Differentiable Clustering and Partial Fenchel-Young Losses

Lawrence Stewart [1] **Francis Bach** [1] **Felipe Llinares-López** [2] **Quentin Berthet** [2]

Abstract

We introduce a differentiable clustering method based on stochastic perturbations of minimum-weight spanning forests. This allows us to include clustering in end-to-end trainable pipelines, with efficient gradients. We show that our method performs well even in difficult settings, such as data sets with high noise and challenging geometries. We also formulate an ad hoc loss to efficiently learn from partial clustering data using this operation. We demonstrate its performance on several data sets for supervised and semi-supervised tasks.

1. Differentiable clustering

1.1. Clustering with k-spanning forests

The complete set of notations used throughout this paper can be found in Appendix A. In this work, the core operation is to cluster n elements, using a similarity matrix $\Sigma \in \mathcal{S}_n$. Informally, pairs of elements (i, j) with high similarity Σ_{ij} should be more likely to be in the same cluster than those with low similarity. The clustering is represented in the following manner.

Definition 1.1 (Cluster connectivity matrix)**.** Let $\pi : [n] \to [k]$ be a partition function, assigning n elements to one of k clusters. We represent it with a $n \times n$ binary cluster connectivity matrix M (also known as the coincidence matrix)

$$M_{ij} = 1 \quad \text{if and only if} \quad \pi(i) = \pi(j)\,.$$

We denote by $\mathcal{B}_k \subseteq \{0,1\}^{n\times n}$ the set of binary cluster connectivity matrices with k clusters.

Using this definition, we define an operation $M_k^*(\cdot)$ mapping a similarity matrix Σ to a clustering, in the form of such a connectivity matrix. Up to a permutation (i.e., naming the clusters), M allows to recover π entirely. It is based on a maximum spanning forest primitive $A_k^*(\cdot)$, and both are defined below.

Definition 1.2 (Maximum k-spanning forest)**.** Let Σ be an $n \times n$ similarity matrix. We denote by $A_k^*(\Sigma)$ the adjacency matrix of the k-spanning forest with maximum similarity, defined as

$$A_k^*(\Sigma) = \underset{A \in \mathcal{C}_k}{\operatorname{argmax}} \langle A, \Sigma\rangle.$$

This defines a mapping $A_k^* : \mathcal{S}_n \to \mathcal{C}_k$. We denote by F_k the value of this maximum, i.e.,

$$F_k(\Sigma) = \max_{A \in \mathcal{C}_k} \langle A, \Sigma\rangle.$$

Definition 1.3 (Spanning forest clustering)**.** Let A be the adjacency matrix of a k-spanning forest. We denote by $M^*(A)$ the connectivity matrix of the k connected components of A, i.e.,

$$M_{ij} = 1 \quad \text{if and only if} \quad i \sim j\,.$$

Given an $n \times n$ similarity matrix Σ, we denote by $M_k^*(\Sigma) \in \mathcal{B}_k$ the clustering induced by the maximum k-spanning forest, defined by

$$M_k^*(\Sigma) = M^*(A_k^*(\Sigma))\,.$$

This defines a mapping $M_k^* : \mathcal{S}_n \to \mathcal{B}_k$, our clustering operator.

Remarks. Both $A_k^*(\Sigma)$ and $M_k^*(\Sigma)$ can be obtained using a modified version of Kruskal's algorithm (Kruskal, 1956). This is equivalent to single linkage hierarchical agglomerative clustering with cluster merging carried out until k clusters remain (for more details see Appendix B). The mapping M^* is of course many-to-one, and yields a partition of $\mathcal{C}_k$ into equivalence classes of k-forests that yield the same clusters. Further illustrations are given in Appendix D.

Clustering with constraints. We introduce constrained versions of the M_k^* and A_k^* in order to infer clusterings and spanning forests that are consistent with observed information e.g. weak supervision. We represent these enforced connectivity values as a matrix M_Ω defined as follows.

Definition 1.4 (Partial cluster connectivity matrix)**.** Let M be a cluster connectivity matrix, and Ω a subset of $[n] \times [n]$,

[*]Equal contribution [1]Ecole Normale Superieure and INRIA, Paris. [2]Google DeepMind, Paris.. Correspondence to: Lawrence Stewart <lawrence.stewart@ens.fr>.

Published at the Differentiable Almost Everything Workshop of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. July 2023. Copyright 2023 by the author(s).

representing the set of observations. We denote by M_Ω the $n \times n$ matrix

$$M_{\Omega,ij} = M_{ij} \quad \text{if } (i,j) \in \Omega\,, \qquad M_{\Omega,ij} = * \quad \text{otherwise}\,.$$

Remarks. The symbol "$*$" in this definition is only used as a placeholder, an indicator of "no information", and does not have a mathematical use. $M_{\Omega,ij} = 1$ corresponds to a *must-link* constraint, whilst $M_{\Omega,ij} = 0$ corresponds to a *must-not-link* constraint.

Definition 1.5 (Constrained maximum k-spanning forest)**.** Let Σ be an $n \times n$ similarity matrix. We denote by $A^*_k(\Sigma, M_\Omega)$ the adjacency matrix of the k-spanning forest with maximum similarity, constrained to satisfy the connectivity constraints in M_Ω. It is defined as

$$A^*_k(\Sigma, M_\Omega) = \operatorname*{argmax}_{A \in \mathcal{C}_k(M_\Omega)} \langle A, \Sigma\rangle\,,$$

where for any partial clustering matrix M_Ω, $\mathcal{C}_k(M_\Omega)$ is the set of k-spanning forests whose clusters agree with M_Ω, i.e.,

$$\mathcal{C}_k(M_\Omega) = \{A \in \mathcal{C}_k \,:\, M^*(A)_{ij} = M_{\Omega,ij}\ \forall\,(i,j) \in \Omega\}\,.$$

For any partial connectivity matrix M_Ω, this defines another mapping $A^*(\cdot, M_\Omega) : \mathcal{S}_n \to \mathcal{C}_k$.

We denote by $F_k(\Sigma, M_\Omega)$ the value of this maximum, i.e.,

$$F_k(\Sigma, M_\Omega) = \max_{A \in \mathcal{C}_k(M_\Omega)} \langle A, \Sigma\rangle\,.$$

1.2. Perturbed Clustering

The clustering operations defined above are efficient and perform well (see Figure 2) but by their nature as discrete operators, they have a major drawback: they are piece-wise constant and as such cannot be conveniently included in end-to-end differentiable pipelines, such as those used to train models such as neural networks. To overcome this obstacle, we use a *perturbed proxy* (Abernethy et al., 2016; Berthet et al., 2020; Paulus et al., 2020; Struminsky et al., 2021). In these definitions and the following, we consider $Z \sim \mu$ from a distribution with positive, differentiable density over $\mathcal{S}_n$, and $\varepsilon > 0$, following the methodology of Berthet et al. (2020).

Definition 1.6. We define the *perturbed maximum spanning forest* as the expected maximum spanning forest under stochastic perturbation on the inputs. Formally, for a similarity matrix Σ, we have

$$A^*_{k,\varepsilon}(\Sigma) = \mathbf{E}[A^*_k(\Sigma + \varepsilon Z)] = \mathbf{E}\big[\operatorname*{argmax}_{A \in \mathcal{C}_k} \langle A, \Sigma + \varepsilon Z\rangle\big]\,,$$

$$F_{k,\varepsilon}(\Sigma) = \mathbf{E}[F_k(\Sigma + \varepsilon Z)]\,.$$

We define analogously $A^*_{k,\varepsilon}(\Sigma; M) = \mathbf{E}[A^*_k(\Sigma + \varepsilon Z; M)]$ and $F_{k,\varepsilon}(\Sigma; M) = \mathbf{E}[F_k(\Sigma + \varepsilon Z; M)]$, as well as clustering $M^*_{k,\varepsilon}(\Sigma;) = \mathbf{E}[M^*_k(\Sigma + \varepsilon Z)]$ and $M^*_{k,\varepsilon}(\Sigma; M_\Omega) = \mathbf{E}[M^*_k(\Sigma + \varepsilon Z; M_\Omega)]$.

We note that this defines operations $A^*_{k,\varepsilon}(\cdot)$ and $A^*_{k,\varepsilon}(\cdot, M_\Omega)$ mapping $\Sigma \in \mathcal{S}_n$ to cvx$(\mathcal{C}_k)$, the *convex hull* of $\mathcal{C}_k$. These operators have several advantageous features: they are differentiable, and both their values and their derivatives can be estimated using Monte-Carlo methods, by averaging copies of $A^*_k(\Sigma + \varepsilon Z^{(i)})$.

This is particularly convenient, as it does not require to implement a different algorithm to compute the differentiable version. Moreover, the use of parallelization in modern computing hardware makes the computational overhead almost nonexistent.

Our method allows for clustering on learn-able representations, without breaking the differentiability of a pipeline. These representations are informed by gradients transmitted backward through the clustering algorithm. Since $M^*_{k,\varepsilon}(\cdot)$ is a differentiable operator from $\mathcal{S}_n$ to cvx$(\mathcal{B}_k)$, it is possible to use any loss function on cvx$(\mathcal{B}_k)$ to design a loss based on Σ and some ground-truth clustering information M_Ω, such as $L(\Sigma; M_\Omega) = \|M^*_{k,\varepsilon}(\Sigma) - M_\Omega\|_2^2$. This flexibility is one of the advantages of our method.

In the following section, we introduce a loss tailored to be efficient to compute and performant in several learning tasks, that we call a *partial Fenchel-Young loss*.

2. Learning with differentiable clustering

2.1. Partial Fenchel-Young losses

The framework of *Fenchel-Young* losses (Blondel et al., 2020) can tackle complex structures which can be encoded as an LP, such as spanning k-forests. However, in most cases, clustering data can reasonably only be expected to be present as a connectivity matrix, which unfortunately does not take the form of an argmax of an LP. To address this problem, we introduce the *Partial Fenchel-Young* loss, which leverages the relation between M^*_k and A^*_k (the prior partitions the latter into equivalence classes).

Definition 2.1 (Partial Fenchel-Young loss)**.** Let F be a convex function, $\mathcal{C}$ a convex set and L_{FY} the associated Fenchel-Young loss. For every $p \in \mathcal{P}$ a convex constraint subset $\mathcal{C}(p) \subseteq \mathcal{C}$, we define:

$$\bar{L}_{\mathsf{FY}}(\theta, p) = \min_{y \in \mathcal{C}(p)} L_{\mathsf{FY}}(\theta; y)\,.$$

This allows to learn from incomplete information about y, when we do not know its value, but only a subset of $\mathcal{C}(p) \subseteq \mathcal{C}$ to which it might belong, we can minimize the infimum of the FY losses that are consistent with the partial label information $y \in \mathcal{Y}(p)$.

Proposition 2.2. *When F is the support function of a compact convex set $\mathcal{C}$, the partial Fenchel-Young loss (see Definition 2.1) satisfies:*

1. *The loss is a difference of convex functions in θ given explicitly by*
$$\bar{L}_{\mathsf{FY}}(\theta, p) = F(\theta) - F(\theta; p)\,,$$
$$\textit{where} \quad F(\theta; p) = \max_{y \in \mathcal{C}(p)} \langle y, \theta \rangle,$$
2. *The gradient with respect to θ is given by*
$$\nabla_\theta \bar{L}_{\mathsf{FY}}(\theta, p) = y^*(\theta) - y^*(\theta; p)\,,$$
$$\textit{where} \quad y^*(\theta; p) = \operatorname*{argmax}_{y \in \mathcal{C}(p)} \langle y, \theta \rangle.$$
3. *The perturbed partial Fenchel-Young loss given by $\bar{L}_{\mathsf{FY},\varepsilon}(\theta; p) = \mathbf{E}[\bar{L}_{\mathsf{FY}}(\theta + \varepsilon Z; p)]$ satisfies*
$$\nabla_\theta \bar{L}_{\mathsf{FY},\varepsilon}(\theta, p) = y^*_\varepsilon(\theta) - y^*_\varepsilon(\theta; p)\,,$$
$$\textit{where} \quad y^*_\varepsilon(\theta; p) = \mathbf{E}[\operatorname*{argmax}_{y \in \mathcal{C}(p)} \langle y, \theta + \varepsilon Z \rangle]\,.$$

Another possibility would be to define the partial loss as $\min_{y \in \mathcal{C}(p)} L_{\mathsf{FY},\varepsilon}(\theta; y)$, that is, the infimum of smoothed losses instead of the smoothed infimum loss $L_{\mathsf{FY},\varepsilon}(\theta; p)$ defined above. However, there is no direct method to minimizing the smoothed loss with respect to $y \in \mathcal{C}(p)$.

Proposition 2.3. *Letting $L_{\mathsf{FY},\varepsilon}(\theta; y) = \mathbf{E}[L_{\mathsf{FY},\varepsilon}(\theta + \varepsilon Z; y)]$ and $\bar{L}_{\mathsf{FY},\varepsilon}$ as in Definiton 2.1, we have*
$$\bar{L}_{\mathsf{FY},\varepsilon}(\theta; p) \leq \min_{y \in \mathcal{C}(p)} L_{\mathsf{FY},\varepsilon}(\theta; y)\,.$$

The proofs of the above propositions are given in Appendix E.

2.2. Applications to differentiable clustering

We apply this framework, as detailed in the following section and in Section 3, to clustering. This is done naturally by transposing notations and taking $\mathcal{C} = \mathcal{C}_k$, $\theta = \Sigma$, $y^* = A^*_k$, $p = M_\Omega$, $\mathcal{C}(p) = \mathcal{C}_k(M_\Omega)$, and $y^*(\theta, p) = A^*_k(\Sigma; M_\Omega)$. In this setting the perturbed partial Fenchel-Young loss satisfies
$$\nabla_\Sigma \bar{L}_{\mathsf{FY},\varepsilon}(\Sigma, M_\Omega) = A^*_{k,\varepsilon}(\Sigma) - A^*_{k,\varepsilon}(\Sigma; M_\Omega)\,.$$

We learn representations of a data that fit with clustering information (either complete or partial). As described above, we consider settings with n elements described by their features $\mathbf{X} = X_1, \ldots, X_n$ in some feature space $\mathcal{X}$, and M_Ω some clustering information. Our pipeline to learn representations includes the following steps (see Figure 1)

i) Embed each X_i in $\mathbb{R}^d$ with a parameterized model $v_i = \Phi_w(X_i) \in \mathbb{R}^d$, with weights $w \in \mathbb{R}^p$.

ii) Construct a similarity matrix from these embeddings, e.g. $\Sigma_{w,ij} = -\|\Phi_w(X_i) - \Phi_w(X_j)\|_2^2$.

iii) Stochastic optimization of the expected loss of $\bar{L}_{\mathsf{FY},\varepsilon}(\Sigma_{w,b}, M_{\Omega,b})$, using mini-batches of $\mathbf{X}$.

Further details on each of these steps is detailed in Appendix C.

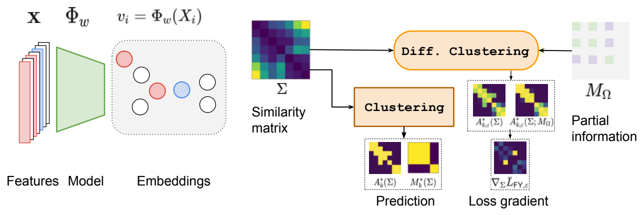


Figure 1. Our pipeline, in a semi-supervised example: data points are embedded by a parameterized model, which produces a similarity matrix. Partial clustering information may be available, in the form of must-link or must-not-link constraints.

3. Experiments

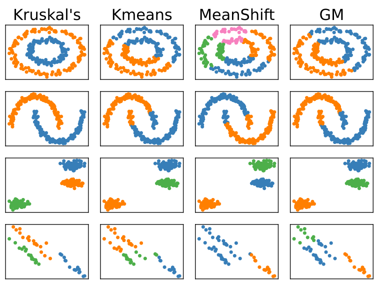
We apply our framework for learning through clustering in both a supervised and a semi-supervised setting, as illustrated in Figure 1. Formally, for large training data sets of size n, we either have access to a full cluster connectivity matrix M_Ω or a partial one (typically built by using partial label information, see below).

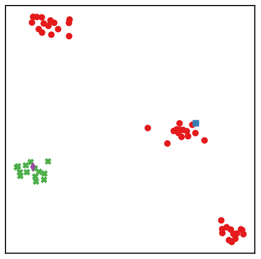
We use this clustering information M_Ω, from which minibatches can be extracted, as supervision. We minimize our partial Fenchel-Young loss with respect to the weights of an embedding model, and evaluate these embeddings in two main manners on a test dataset: first, by evaluating the clustering accuracy (i.e. proportion of correct coefficients in the predicted cluster connectivity matrix), and second by training a shallow model on a classification task (using clusters as classes) on a holdout set, evaluating it on a test set.

3.1. Supervised learning

We apply first our method to synthetic data sets - purely to provide an illustration of both our internal clustering algorithm, and of our learning procedure. In Figure 2, we show how the clustering operator that we use, based on spanning forests (i.e. single linkage), with Kruskal's algorithm is efficient on some standard synthetic examples, even when they are not linearly separable (compared to k-Means, mean-shift, Gaussian mixture-model). We also show that our method allows us to perform supervised learning based on cluster information, in a linear setting. For X_{signal} consisting of $n = 60$ points in two dimensions consisting of data in four well-separated clusters (see Figure 2), we construct X by appending two noise dimensions, such that clustering based on pairwise square distances between X_i mixes the original clusters. We learn a linear de-noising transformation θX, with $\theta \in \mathbb{R}^{4\times 2}$ through clustering, by minimizing our perturbed partial FY loss with SGD, using the known labels as supervision.

We also show that our method is able to cluster virtually all of some classical data sets. We train a CNN (LeNet-5





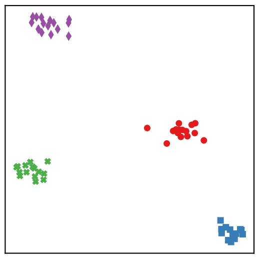
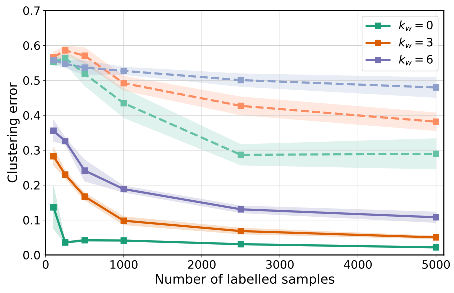
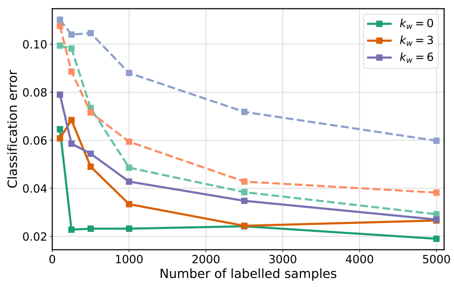


Figure 2. Comparing clustering methods **(left)**, and learning a linear representation with clustering information from noised signal **(middle)** and after training **(right)**.





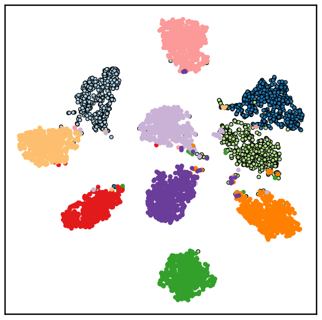


Figure 3. Performance on semi-supervised learning task. Our trained model (full line) is evaluated on clustering (**left**) and classification errors (**middle**), compared to a model entirely trained on the classification task (dashed line). The **rightmost** figure shows the t-SNE of learnt embeddings for MNIST with $k_w = 3$ withheld classes (highlighted).

(LeCun et al., 1998)) on mini-batches of size 64 using the partial Fenchel-Young loss to learn a clustering, with a batch-wise clustering precision of 0.99 for MNIST and 0.96 on Fashion MNIST (Xiao et al., 2017). The experimental setup and visualization of learnt clusters is detailed in Appendix F.

3.2. Semi-supervised learning

We show that our method is particularly useful in settings where labelled examples are scarce, even in the particularly challenging case of having no labelled examples for some classes. To this end, we conduct a series of experiments on the MNIST dataset (LeCun et al., 2010) where we independently vary both the total number of labelled examples $n_\ell = \{100, 250, 500, 1000, 2500, 5000\}$ as well as the number of withheld classes for which no labelled examples are present in the training set $k_w = \{0, 3, 6\}$. We train the same embedding model Φ_w as above (CNN LeNet-5), using our partial Fenchel-Young loss with batches of size 64. We use $\varepsilon = 0.1$ and $B = 100$ for the estimated loss gradients, and optimize weights using Adam (Kingma & Ba, 2015). The two metrics reported in Figure 3, both on a test set are clustering error (evaluated on mini-batches of the same size) and classification error, evaluated by training an additional linear layer (freezing our CNN) on hold-out data with all the classes present. We compare both metrics to the performance of a model composed of an identical CNN, with an additional linear head, trained on the same data using the cross-entropy loss (see Appendix F for further details).

We observe that learning through clustering allows to find a representation where class semantics are easily recoverable from the local topology. Strikingly, our proposed approach achieves a lower clustering error in the most challenging setting ($n_\ell = 100$ labelled examples and $k_w = 6$ withheld classes) than the classification-based baseline in the most lenient setting ($n_\ell = 5000$ labelled examples and all classes observed). Most importantly, this advantage is not limited to clustering metrics: learning through clustering also leads to a lower classification error for all conditions under study, with the gap being most pronounced when few labelled examples are available.

Moreover, besides this pronounced improvement in data efficiency, we found (see Figure 2.2) that our method is capable of clustering classes for which no labelled examples are seen during training. Therefore, investigating potential applications of learning through clustering to zero-shot and self-supervised learning are promising avenues for future work.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: a system for large-scale machine learning. In *Osdi*, volume 16, pp. 265–283. Savannah, GA, USA, 2016.

Abernethy, J., Lee, C., and Tewari, A. Perturbation techniques in online learning and optimization. *Perturbations, Optimization, and Statistics*, 233, 2016.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. Learning with differentiable pertubed optimizers. *Advances in Neural Information Processing Systems*, 33:9508–9519, 2020.

Blondel, M., Martins, A. F., and Niculae, V. Learning with Fenchel-Young losses. *The Journal of Machine Learning Research*, 21(1):1314–1382, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Defazio, A., Bach, F., and Lacoste-Julien, S. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in Neural Information Processing Systems*, 27, 2014.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

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

LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Paulus, M., Choi, D., Tarlow, D., Krause, A., and Maddison, C. J. Gradient estimation with stochastic softmax tricks. *Advances in Neural Information Processing Systems*, 33: 5691–5704, 2020.

Struminsky, K., Gadetsky, A., Rakitin, D., Karpushkin, D., and Vetrov, D. P. Leveraging recursive gumbel-max trick for approximate inference in combinatorial spaces. *Advances in Neural Information Processing Systems*, 34: 10999–11011, 2021.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

A. Notations

For a positive integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$. We denote $\mathcal{S}_n$ as the set of $n \times n$ symmetric matrices. We consider all graphs to be undirected. For the complete graph with n vertices K_n over nodes $[n]$, we denote by $\mathcal{T}$ the set of *spanning trees* on K_n, i.e., subgraphs with no cycles and one connected components. For any positive integer $k \leq n$ we also denote by $\mathcal{C}_k$ the set of k*-spanning forests* of K_n, i.e., subgraphs with no cycles and k connected components. With a slight abuse of notation, we also refer to $\mathcal{T}$ and $\mathcal{C}_k$ for the set of *adjacency matrices* of these graphs. For two nodes i and j in a general graph, we write $i \sim j$ if the two nodes are in the same connected component. We denote by $\mathcal{S}_n$ the set of $n \times n$ symmetric matrices.

B. Algorithms for Spanning Forests

As mentioned in Section 1, both $A^*_k(\Sigma)$ and $M^*_k(\Sigma)$ are calculated using Kruskal's algorithm (Kruskal, 1956). Our implementation of Kruskal's is tailored to our use: we first initialize both $A^*_k(\Sigma)$ and $M^*_k(\Sigma)$ as the identity matrix, and then sort the upper triangular entries of Σ. We build the maximum-weight spanning forest in the usual greedy manner, using $A^*_k(\Sigma)$ to keep track of edges in the forest and $M^*_k(\Sigma)$ to check if an edge can be added without creating a cycle, updating both matrices at each step of the algorithm. Once the forest has k connected components, the algorithm terminates. This is done by keeping track of the number of edges that have been added at any time.

We remark that our implementation takes the form as a single loop, with each step of the loop consisting only of matrix multiplications. For this reason it is fully compatible with auto-differentiation engines, such as JAX (Bradbury et al., 2018), Pytorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2016), and suitable for GPU/TPU acceleration. Therefore, our implementation differs from that of the standard Kruskal's algorithm, which used a disjoint union-find data structure (and hence is not compatible with auto-differentiation frameworks). Finally, our implementation of Kruskal's compatible with the perturbations module of the JAXOPT library, so that this operation and the Fenchel-Young losses can be automatically differentiated without requiring to be implemented.

B.1. Constrained Spanning Forests

To solve the constrained problem detailed in Section 1.5, we make the modifications below to our implementation of Kruskal's, under the assumption that M_Ω represents *valid* clustering information (i.e. with no contradiction):

1. **Regularization** (Optional) : It is possible to bias the optimization problem over spanning forests to encourage or discourage edges between some of the nodes, according to the clustering information. Before performing the sort on the upper-triangular of Σ, we add a large value to all entries of Σ_{ij} where $(M_\Omega)_{ij} = 1$, and subtract this same value from all entries of Σ_{ij} where $(M_\Omega)_{ij} = 1$. Entries Σ_{ij} corresponding to where $(M_\Omega)_{ij} = \star$ are left unchanged. This biasing ensures that any edge between points that are constrained to be in the same cluster will always be processed before unconstrained edges. Similarly, any edge between points that are constrained to not be in the same cluster, will be processed after unconstrained edges. In most cases, i.e. when all clusters are represented in the parial information, such as when $\Omega = [n] \times [n]$ (full information), this is not required to solve the constrained linear program, but we have found that this regularization was helpful in practice.

2. **Constraint enforcement** : We ensure that adding an edge does not violate the constraint matrix. In other words, when considering adding the edge (i, j) to the existing forest, we check that none of the points in the connected component of i are forbidden from joining any of the points in the connected component of j. This is implemented using further matrix multiplications and done alongside the existing check for cycles. The exact implementation is detailed in our code base.

C. Details of Pipeline

To be more specific on each of the three phases listed in Section 2.2 :

1. We embed each X_i individually with a model Φ_w, using in our application neural networks and a linear model. This allows us to use learning through clustering as a way to learn representations, and to apply this model to other elements,

for which we have no clustering information, or for use in other downstream tasks.

2. We focus on cases where the similarity matrix is the negative squared distances between those embeddings. This creates a connection between a model acting individually on each element, and a pairwise similarity matrix that can be used as an input for our differentiable clustering operator. This mapping, from w to Σ, has derivatives that can therefore be automatically computed by backpropagation, as it contains only conventional opereations (at least when Φ_w is itself a conventional model, such as commonly used neural networks).

3. We use our proposed smoothed partial Fenchel-Young (Section 2.1) as the objective to minimize between the partial information M_Ω and Σ. The full-batch version would be to minimize $L_{\mathsf{FY},\varepsilon}(\Sigma_w, M_\Omega)$ as a function of the parameters $w \in \mathbb{R}^p$ of the model. We focus instead on a mini-batch formulation for two reasons: first, stochastic optimization with mini-batches is a commonly used and efficient method for generalization in machine learning; second, it allows to handle larger-scale data sets. As a consequence, the stochastic gradients of the loss are given, for a mini-batch b, by

$$\begin{aligned}\nabla_w \bar{L}_{\mathsf{FY},\varepsilon}(\Sigma_{w,b}, M_{\Omega,b}) &= \partial_w \Sigma_w \cdot \nabla_\Sigma \bar{L}_{\mathsf{FY},\varepsilon}(\Sigma_{w,b}, M_{\Omega,b}) \\ &= \partial_w \Sigma_w \cdot \big(A^*_{k,\varepsilon}(\Sigma_w) - A^*_{k,\varepsilon}(\Sigma_w; M_{\Omega,b})\big)\,.\end{aligned}$$

The simplicity of these gradients is due to our particular choice of smoothed partial Fenchel-Young loss. They can be efficiently estimated automatically, as described in Section 1.2, which results in a doubly stochastic scheme for the loss optimization.

D. Illustration of Method

We recall that a k-forest on a graph is a subgraph with no cycles consisting in k connected components, potentially single nodes.

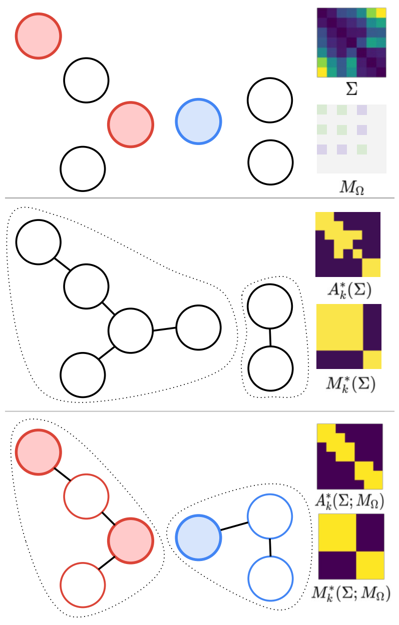


Figure 4. Method illustration, for $k = 2$ - **top** Similarity matrix based on pairwise square distance, partial cluster connectivity information - **center** Clustering using spanning forests without (partial) clustering constraints - **bottom** Constrained clustering.

E. Proofs of Technical Results

Proof of Proposition 2.2. We show these properties successively

1) We have by Definition 2.1

$$\bar{L}_{\mathsf{FY}}(\theta, p) = \min_{y \in \mathcal{C}(p)} L_{\mathsf{FY}}(\theta; y)\,.$$

We expand this

$$\bar{L}_{\mathsf{FY}}(\theta, p) = \min_{y \in \mathcal{C}(p)} \big\{F(\theta) - \langle y, \theta\rangle\big\} = F(\theta) - \max_{y \in \mathcal{C}(p)} \langle y, \theta\rangle\,.$$

As required, this implies, following the definitions given

$$\bar{L}_{\mathsf{FY}}(\theta, p) = F(\theta) - F(\theta; p)\,, \quad \text{where} \quad F(\theta; p) = \max_{y \in \mathcal{C}(p)} \langle y, \theta\rangle,$$

2) By linearity of derivatives and 1) above, we have

$$\nabla_\theta \bar{L}_{\mathsf{FY}}(\theta, p) = \nabla_\theta F(\theta) - \nabla_\theta F(\theta; p) = y^*(\theta) - y^*(\theta; p)\,, \quad \text{where} \quad y^*(\theta; p) = \operatorname*{argmax}_{y \in \mathcal{C}(p)} \langle y, \theta\rangle,$$

Since the argmax of a constrained linear optimization problem is the gradient of its value. We note that this property holds almost everywhere (when the argmax is unique), and almost surely for costs with positive, continuous density, which we always assume (e.g. see the following).

3) By linearity of expectation and 1) above, we have

$$\nabla_\theta \bar{L}_{\mathsf{FY},\varepsilon}(\theta, p) = \nabla_\theta \mathbf{E}[F(\theta + \varepsilon Z) - F(\theta + \varepsilon Z; p)] = \nabla_\theta F_\varepsilon(\theta) - \nabla_\theta F_\varepsilon(\theta; p) = y^*_\varepsilon(\theta) - y^*_\varepsilon(\theta; p)\,,$$

using the definition

$$y^*_\varepsilon(\theta; p) = \mathbf{E}[\operatorname*{argmax}_{y \in \mathcal{C}(p)} \langle y, \theta + \varepsilon Z\rangle]\,.$$

□

Proof of Proposition 2.3. By Jensen's inequality and the definition of the Fenchel-Young loss:

$$\begin{aligned}
\bar{L}_{\mathsf{FY},\varepsilon}(\theta; p) &= \mathbf{E}[\min_{y \in \mathcal{C}(p)} L_{\mathsf{FY}}(\theta + \varepsilon Z; y)] \\
&\leq \min_{y \in \mathcal{C}(p)} \mathbf{E}[L_{\mathsf{FY}}(\theta + \varepsilon Z; y)] \\
&= \min_{y \in \mathcal{C}(p)} F_\varepsilon(y) - \langle \theta, y\rangle \\
&\leq \min_{y \in \mathcal{C}(p)} L_{\mathsf{FY},\varepsilon}(\theta; y)\,.
\end{aligned}$$

□

F. Additional Experimental Information

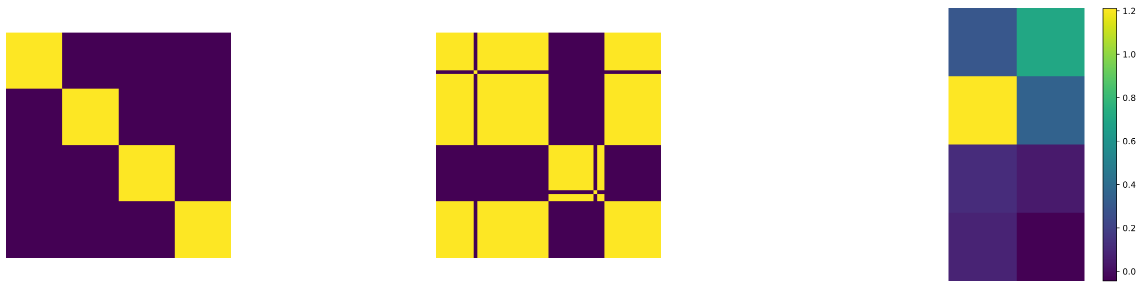


Figure 5. **left**: M_{signal}, **middle**: $M^*_4(\Sigma)$, **right**: θ after training.

We provide the details of the synthetic denoising experiment depicted in Figure 2 and described in Section 3.1. We create the signal data $X_{\text{signal}} \in \mathbb{R}^{60 \times 2}$ by sampling iid. from four isotropic Gaussians (15 points coming from each of the Gaussians) each having a standard deviation of 0.2. We randomly sample the means of the four Gaussians; for the example seen in Section 3.1 the sampled means were:

$$\begin{pmatrix} 0.97627008 & 4.30378733 \\ 2.05526752 & 0.89766366 \\ -1.52690401 & 2.91788226 \\ -1.24825577 & 7.83546002 \end{pmatrix}.$$

Let Σ_{signal} be the pairwise euclidean similarity matrix corresponding to X_{signal}, and furthermore let $M_{\text{signal}} := M_4^*(\Sigma_{\text{signal}})$ be the equivalence matrix corresponding to the signal (M_{signal} will be the target equivalence matrix to learn).

We append an additional two 'noise dimensions' to X_{signal} in order to form the train data $X \in \mathbb{R}^{60\times 4}$, where the noise entries were sampled iid from a continuous unit uniform distribution. Similarly letting Σ be the pairwise euclidean similarity matrix corresponding to X, we calculate $M_4^*(\Sigma) \neq M_{\text{signal}}$. Both the matrices M_{signal} and $M_4^*(\Sigma)$ are depicted in Figure 2 ; we remark that adding the noise dimensions leads to most points being assigned one of two clusters, and two points being isolated alone in their own clusters. We also create a validation set of equal size (in exactly the same manner as the train set), to ensure the model has not over-fitted to the train set.

The goal of the experiment is to learn a linear transformation of the data that recovers M_{signal} i.e. a denoising, by minimizing the partial loss. There are multiple solutions to this problem, the most obvious being a transformation that removes the last two noise columns from X:

$$\theta^* = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \text{for which} \quad X\theta^* = X_{\text{signal}}$$

For any $\theta \in \mathbb{R}^{4\times 2}$, we define Σ_θ to be the pairwise similarity matrix corresponding to $X\theta$, and $M_4^*(\Sigma_\theta)$ to its corresponding equivalence matrix. Then the problem can be summarized as:

$$\min_{\theta \in \mathbb{R}^{4\times 2}} \bar{L}_{\mathsf{FY},\varepsilon}(\Sigma_\theta, M_{\text{signal}}). \tag{1}$$

We initialized θ from a standard Normal distribution, and minimized the partial loss via stochastic gradient descent, with a learning rate of 0.01 and batch size 32.

For perturbations, we took $\varepsilon = 0.1$ and $B = 1000$, where ε denotes the noise amplitude in randomized smoothing and B denotes the number of samples in the Monte-Carlo estimate. The validation clustering error converged to zero after 25 gradient batches. We verify that the θ attained from training is indeed learning to remove the noise dimensions (see Figure 5).

F.1. Supervised Differentiable Clustering

As mentioned in Section 3.1, our method is able to cluster classical data sets such as MNIST and Fashion MNIST. We trained a CNN with the LeNet-5 architecture (LeCun et al., 1998) using our proposed partial loss as the objective function. The exact details of the CNN architecture are depicted in Figure 6. For this experiment and all experiments described below, we trained on a single Nvidia V100 GPU ; training the CNN with our proposed pipeline took < 15 minutes.

The model was trained for 30k gradient steps on mini-batches of size 64. We used the Adam optimizer (Kingma & Ba, 2015) with learning rate $\eta = 3 \times 10^{-4}$, momentum parameters $(\beta_1, \beta_2) = (0.9, 0.999)$, and an ℓ_2 weight decay of 10^{-4}. We validated / tested the model using the zero-one error between the true equivalence matrix and the equivalence matrix corresponding to the output of the model. We used an early stopping of 10k steps (i.e. training was stopped if the validation clustering error did not improve over 10k steps). For efficiency (and parallelization), we also computed this clustering error batch-wise with batch-size 64. As stated in Section 3.1, we attained a batch-wise clustering precision of 0.99 for MNIST and 0.96 on Fashion MNIST.

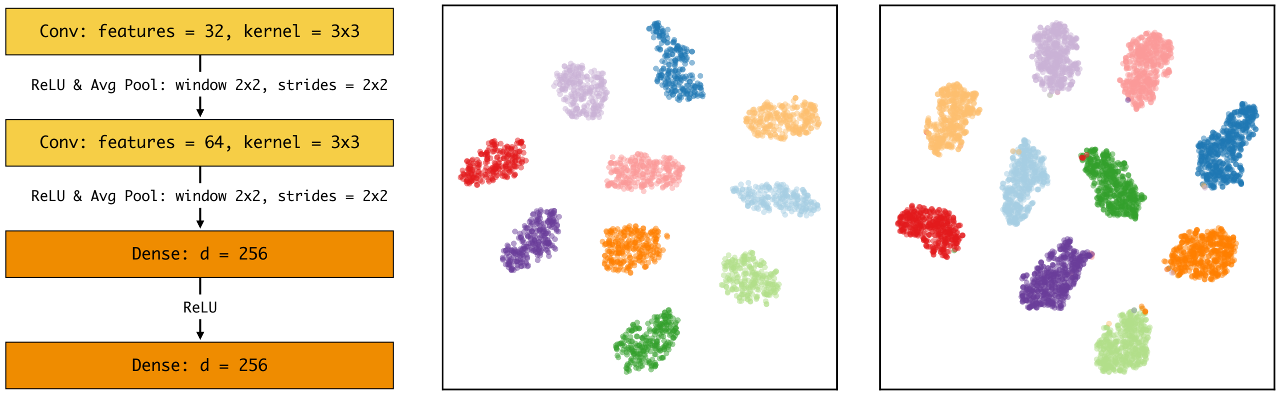

Conv: features = 32, kernel = 3x3
ReLU & Avg Pool: window 2x2, strides = 2x2
Conv: features = 64, kernel = 3x3
ReLU & Avg Pool: window 2x2, strides = 2x2
Dense: d = 256
ReLU
Dense: d = 256

Figure 6. **left**: Architecture of the CNN, **middle:** t-SNE visualization of train data embeddings, **right** : tSNE visualization of validation data embeddings.

The t-SNE visualizations of the embedding space of the model trained on MNIST for a collection of train and validation data points are depicted in Figure 6. It can be seen that the model has learnt a distinct cluster for each of the ten classes.

F.2. Semi-Supervised Differentiable Clustering

As mentioned in Section 3.2, we show that our method is particularly useful in settings where labelled examples are scarce, even in the particularly challenging case of having no labelled examples for some classes. Our approach allows a model to leverage the semantic information of unlabeled examples when trying to predict a target equivalence matrix M_Ω ; this is owing to the fact that the prediction of a class for a single point depends on the values of all other points in the batch, which is in general not the case for more common problems such as classification and regression.

To demonstrate the performance of our approach, we assess our method on two tasks:

1. **Semi-Supervised Clustering:** Train a CNN to learn an embedding of the data which leads to a good clustering error. We can compare our methodology to that of a baseline model trained using the cross-entropy loss. This is to check that our model has leveraged information from the unlabeled data and that our partial loss is indeed leading to good clustering performance.
2. **Downstream Classification:** Assess the trained model's capacity to serve as a backbone in a downstream classification task (transfer learning), where its weights are frozen and a linear layer is trained on top of the backbone.

We describe our data processing for both of these tasks below.

F.2.1. Data Sets

In our semi-supervised learning experiments, we divided the standard MNIST train split in the following manner:

- We create a balanced hold-out data set consisting of 1k images (100 images from each of the 10 classes). This hold-out data set will be used to assess the utility of the frozen clustering model on a downstream classification problem.
- From the remaining 59k images, we select a labeled train set of $n_\ell = \{100, 250, 500, 1000, 2500, 5000\}$ images. Our experiments also vary $k_w \in \{0, 3, 6\}$, the number of digits to withhold all labels from. For example, if $k_w = 3$, then the labels for the images corresponding to digits $\{0, 1, 2\}$ will never appear in the labeled train data.

F.2.2. Semi-Supervised Clustering Task

For each of the 18 data sets above, we train the same CNN architecture as that described in Section F.1 using the following two approaches:

1. **Ours:** The CNN is trained on mini-batches, where half the batch is labeled data and half the batch is unlabeled data (i.e. a semi-supervised learning regime), to minimize the partial loss.

2. **Baseline:** The baseline model consists of a CNN with the same architecture as that described in Section F.1, but with an additional dense layer with output dimension 10 (the number of classes). We train the model using mini-batches consisting of labeled points, minimizing the cross-entropy loss. The training regime is fully-supervised learning (classification). The baseline backbone refers to all of the model, minus the dense output layer.

Both models were trained with mini-batches of size 64, with points sampled uniformly without replacement. All hyper-parameters and optimization metrics were identical to those detailed in Section 3.1. We ran each experiment on each dataset for five different random seeds $s \in \{1, \dots, 5\}$, in order to report population statistics on the clustering error.

F.2.3. Transfer-Learning: Downstream Classification

In this task both models are frozen, and their utility as a foundational backbone is assessed on a downstream classification task using the hold-out data set. We train a linear (a.k.a dense) layer on top of both models using the cross-entropy loss. We refer to this linear layer as the downstream head. Training this linear head is equivalent to performing multinomial logistic regression on the features of the model.

To optimize the weights of the linear head we used the SAGA optimizer (Defazio et al., 2014). The results are depicted in Figure 3. It can be seen that training a CNN backbone using our approach with just 250 labels leads to better downstream classification performance than the baseline trained with 5000 labels. It is worth remarking that the baseline backbone was trained on the same objective function (cross-entropy) and task (classification) as the downstream problem, which is not the case for the backbone corresponding to our approach. This highlights how learning ‘cluster-able embeddings’ and leveraging unlabeled data can be desirable for transfer learning.