable/2309679`. Publisher: Mathematical Association of America.

G. E. Mena, D. Belanger, S. W. Linderman, and J. Snoek. Learning Latent Permutations with Gumbel-Sinkhorn Networks. In *6th International Conference on Learning Representations, {ICLR} 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=Byt3oJ-0W`.

G. Monge. Mémoire sur la théorie des déblais et des remblais. *Mem. Math. Phys. Acad. Royale Sci.*, pages 666–704, 1781.

F. Petersen, C. Borgelt, H. Kuehne, and O. Deussen. Monotonic Differentiable Sorting Networks. In *International Conference on Learning Representations*, 2021.

J. Pitman. Some Developments of the Blackwell-Macqueen URN Scheme. *Lecture Notes-Monograph Series*, 30:245–267, 1996. ISSN 07492170. URL `http://www.jstor.org/stable/4355949`. Publisher: Institute of Mathematical Statistics.

R. L. Plackett. The analysis of permutations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 24(2):193–202, 1975. Publisher: Wiley Online Library.

S. Prillo and J. Eisenschlos. SoftSort: A Continuous Relaxation for the argsort Operator. In *Proceedings of the 37th International Conference on Machine Learning, {ICML} 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7793–7802. PMLR, 2020. URL `http://proceedings.mlr.press/v119/prillo20a.html`.

G.-C. Rota. The Number of Partitions of a Set. *The American Mathematical Monthly*, 71(5):498, May 1964. ISSN 00029890. doi: 10.2307/2312585. Publisher: JSTOR.

Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99, 2000. Publisher: Springer Nature BV.

S. Sabour, N. Frosst, and G. E. Hinton. Dynamic Routing Between Capsules. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/2cad8fa47bbef282badbb8de5374b894-Paper.pdf`.

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

O. Sener and V. Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

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

T. M. Sutter, L. Manduchi, A. Ryser, and J. E. Vogt. Learning Group Importance using the Differentiable Hypergeometric Distribution. In *International Conference on Learning Representations*, 2023.

L. L. Thurstone. A law of comparative judgment. In *Scaling*, pages 81–92. Routledge, 1927.

K. T. Wallenius. Biased sampling; the noncentral hypergeometric probability distribution. Technical report, Stanford Univ Ca Applied Mathematics And Statistics Labs, 1963.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, Sept. 2017. URL `http://arxiv.org/abs/1708.07747`. arXiv:1708.07747 [cs, stat].

S. M. Xie and S. Ermon. Reparameterizable subset sampling via continuous relaxations. *arXiv preprint arXiv:1901.10517*, 2019.

D. Xin, B. Ghorbani, A. Garg, O. Firat, and J. Gilmer. Do Current Multi-Task Optimization Methods in Deep Learning Even Help? *arXiv preprint arXiv:2209.11379*, 2022.

J. I. Yellott. The relationship between Luce's choice axiom, Thurstone's theory of comparative judgment, and the double exponential distribution. *Journal of Mathematical Psychology*, 15(2): 109–144, 1977. Publisher: Elsevier.