Deep embedded self-organizing maps for joint representation learning and topology-preserving clustering

Article *in* Neural Computing and Applications · December 2021

DOI: 10.1007/s00521-021-06331-w

CITATION
1

READS
459

4 authors:

Florent Forest
École Polytechnique Fédérale de Lausanne
13 PUBLICATIONS **115** CITATIONS
SEE PROFILE

Mustapha Lebbah
Université Paris 13 Nord
154 PUBLICATIONS **550** CITATIONS
SEE PROFILE

Hanene Azzag
Université Paris 13 Nord
57 PUBLICATIONS **321** CITATIONS
SEE PROFILE

Jérôme Lacaille
Safran Aircraft Engines
125 PUBLICATIONS **586** CITATIONS
SEE PROFILE

Neural Computing and Applications
https://doi.org/10.1007/s00521-021-06331-w

ORIGINAL ARTICLE

Deep embedded self-organizing maps for joint representation learning and topology-preserving clustering

Florent Forest[1,2] · Mustapha Lebbah[1] · Hanene Azzag[1] · Jérôme Lacaille[2]

Received: 23 January 2021 / Accepted: 10 July 2021

Abstract
A recent research area in unsupervised learning is the combination of representation learning with deep neural networks and data clustering. The success of deep learning for supervised tasks is widely established. However, recent research has demonstrated how neural networks are able to learn representations to improve clustering in their intermediate feature space, using specific regularizations. By considering representation learning and clustering as a joint task, models learn clustering-friendly spaces and outperform two-stage approaches where dimensionality reduction and clustering are performed separately. Recently, this idea has been extended to topology-preserving clustering models, known as self-organizing maps (SOM). This work is a thorough study on the deep embedded self-organizing map (DESOM), a model composed of an autoencoder and a SOM layer, training jointly the code vectors and network weights to learn SOM-friendly representations. In other words, SOM induces a form a regularization to improve the quality of quantization and topology in latent space. After detailing the architecture, loss and training algorithm, we study hyperparameters with a series of experiments. Different SOM-based models are evaluated in terms of clustering, visualization and classification on benchmark datasets. We study benefits and trade-offs of joint representation learning and self-organization. DESOM achieves competitive results, requires no pretraining and produces topologically organized visualizations.

Keywords Self-organizing maps · Clustering · Visualization · Representation learning · Deep learning · Autoencoder

1 Introduction

After the successes of deep neural networks on supervised tasks such as classification and regression in various domains during the last decade, recent research has focused on learning representations with neural networks for unsupervised tasks, and cluster analysis in particular. Representation learning for clustering is currently a very active research area.

Clustering algorithms are a family of unsupervised learning algorithms that try to find groups of similar elements in a data set. However, traditional clustering algorithms tend to be ineffective on high-dimensional data where similarity measures become meaningless. An intuitive solution is to first reduce the dimensionality of data (while minimizing information loss) and then cluster the data in a low-dimensional space. This can be achieved by using for example linear dimensionality reduction techniques such as PCA, or nonlinear models with more expressive power such as deep autoencoders. In this two-stage approach, we:

1. Optimize a pure information loss criterion between data points and their low-dimensional embeddings (this generally takes the form of a reconstruction loss between a data point and its reconstruction, e.g., mean squared error).

✉ Florent Forest
forest@lipn.univ-paris13.fr

Mustapha Lebbah
mustapha.lebbah@lipn.univ-paris13.fr

Hanene Azzag
azzag@univ-paris13.fr

Jérôme Lacaille
jerome.lacaille@safrangroup.com

[1] LIPN (CNRS UMR 7030), Université Sorbonne Paris Nord, 99 Avenue Jean-Baptiste Clément, 93430 Villetaneuse, France

[2] Safran Aircraft Engines, Rond-point René Ravaud, 77550 Moissy-Cramayel, France

Published online: 03 August 2021

2. Optimize a pure clustering criterion using some clustering algorithm (e.g., *k*-means quantization error).

In contrast, so-called *deep clustering* approaches treat clustering and representation learning as a joint task and focus on learning representations that are more *clustering-friendly*. The principle is to learn a representation space that preserves a specific prior knowledge, in this case, cluster structure.

We focus on a specific family of clustering algorithms called self-organizing map models, which perform simultaneous clustering and visualization by projecting high-dimensional data onto a low-dimensional map (typically two-dimensional for visualization purpose) having a grid topology. The grid is composed of *units*, also called *neurons* or *cells*. Each map unit is associated with a *prototype vector* from the original data space (also called *code vector* indifferently). Self-organizing map algorithms enforce a constraint on the topology of the map, so that neighboring units on the map correspond to prototype vectors that are close in the original high-dimensional space, according to a distance metric. The most well-known self-organized model is Kohonen's self-organizing map (SOM) [27, 28]. Due to their visualization capability and interpretability, SOMs have been applied across various applications: in the aerospace industry [5, 12, 14, 15], public health [19, 34], volcanology [4], among many others. As for clustering, self-organized models could benefit from representation learning with neural networks. SOM is part of the more general family of learning vector quantizers (LVQ), and the possible combination of deep architectures with LVQ has already been realized [45].

The deep embedded self-organizing map (DESOM) model and some of its variants were introduced in previous works [16, 17]. Studying this particular model is the main subject of this work, but the combination of autoencoders and self-organizing maps has been tackled in several other concurrent works [13, 19, 34, 40]. In a quite different approach [10], a SOM is trained on the features extracted by pre-trained convolutional networks. More details will be given in the background section.

The DESOM model has already been applied in the literature, to analyze energy consumption of buildings for smart energy management [43]. It was used in [36] and extended to jointly learn feature relevance weights. Finally, it has been applied to seismic facies data in [31]. DESOM outperformed the other two-stage dimensionality reduction + SOM methods (such as autoencoder+SOM), and authors introduced a variant called SDESOM (Sparse DESOM) with a sparsity constraint on the latent space, which improved feature extraction and clustering performance.

The goal of joint representation learning and self-organization is to (1) learn the representation space and the SOM map simultaneously, without using a two-stage approach; (2) find a latent space that is more adapted to the SOM algorithm, according to some quality metric. Using the term coined by [50], we seek a *SOM-friendly* space. The SOM prototypes are then learned in latent space. To learn this new representation, we use an autoencoder neural network, composed of an encoder network that maps data points to the latent space, and a decoder network that reconstructs latent points into vectors of the original data space. For visualization and interpretation of the map, we need the prototypes to lie in the original feature space, so we reconstruct them using the decoder part of the autoencoder network. This approach very much resembles joint representation learning and clustering, but with an additional topology constraint. Our experiments show that this approach has clear advantages:

- Autoencoders with sufficiently high capacity yield meaningful low-dimensional representations of high-dimensional data that facilitate SOM learning and improve clustering performance.
- Self-organization and representation learning can be achieved in a single joint task, without a separate preprocessing step, thus cutting down overall training time.

After this introduction, the rest of the paper is organized as follows:

1. A background and related work section introduces concepts, mathematical notations and references on self-organizing maps, autoencoders and representation learning for clustering and SOM.
2. A section is devoted to introducing our main contribution, titled: Deep Embedded SOM (DESOM).
3. The benchmark datasets used in this paper are described in a datasets section.
4. The experiments section presents results of studies on DESOM's hyperparameters and training process.
5. Results of a large clustering and classification benchmark are presented in the last section.

Finally, we conclude by summarizing the main points of our work and discuss future perspectives. Appendix 1 details all performance metrics that are used throughout the paper to evaluate and compare different aspects of models. Appendix 2 showcases two examples of large map visualizations. Before going on, here are the main contributions of this work:

- We provide a background on a novel field, which is the combination of self-organized clustering algorithms and representations learning through neural networks.
- We present the Deep Embedded SOM (DESOM), a model that jointly trains an autoencoder and a self-

organizing map, using stochastic gradient descent. The model was already introduced in previous works [16, 17], with limited experiments and analyses. In this paper, we conduct more thorough quantitative and qualitative studies.
- We study the architecture, hyperparameters and training dynamics of DESOM through extensive experiments and visualizations.
- We perform a benchmark of several clustering and SOM-based models on standard datasets, evaluating clustering performance, visual quality and classification power.
- In the appendix, we present a collection of internal and external metrics to evaluate different aspects of SOM performance.

All the codes (model and performance metrics) are available online as open-source projects[1,2].

2 Background and related work

This section starts by first providing background on self-organizing maps and autoencoders, and will introduce notations that will be used throughout the rest of the paper. Secondly, we will sift through recent research on joint representation learning for clustering, and more specifically deep self-organizing maps.

2.1 Self-organizing maps

The self-organizing map (SOM) [27, 28] is a bio-inspired clustering model that introduces a topological relationship between clusters. It consists in a network of two layers: an input layer, and an output layer of interconnected nodes, often called *neurons* or *units*. Typically, the topology of this layer is chosen as a two-dimensional grid, because it can be easily visualized. This visualization capability characterizes SOM as an interpretable clustering method.

The set of input data samples is denoted $\mathbb{X} = \{\mathbf{x}_i\}_{1 \leq i \leq N}, \mathbf{x}_i \in \mathbb{R}^D$. A self-organizing map is composed of K units, associated with the set of prototype vectors $\{\mathbf{m}_k\}_{1 \leq k \leq K}$. In the standard SOM, the prototype vectors lie in the same space as the input data, i.e., $\mathbb{R}^D$. A data point is projected on the map by finding its closest prototype vector according to Euclidean distance. The corresponding map unit is called the *best-matching unit* (BMU). We introduce the notation b_i for the BMU of $\mathbf{x}_i$:

$$b_i = \underset{k}{\operatorname{argmin}} ||\mathbf{x}_i - \mathbf{m}_k||_2^2 \tag{1}$$

The grid topology allows to define an inter-node distance $\delta(k, l)$, which is the topographic distance between units k and l on the map, here the Manhattan distance (the length of the shortest path on the map between the two units). We then define the neighborhood function of the SOM and a temperature parameter T, controlling the radius of the neighborhood around a unit. In this work, we will use a Gaussian neighborhood function, expressed as follows:

$$\mathcal{K}^T(d) = e^{-\frac{d^2}{T^2}}$$

The temperature T is decreased at each training iteration, as in simulated annealing. A common choice is exponential decay, starting from an initial temperature T_{max} toward a final temperature T_{min}, i.e., at iteration i:

$$T(i) = T_{max}\left(\frac{T_{min}}{T_{max}}\right)^{i/\text{iterations}}$$

The original SOM learning algorithm, also called *stochastic algorithm* or *Kohonen algorithm*, takes each training sample $\mathbf{x}_i$ and updates every prototype vector by moving them closer to the point $\mathbf{x}_i$. The updates are weighted by the neighborhood around the best-matching unit, so that neighboring units receive a large update and very distant units are not updated at all. This expresses as the following update rule:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k + \alpha \mathcal{K}^T(\delta(b_i, k))(\mathbf{x}_i - \mathbf{m}_k) \tag{2}$$

where α is a learning rate that is decreased during training. The stochastic algorithm is detailed in algorithm 1.

[1] https://github.com/FlorentF9/DESOM.

[2] https://github.com/FlorentF9/SOMperf.

Input: training set $\mathbb{X}$; SOM map size; temperatures T_{max}, T_{min}; *iterations*
Output: SOM code vectors $\{\mathbf{m}_k\}$
Initialize SOM parameters $\{\mathbf{m}_k\}$;
for $n = 1, \ldots,$ *iterations* **do**
  $T \leftarrow T_{max} \left(\frac{T_{min}}{T_{max}}\right)^{n/iterations}$;
  Load next training sample $\mathbf{x}_i$;
  Compute BMU b_i ;
  for $k = 1, \ldots, K$ **do**
    Update prototype $\mathbf{m}_k$ (by equation 2) ;
  end
end

Algorithm 1: Stochastic SOM algorithm.

A disadvantage of this algorithm is that it converges slowly, is sequential and cannot be parallelized. Therefore, another algorithm was introduced: the batch SOM algorithm. It consists in minimizing following cost function, called *distortion*:

$$\mathcal{L}_{\text{SOM}}(\{\mathbf{m}_k\}, \mathbb{X}, b, T) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\mathcal{K}^T(\delta(b_i, k))||\mathbf{x}_i - \mathbf{m}_k||_2^2$$

Distortion is not directly differentiable because of the BMU assignments b. However, it can be empirically minimized using a dynamic clusters method [7] (similar to k-means) which alternates between two steps:

1. Assignment of best-matching units using Eq. 3
2. Minimization of distortion by fixing assignments, using following update rule:

$$\mathbf{m}_k \leftarrow \frac{\sum_{l=1}^{K}\mathcal{K}^T(\delta(k,l))\sum_{i=1}^{N}\mathbb{1}_{[b_i=l]}\mathbf{x}_i}{\sum_{l=1}^{K}\mathcal{K}^T(\delta(k,l))\sum_{i=1}^{N}\mathbb{1}_{[b_i=l]}} \tag{3}$$

The batch algorithm pseudo-code is detailed in algorithm 2.

2.2 Autoencoders

Autoencoders (AE) are neural networks trained to learn to reconstruct their inputs, in order to extract useful intermediate representations in an unsupervised way while minimizing information loss during this process [23]. These representations or features can then be used to improve downstream tasks such as clustering or supervised learning, that benefit from dimensionality reduction and higher-level features.

An autoencoder is composed of an encoder network, mapping the input to an intermediate latent space, and a decoder network, reconstructing a latent sample back into original space. In its simplest form, the model is trained to minimize information loss between input and reconstruction, taking the form of a reconstruction loss, often the mean squared error (MSE) for real-valued data. In most cases, the latent space is chosen to have a much smaller dimension than input space, called *undercomplete* autoencoders. This forces the network to compress data and not learn the identity function; thus, it is a form of regularization. This is the family of AE we will use in this work. However, *overcomplete* autoencoders are also useful, and they use different regularization criteria, see for example denoising AE [46] or sparse AE [2, 39]. The encoder and

Input: training set $\mathbb{X}$; SOM map size; temperatures T_{max}, T_{min}; *iterations*
Output: SOM code vectors $\{\mathbf{m}_k\}$
Initialize SOM parameters $\{\mathbf{m}_k\}$;
for $n = 1, \ldots,$ *iterations* **do**
  $T \leftarrow T_{max} \left(\frac{T_{min}}{T_{max}}\right)^{n/iterations}$;
  Compute all BMUs $\{b_i\}_{i=1\ldots N}$;
  for $k = 1, \ldots, K$ **do**
    Update prototype $\mathbf{m}_k$ (by equation 3) ;
  end
end

Algorithm 2: Batch SOM algorithm.

decoder can be deep nonlinear neural networks, able to learn richer representations than linear methods such as principal component analysis (PCA). The latent space can be either continuous or discrete, by using quantization in latent space (see [47] for a review on quantized autoencoders). Here, we will only consider continuous latents. Finally, a probabilistic extension is the variational autoencoder family (VAE) [26, 41]. VAE are generative models that enforce a probabilistic structure on latent space in order to sample from it, and improve the robustness of learned codes. While this is promising, our work will only use deterministic AE.

2.3 Representation learning for clustering

Reviews on deep clustering are available in [1] and [37]. To our knowledge, the first work of this kind was proposed in 2014 by authors of [42]. They propose to jointly learn representations with an autoencoder and clusters using *k*-means in latent space. The AE reconstruction loss is regularized by the *k*-means loss, and the training procedure alternates between updating the AE network and the cluster centers using *k*-means. Another early approach, deep embedded clustering (DEC) [49], jointly learns representations and soft cluster assignments by optimizing a Kullback–Leibler (KL) divergence that minimizes within-cluster distance, by pushing latent points together to form clusters; IDEC [20] improves on this approach by optimizing the reconstruction loss jointly with the KL-divergence. The deep clustering network (DCN) [50] combines representation learning with *k*-means clustering using stochastic gradient descent in an alternating training procedure, to alternately update the autoencoder weights, cluster assignments and centroid vectors, similarly to [42]. They introduce the term *k-means-friendly space* to describe the regularizing effect of joint training that improves clustering performance. More recently, [11] overcame the non-differentiability of hard cluster assignments by introducing a smoothed version of the *k*-means loss with simulated annealing. Deep clustering methods have been applied to image clustering in particular, leveraging convolutional network architectures [9, 21].

Most recent approaches perform latent space clustering using generative models such as variational autoencoders (VAE) [8, 24] or generative adversarial networks (GAN) [22, 38, 51] with a Gaussian mixture model (GMM) prior in latent space, achieving state-of-the-art results. While most approaches rely on (soft) *k*-means or GMM as the clustering component, the framework has been extended to mean-shift clustering [33].

2.4 Deep self-organizing maps

Although less attention has been given to self-organizing map models than pure clustering, this attention raised in 2018 and 2019, with several works on this subject.

In [10], authors have shown evidence that using higher-level features extracted by convolutional layers of pretrained networks improves SOM quality, as measured by several external label-oriented indices but also visual quality. However, we think these results are somewhat biased because the features are extracted from a network that was pretrained in a supervised manner on the same dataset. Thus, the top layers were trained to learn a feature space where the classes are well-separated. Of course, this will help the subsequent SOM to separate the classes and improve the label-oriented results. Maybe the performance improvements would have been lower using features from a network that was pretrained on another generic dataset, e.g., ImageNet. Therefore, although the idea is to use a generic pretrained network for feature extraction in order to have a completely unsupervised process, the experiments in the paper differ from this intent.

Other works, including ours, are based on unsupervised neural networks, namely autoencoders. An early approach, deep neural maps [40], combines a denoising AE, the soft clustering loss from [49] and a SOM. Their procedure alternates between (1) assigning and updating cluster centers using the SOM stochastic algorithm (2) optimize the representations, regularized by the KL-divergence clustering term. They compared this approach with other dimensionality reduction and visualization tools; however, they did not quantitatively assess the clustering or self-organization performance. [13] proposes the denoising autoencoder self-organizing map (DASOM) and several variants in an extensive study on the combination of a nonlinear denoising autoencoder (DAE) with SOM. The Deep Embedded SOM (DESOM) model, subject of this paper, was introduced in [16, 17]. Another series of work performing joint representation learning with a SOM is the SOM-VAE model, introduced in [19]. Their model is based on the VQ-VAE (Vector Quantization Variational Autoencoder) model which enables to train variational autoencoders (VAEs) with a discrete latent space [44]. [19] have added a topology constraint on the discrete latent space by modifying the loss function of VQ-VAE. Their loss function is composed of three terms: the VAE evidence lower bound (ELBO) reconstruction loss, the vector quantization loss (VQ) and the SOM loss. However, there are many important differences between our DESOM model and SOM-VAE. First, SOM-VAE utilizes a discrete latent space to represent the SOM prototypes, whereas in DESOM, the SOM is learned in a continuous latent space.

Table 1 Comparison of the properties of deep SOM models

Model	Latent	AE	Rec. loss	SOM loss	Joint	Pretraining
ConvSOM [10]	Continuous	AE	MSE	SOM	✗	✓
DASOM [13]	Continuous	DAE	MSE	SOM	✓	✓
DESOM (ours)	Continuous	AE	MSE	SOM	✓	✗
Deep neural maps [40]	Continuous	AE	MSE	KL+SOM	✓	✓
SOM-VAE [19]	Discrete	VQ-VAE	ELBO	VQ+SOM	✓	✓
DPSOM [34]	Continuous	VAE	ELBO	KL+SOM	✓	✓

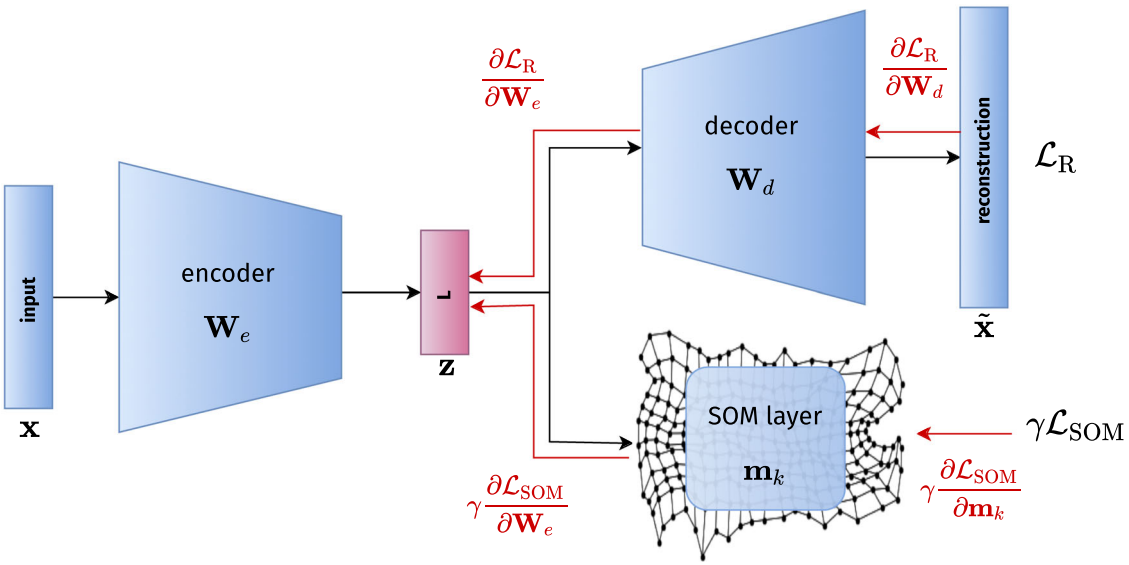

Fig. 1 DESOM architecture and gradients paths. The input $\mathbf{x}$ is projected into latent space by the encoder $\mathbf{W_e}$, where the SOM prototypes $\mathbf{m}_k$ are learned via the SOM loss $\mathcal{L}_\mathrm{SOM}$. Latent samples $\mathbf{z}$ are reconstructed using the decoder $\mathbf{W_d}$ via the reconstruction loss $\mathcal{L}_\mathrm{R}$

Secondly, they use a fixed window neighborhood to update the map prototypes, whereas we use a Gaussian neighborhood with exponential radius decay. Finally, the DESOM model presented in this work is based on a deterministic autoencoder and not a VAE. The Deep Probabilistic SOM (DPSOM) [34], a very recent unpublished work improving on the SOM-VAE, achieves state-of-the-art clustering results and proposes an interpretable application in the public health domain. Their loss function combines the ELBO (without discrete quantization), a KL-divergence clustering loss and a SOM loss with fixed neighborhood. Table 1 summarizes the properties of the different deep SOM models.

3 Deep embedded self-organizing map

3.1 Architecture

This work is based on the DESOM model introduced in [16, 17]. Therefore, we will first present the model architecture, its loss function, hyperparameters and training procedure. We propose an approach where self-organization of the prototypes and representation learning through a deterministic autoencoder are performed jointly by stochastic gradient descent (SGD). The architecture is illustrated in Fig. 1. The encoder and decoder networks are generic and can be fully connected, convolutional or even recurrent [16]. In this work, we experiment with a fully connected (DESOM) and a convolutional (ConvDESOM) version.

The architecture is composed of three neural network modules: an encoder, a decoder and a SOM layer. The encoder projects the inputs onto a latent, intermediate space. The SOM is trained in this latent space and receives encoded inputs. The decoder reconstructs the latent code back into original space, trying to match the input as closely as possible.

3.2 Loss function

The encoder and decoder parameter weights are, respectively, noted $\mathbf{W_e}$ and $\mathbf{W_d}$. The encoding function is denoted by $\mathbf{f}_{\mathbf{W_e}}$ and the decoding function by $\mathbf{g}_{\mathbf{W_d}}$. Thus, $\mathbf{z}_i = \mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i) \in \mathbb{R}^L$ is the embedded version of $\mathbf{x}_i$ in the intermediate latent space, and $\tilde{\mathbf{x}}_i = \mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i)) \in \mathbb{R}^D$ is its reconstruction by the decoder. Our goal is to jointly optimize the autoencoder network weights and the SOM prototype vectors. For this task, we define a hybrid loss function composed of two terms, that can be written as:

$$\mathcal{L}(\mathbf{W_e}, \mathbf{W_d}, \mathbf{m}_1, \ldots, \mathbf{m}_K) = \mathcal{L}_{\mathrm{R}}(\mathbf{W_e}, \mathbf{W_d}) + \gamma \mathcal{L}_{\mathrm{SOM}}(\mathbf{W_e}, \mathbf{m}_1, \ldots, \mathbf{m}_K) \tag{4}$$

The first term $\mathcal{L}_R$ is the autoencoder reconstruction loss. We use a mean squared error loss, which corresponds to reconstructing the mean of a Gaussian output distribution:

$$\mathcal{L}_{\mathrm{R}} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{\mathrm{R}}^i = \frac{1}{N}\sum_i ||\tilde{\mathbf{x}}_i - \mathbf{x}_i||_2^2$$

The second term is the self-organizing map loss, denoted $\mathcal{L}_{\mathrm{SOM}}$. It depends on the set of parameters $\{\mathbf{m}_k\}_{1 \le k \le K}$ and on the best-matching units, denoted b_i, assigning a latent data point to its closest prototype according to Euclidean distance, i.e.:

$$b_i = \underset{k}{\operatorname{argmin}} ||\mathbf{z}_i - \mathbf{m}_k||_2^2$$

The expression of the self-organizing map loss is:

$$\mathcal{L}_{\mathrm{SOM}} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{\mathrm{SOM}}^i = \frac{1}{N}\sum_i \sum_{k=1}^{K} \mathcal{K}^T(\delta(b_i, k)) ||\mathbf{z}_i - \mathbf{m}_k||_2^2$$

The gamma γ coefficient is a hyperparameter that trades off between minimizing the autoencoder reconstruction error and the self-organizing map error. Therefore, the SOM loss acts as a SOM-guided regularizer.

3.3 Topological organization interpretation

The SOM loss can be decomposed into two terms, the first one being the squared distance between the best matching unit and the latent point, and the second one corresponding to the topological relationship with neighboring units.

$$\begin{aligned}
\mathcal{L}_{\mathrm{SOM}} &= \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} \mathcal{K}^T(\delta(b_i,k)) ||\mathbf{z}_i - \mathbf{m}_k||_2^2 \\
&= \frac{1}{N}\sum_{i=1}^{N} \left[\mathcal{K}^T(\delta(b_i,b_i)) ||\mathbf{z}_i - \mathbf{m}_{b_i}||^2 + \sum_{k \neq b_i} \mathcal{K}^T(\delta(b_i,k)) ||_2 \mathbf{z}_i - \mathbf{m}_k||_2^2 \right] \\
&= \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{z}_i - \mathbf{m}_{b_i}||_2^2 + \frac{1}{N}\sum_{i=1}^{N}\sum_{k \neq b_i} \mathcal{K}^T(\delta(b_i,k)) ||\mathbf{z}_i - \mathbf{m}_k||_2^2
\end{aligned}$$

For large values of T, the second term is prevalent and leads to topological organization. When the temperature approaches zero, the first term prevails and the SOM loss becomes identical to a k-means loss, where the centroids correspond to the map prototypes:

$$\lim_{T \to 0} \mathcal{L}_{\mathrm{SOM}} = \frac{1}{N}\sum_i ||\mathbf{z}_i - \mathbf{m}_{b_i}||_2^2 = \mathcal{L}_{k-\mathrm{means}}$$

Thus, when temperature is close to zero, the hybrid loss function 4 can be written as follows:

$$\lim_{T \to 0} \mathcal{L} = \mathcal{L}_{\mathrm{R}} + \gamma \mathcal{L}_{k-\mathrm{means}}$$

Hence, our model becomes identical to the DCN model [50] or the DKM model [11] (at the end of their hyperparameter annealing).

3.4 Training procedure

We use a joint training procedure, optimizing both the network parameters and the prototypes by backpropagation and stochastic gradient descent. The assignments to the best-matching units are fixed between each optimization step, as it is non-differentiable. Thus, the weighting terms $w_{i,k} \equiv \mathcal{K}^T(\delta(b_i,k))$ become simple coefficients for each data point and prototype, constant with respect to the network parameters and the prototypes. The gradients of the loss function w.r.t. autoencoder weights and prototypes are easy to derive if we consider the assignments to be fixed at each step.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{W_e}} &= \frac{\partial \mathcal{L}_{\mathrm{R}}}{\partial \mathbf{W_e}} + \gamma \frac{\partial \mathcal{L}_{\mathrm{SOM}}}{\partial \mathbf{W_e}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W_d}} &= \frac{\partial \mathcal{L}_{\mathrm{R}}}{\partial \mathbf{W_d}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{m}_k} &= \gamma \frac{\partial \mathcal{L}_{\mathrm{SOM}}}{\partial \mathbf{m}_k}
\end{aligned}$$

The gradients for a single data point $\mathbf{x}_i$ are:

$$\frac{\partial \mathcal{L}_{\mathrm{R}}^{i}}{\partial \mathbf{W_e}} = 2(\mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i)) - \mathbf{x}_i)\frac{\partial \mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i))}{\partial \mathbf{W_e}}$$

$$\frac{\partial \mathcal{L}_{\mathrm{R}}^{i}}{\partial \mathbf{W_d}} = 2(\mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i)) - \mathbf{x}_i)\frac{\partial \mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i))}{\partial \mathbf{W_d}}$$

$$\frac{\partial \mathcal{L}_{\mathrm{SOM}}^{i}}{\partial \mathbf{W_e}} = 2\sum_{k=1}^{K} w_{i,k}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i) - \mathbf{m}_k)\frac{\partial \mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i)}{\partial \mathbf{W_e}}$$

$$\frac{\partial \mathcal{L}_{\mathrm{SOM}}^{i}}{\partial \mathbf{m}_k} = 2w_{i,k}(\mathbf{m}_k - \mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i))$$

The paths of the gradients of the loss function are illustrated on Fig. 1. We optimize Eq. 4 using minibatch stochastic gradient descent (SGD), with a learning rate l_r (in our experiments Adam is used instead, but the equations are derived for vanilla SGD). Minibatch SGD is a standard technique used to train autoencoders and deep clustering models [1, 19, 46], and as we optimize all parameters jointly, the SOM distortion loss is optimized in the same manner. Using batch gradient descent would not be practical for large data sets and result in slow convergence. On the other side, using a batch size of 1 (as in the stochastic Kohonen algorithm) would also be inefficient to train the AE and in particular would not benefit from parallelized implementations. Given a batch $\mathcal{B}$ of n_b samples, the encoder's weights are updated by:

$$\mathbf{W_e} \leftarrow \mathbf{W_e} - \frac{l_r}{n_b}\sum_{i\in\mathcal{B}}\left(\frac{\partial \mathcal{L}_{\mathrm{R}}^{i}}{\partial \mathbf{W_e}} + \gamma\frac{\partial \mathcal{L}_{\mathrm{SOM}}^{i}}{\partial \mathbf{W_e}}\right) \tag{5}$$

The decoder's weights are updated by:

$$\mathbf{W_d} \leftarrow \mathbf{W_d} - \frac{l_r}{n_b}\sum_{i\in\mathcal{B}}\frac{\partial \mathcal{L}_{\mathrm{R}}^{i}}{\partial \mathbf{W_d}} \tag{6}$$

And finally, the map prototypes are updated by the following update rule:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k - \frac{l_r}{n_b}\sum_{i\in\mathcal{B}}\gamma\frac{\partial \mathcal{L}_{\mathrm{SOM}}^{i}}{\partial \mathbf{m}_k} \tag{7}$$

By expanding the prototypes update rule 7, we obtain an expression somewhat in between of the stochastic SOM and the batch SOM algorithms presented in the previous section (see Eqs. 2 and 3), that we can call *minibatch SOM*:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k + 2\gamma\frac{l_r}{n_b}\sum_{i\in\mathcal{B}}\mathcal{K}^{T}(\delta(b_i,k))(\mathbf{z}_i - \mathbf{m}_k) \tag{8}$$

As in batch SOM, we alternate between BMU assignments and minimization, but minimization happens via a gradient descent step as in stochastic SOM. Thus, we think optimizing SOM with our procedure is a sound choice.

3.5 Implementation

The code for the DESOM model is available as an open-source project[3]. It was implemented in the Keras framework. The main novelty of our model is a new custom layer called *SOM layer*. Its parameters are the set of SOM code vectors in latent space, i.e., a $K \times L$ matrix where K is the number of prototypes (e.g., 64 for an 8×8 map) and L is the dimensionality of the latent space. The outputs of this layer are defined as the pairwise squared Euclidean distances between the input batch and the prototypes: this allows to express the SOM loss as a simple weighted sum, using the neighborhood weight terms $w_{i,k}$.

First, the autoencoder is initialized either randomly using the Glorot uniform initializer, or by pretraining, and SOM parameters are initialized either at random with an encoded data sample, or by a standard SOM (this choice is studied in the experiments). At each iteration, the temperature is updated using exponential decay. Then, we perform inference on the current batch to obtain the pairwise distances between latent samples and SOM prototypes, in order to compute the weights $w_{i,k}$ using the neighborhood function. Finally, we perform a training step to update all parameters. In addition, we collect losses and performance metrics at a fixed interval on the training and test sets, using the library SOMperf[4] [18].

[3] https://github.com/FlorentF9/DESOM.

[4] https://github.com/FlorentF9/SOMperf.

Input: training set $\mathbb{X}$; AE architecture; SOM map size; temperatures T_{max}, T_{min}; *iterations*; *batchSize*
Output: AE weights $\mathbf{W_e}$, $\mathbf{W_d}$; SOM code vectors $\{\mathbf{m}_k\}$
Initialize AE weights $\mathbf{W_e}$, $\mathbf{W_d}$ (random or pretrain) ;
Initialize SOM parameters $\{\mathbf{m}_k\}$ (random data sample or pretrain) ;
for $n = 1, \ldots,$ *iterations* **do**
  $T \leftarrow T_{max} \left(\frac{T_{min}}{T_{max}}\right)^{n/iterations}$;
  Load next training batch $\mathcal{B}$;
  Encode current batch ;
  Compute assignments and weights $w_{i,k}$ on batch ;
  Train DESOM on batch by taking a SGD step (by equations 5, 6 and 7) ;
end

Algorithm 3: DESOM training procedure.

The training procedure of DESOM is detailed in algorithm 3, omitting the test set for sake of brevity. Differently from the original version of DESOM [17], we also try updating the self-organizing map not at every training iteration, but only every `update_interval` iterations, introducing an additional hyperparameter. This follows remarks from [20] and [32] and should help better training the encoder. To achieve this, we set all gradients coming from the SOM loss to zero between each update interval. The impact of this update interval will be mentioned in the next section. Finally, the number of iterations is set to train until convergence of the loss for every data set.

3.6 Training parameters

Across all experiments, we use rectangular SOM topologies with δ being the Manhattan distance between units (each unit has four direct neighbors, excepted on the map borders). For optimization, we use Adam with $l_r = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. The T_{min} parameter defines the final neighborhood radius at the end of training. It has a direct impact on the trade-off between quantization and topographic error: by choosing a T_{min} smaller than 1, the prototype vectors will be finetuned locally and improve quantization; however, it may harm the topology of the map. This choice depends on the priority of the practitioner. In our experiments, we set T_{min} to 0.1, in order to obtain good quantization and clustering, and we observed no visual degradation of the map. This will be further discussed. T_{max} is always set equal to the map size, in order to organize all units in the early stage of training. Similarly, the trade-off between quantization and topology also depends on the number of training iterations. This fact is illustrated by the learning curves (Fig. 6) and is discussed thereafter. In this work, we choose to not use early stopping, and to train until full convergence. All other parameters such as map size, latent space dimension, gamma γ, pretraining, initialization and batch sizes are studied in the experiments. As there is a large number of parameters, we recapitulate them in Table 2. The last column indicates whether the value of the parameter is modified and studied in the experiments section; otherwise, it is fixed to the value indicated in the default value column.

Table 2 DESOM training parameters

Parameter	Notation	Default value	Studied in experiments
Gamma	γ	10^{-3}	✔
Latent code dimension	L	10	✔
Map grid topology	–	rectangular	✗
Distance between units	$\delta(\cdot,\cdot)$	Manhattan	✗
Map size	–	8×8	✔
Initial temperature	$T_{\max}$	8.0	✔
Final temperature	$T_{\min}$	0.1	✔
Iterations	–	convergence	✗
Batch size	n_b	256	✔
Learning rate	l_r	0.001	✗
First moment decay	β_1	0.9	✗
Second moment decay	β_2	0.999	✗

Table 3 Dataset statistics

Dataset	Description	Total	Train	Test	Classes	Dimension
MNIST	Images (digits)	70,000	60,000	10,000	10	784
Fashion-MNIST	Images (clothing)	70,000	60,000	10,000	10	784
USPS	Images (digits)	9298	7291	2007	10	256
REUTERS-10k	Text (tf-idf)	10,000	7769	2231	4	2000

4 Architecture and hyperparameter study

In previous works [16, 17], experiments and benchmarks used a single architecture and only two hyperparameters were studied: the weighting between reconstruction loss and SOM loss (gamma γ), and the use of autoencoder pretraining. However, the DESOM model is governed by other hyperparameters that are coupled in the training process. After introducing the datasets used in our experiments in the first paragraph of this section, a thorough exploration of architecture and training hyperparameters and their influence on performance metrics is the goal of the experiments detailed in the second paragraph. The subsequent paragraph discusses initialization strategies for the AE and SOM weights. In the third paragraph, we study the learning dynamics of DESOM, in particular the evolution of learning curves and the interaction between its two components (AE and SOM), with different parameters.1 Then, the next paragraph studies the visual quality of prototypes for image datasets. The performance metrics used here are all described in Appendix . All experiments are run 10 times to obtain meaningful means and standard deviations, displayed on the graphs. We insist on the fact that unlike our previous work, where training and test sets were concatenated (as is often done in unsupervised learning), here we use the standard train/test split (see Table 3) and always report results on the test set (unless specified otherwise).

4.1 Datasets

We experiment, evaluate and compare models on four different classification benchmark datasets, three image datasets and one text dataset:

- **MNIST** [29]: the MNIST dataset consists in 70000 grayscale images of handwritten digits, of size 28-by-28 pixels. We used the dataset available in the Keras library, divided the pixel intensities by 255 to obtain floating-point values between 0 and 1, and flattened the images to 784-dimensional vectors (except for the convolutional architecture).
- **Fashion-MNIST** [48]: the Fashion-MNIST dataset was designed as a drop-in replacement for the original MNIST dataset, but with images of clothing instead of digits, and provides a more challenging classification task. The dataset is also available in Keras, and we applied the same preprocessing.
- **USPS**: this dataset also consists in images of grayscale handwritten digits and contains 9298 16-by-16 pixel digits. We downloaded it from the Kaggle website and did not perform any preprocessing.
- **REUTERS-10k** [30]: the REUTERS-10k dataset is built from the RCV1-v2 corpus, that contains 804,414 English news stories labeled with a category tree, with a total of 103 topics. REUTERS-10k is created by restricting the documents to 4 root categories (corporate/industrial, government/social, markets and economics), excluding documents with multiple labels, then sampling a subset of 10000 examples and computing TF-IDF features on the 2000 most

Table 4 Comparison of external clustering metrics (purity and NMI) with different values of hyperparameter gamma. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t test)

γ	MNIST		Fashion-MNIST	
	Pur	NMI	Pur	NMI
10^{-4}	.929 ± .004	.652 ± .003	**.759 ± .006**	**.543 ± .004**
10^{-3}	**.934 ± .004**	**.658 ± .004**	.751 ± .009	**.541 ± .004**
10^{-2}	.911 ± .006	.639 ± .006	.737 ± .007	.529 ± .005
0.1	.876 ± .008	.609 ± .006	.721 ± .005	.520 ± .004
0.5	.836 ± .033	.584 ± .016	.718 ± .009	.517 ± .006
1.0	.810 ± .025	.566 ± .019	.715 ± .012	.517 ± .007
10.0	.114 ± .000	.006 ± .003	.678 ± .008	.484 ± .007
γ	USPS		Reuters-10k	
	Pur	NMI	Pur	NMI
10^{-4}	.837 ± .014	.573 ± .009	.795 ± .019	.352 ± .012
10^{-3}	**.857 ± .011**	**.592 ± .010**	**.808 ± .017**	**.364 ± .011**
10^{-2}	.839 ± .013	.583 ± .008	.801 ± .017	.352 ± .014
0.1	.815 ± .014	.566 ± .007	**.809 ± .014**	**.365 ± .016**
0.5	.806 ± .018	.561 ± .012	**.819 ± .020**	**.371 ± .012**
1.0	.806 ± .013	.559 ± .006	**.804 ± .030**	**.354 ± .027**
10.0	.760 ± .029	.523 ± .021	.466 ± .101	.062 ± .078

frequently occurring words. We downloaded the raw RCV1-v2 topics and tokens and used the same code as in [20] to build the dataset.

The properties of each dataset are described in Table 3. In particular, we use the default train/test splits. These datasets were selected because they all have a high dimensionality (256 to 2000) and can benefit greatly from the representation learning through a deep neural network. Using Euclidean distance directly on a high-dimensional space, as is done in traditional SOM, is known to be problematic, as explained in Sect. 1. Image datasets also have the advantage of being easily visualized on a self-organizing map. Finally, these datasets were used in many previous works on deep learning-based clustering models, allowing direct comparisons.

In the following paragraphs, we study the influence of five fundamental parameters: the gamma γ hyperparameter, the latent code dimension, the map size and the nature of the autoencoder (fully connected or convolutional). All other parameters that are either related to initialization or to training dynamics are fixed to reasonable values and will be studied later.

4.1.1 Gamma γ parameter study

This parameter defines the relative weight of reconstruction and SOM in the loss function. We evaluate external clustering metrics on four datasets for different values of gamma, ranging from 10^{-4} to 10 (see Table 4). Our goal is not to cross-validate and find the best value according to some external quality metric, as we are in an unsupervised setting, but to find a reasonable order of magnitude across multiple datasets.

The gamma hyperparameter is a trade-off between preserving information (obtaining good reconstructions) and SOM clustering. As the SOM loss takes larger values than the reconstruction loss, gamma must be set smaller than one. A good value is $\gamma = 10^{-3}$ across all datasets; it represents the optimum for MNIST, USPS, and is within the variance interval for Fashion-MNIST and Reuters-10k (for the latter dataset, variance of our AE is very high). Higher values of gamma lead to degenerate solutions for the encoder and decoder, which translates into low scores across all datasets, and the autoencoder being unable to produce any reconstructions. This is due to the fact that the model tries hard to optimize the SOM loss, which is much easier to optimize than the reconstruction loss as we observed during our experiments, thus neglecting code quality.

Figure 2 represents SOM quantization and topographic error as a function of gamma (for MNIST). Quantization error exhibits the trade-off between the reconstruction and SOM clustering: it decreases with gamma, as high gamma values result in the SOM being more finetuned. As a drawback, this finetuning also increases topographic error, as can been seen on the rightmost graph. Behavior is similar for other datasets.

As a conclusion, DESOM is not very sensitive to gamma as long as it stays in the right order of magnitude, and we select $\gamma = 10^{-3}$ for the rest of the paper, in accordance with the unsupervised setting (even if better results could be obtained by choosing an adapted value for each dataset).

4.1.2 Latent code dimension study

Authors in [34] have found that DEC performed better with a lower-dimensional AE latent space ($L = 10$), while their VAE performed better with a higher code dimension ($L = 100$).

Let us assume that the size of the self-organizing map has been fixed by the user. The dimensionality of the latent space where this SOM will be learned is expected to be a determining parameter. Concretely, we expect following behavior: on the one hand, a latent space too small will result in a loss of information and a lower performance as measured by external indices (label-based). However, the SOM may fit the latent code space very well and produce high latent quality metrics (e.g., latent quantization error). On the other hand, with a large latent space, the autoencoder will not use all latent variables to extract useful features and likely overfit the training set, and in addition, the low-dimensional SOM will have difficulties to fit this high-dimensional space (translating into low latent quality metrics). A straightforward experiment consists in comparing performance metrics with different latent space dimensions, leaving all other parameters unchanged.

These intuitions are confirmed by Table 5, showing that a latent space dimension too low or too high both hurt the model's performance. An optimal value exists: it is $L = 10$ for MNIST, Fashion-MNIST and USPS, but not for Reuters-10k, suggesting that the optimal value depends on the intrinsic dimensionality of the latent factors of variation in the data distribution. Latent quantization error, represented as a function of L on Fig. 3, naturally varies like $\sqrt{L}$.

When not mentioned, we use a latent dimension equal to 10, even if better results could be obtained by tuning the dimension for each dataset (to remain in an unsupervised setting).

4.1.3 Map size study

This paragraph studies the influence of the SOM map size (number of neurons) on the results. The shape is kept

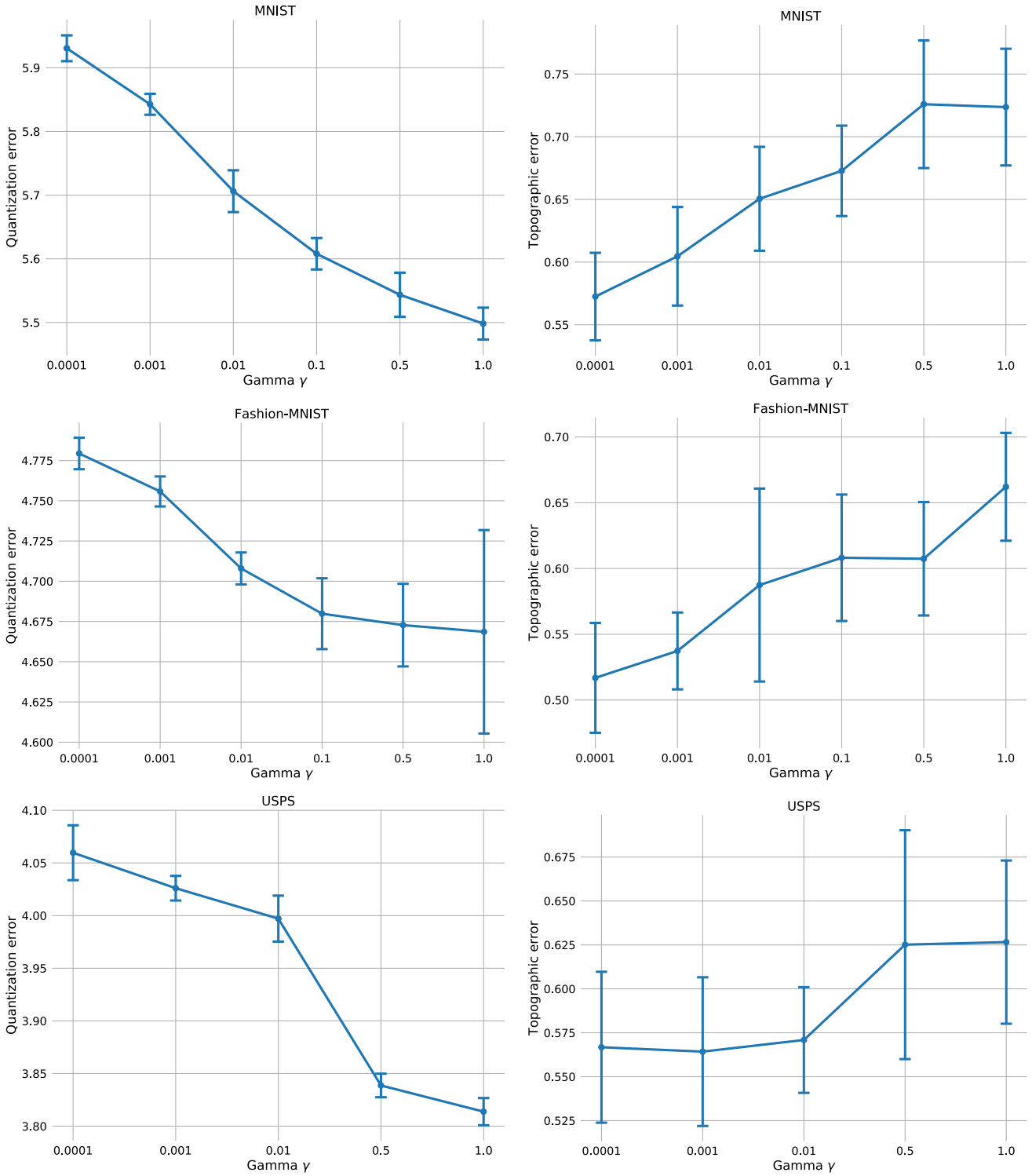

Fig. 2 Quantization and topographic error as a function of gamma on MNIST, Fashion-MNIST and USPS

square, with equal width and height, varying between 5 and 20 units.

Metrics behave as expected, i.e., clustering purity naturally increases with map size, as there are more clusters, but NMI decreases (see Appendix 1 for definitions of clustering performance metrics). Quantization error improves while topographic error increases; combined error shows a minimum around size 8×8 on MNIST and Fashion-MNIST, which corresponds to the size we use throughout the paper, to compare with previous works (see

Table 5 Comparison of purity and NMI with different latent code dimensions (L). Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t test)

L	MNIST		Fashion-MNIST	
	Pur	NMI	Pur	NMI
2	.768 ± .012	.552 ± .011	.688 ± .023	.499 ± .010
5	.901 ± .010	.628 ± .007	.736 ± .011	.530 ± .005
10	**.931 ± .007**	**.654 ± .006**	**.756 ± .008**	**.542 ± .003**
20	**.925 ± .006**	.647 ± .006	**.752 ± .008**	**.542 ± .003**
50	.921 ± .006	.641 ± .005	.747 ± .009	**.541 ± .004**
100	.921 ± .004	.643 ± .004	.747 ± .006	**.541 ± .003**
L	USPS		Reuters-10k	
	Pur	NMI	Pur	NMI
2	.800 ± .013	.563 ± .007	**.840 ± .010**	**.375 ± .012**
5	.843 ± .004	.583 ± .006	.808 ± .016	.360 ± .012
10	**.855 ± .010**	**.591 ± .008**	.800 ± .024	.358 ± .017
20	.828 ± .010	.571 ± .007	.794 ± .028	.353 ± .022
50	.817 ± .012	.562 ± .011	.789 ± .030	.346 ± .017
100	.806 ± .021	.558 ± .012	.787 ± .019	.344 ± .015

Fig. 4). On USPS and Reuters-10k, the underlying manifold seems to require fewer units. SOM entropy and class scatter index (the number of label groups formed on the map) vary proportionally to the number of units.

4.1.4 Convolutional architecture study

The fact that CNN-based deep clustering models tend to outperform similar approaches using dense autoencoders can be seen in [1]. We compare the standard DESOM with a [500, 500, 2000, 10] fully connected encoder with a convolutional version, ConvDESOM [16]. The ConvDESOM autoencoder architecture is similar to the one used in [34]: 4 convolutional layers with [32, 64, 128, 256] filters of size 3×3, and 2×2 max pooling after each convolution. We use no batch normalization and activations are basic ReLUs. We also apply it to MNIST and Fashion-MNIST datasets.

From the comparison in Table 6, the convolutional architecture is superior in terms of clustering purity and NMI, but slightly hurts quantization and topographic errors. On most other metrics we compared, ConvDESOM is equivalent to DESOM. As a conclusion, a convolutional AE performs better on images, but makes little difference on toy datasets, as the fully connected version is able to learn sufficient representations, but we believe that for larger, more complex and high-dimensional data, ConvDESOM or other architectures should produce superior results.

4.2 Initialization and pretraining

We now study the influence of initialization and pretraining. Usually, we seek a good initial solution for the model parameters and avoid local minima. In our model, two components must be initialized: AE and SOM.

AE pretraining Pretraining the autoencoder consists in training it with only the reconstruction loss before performing the joint task. There are several ways to pretrain an autoencoder: traditional end-to-end training, stacked denoising autoencoders [46] (also called layer-wise pretraining), RBM pretraining [23], etc. End-to-end training can be problematic because it could learn the identity function (but not an issue in case of undercomplete autoencoders), is less robust and prone to overfitting. Pretraining is used in most deep clustering approaches, either layer-wise [24, 49, 50], RBM [42] or end-to-end [11], and improves results. We compared the two following strategies for DESOM:

- No pretraining.
- Pretraining the AE in a simple end-to-end fashion for 100 epochs using MSE reconstruction loss.

SOM initialization The SOM weights are initialized using one of the following strategies:

- Random initialization: SOM weights are initialized with a random sample of encoded samples (taken without replacement).

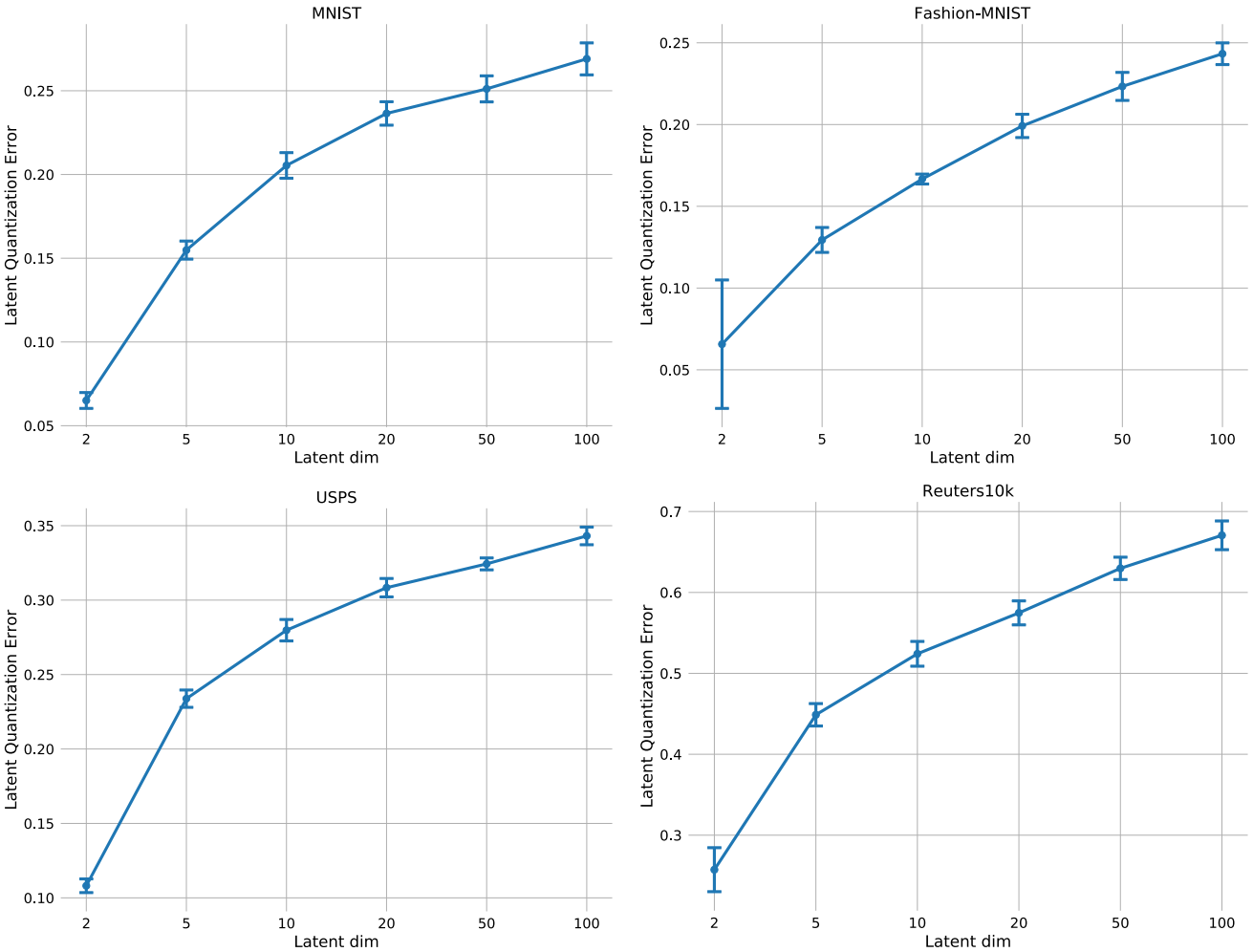

Fig. 3 Latent quantization error as a function of latent space dimension

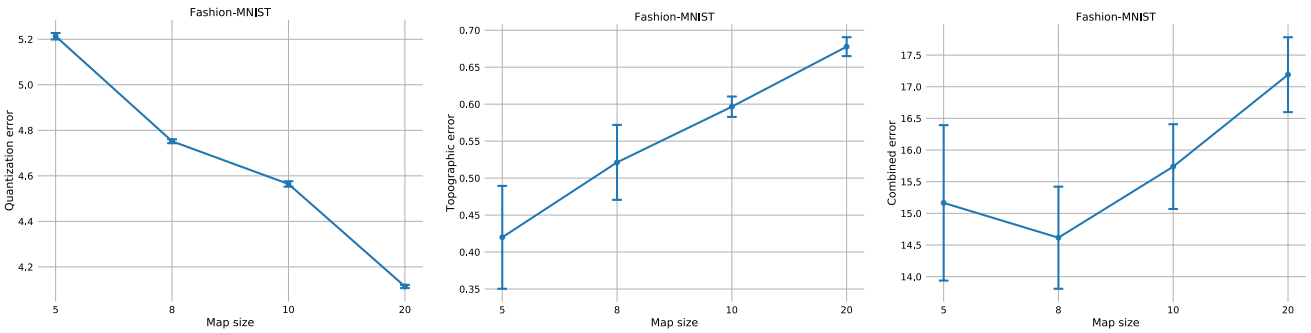

Fig. 4 Quantization, topographic and combined errors as a function of the map size (5×5, 8×8, 10×10 and 20×20) for Fashion-MNIST. Quantization improves with map size, while topographic error increases. Combined error acts as a trade-off and indicates 8×8 as a good compromise

- SOM initialization: a standard SOM is trained for 10 epochs on the encoded dataset (we used the minisom[5] package).

Other more sophisticated initialization schemes do exist (using for instance principal components), but we limit ourselves to these two simple strategies.

Initial temperature In the case when the AE is pretrained and SOM is initialized, we may not want to disturb the map topology and only finetune the prototypes and

[5] https://github.com/JustGlowing/minisom.

Table 6 Comparison between DESOM and ConvDESOM in terms of purity, NMI, quantization and topographic errors, for MNIST and Fashion-MNIST

Method	MNIST			
	Pur	NMI	QE	TE
DESOM	.934 ± .004	.658 ± .004	**5.843 ± 0.016**	**0.605 ± 0.039**
ConvDESOM	**.948 ± .004**	**.673 ± .005**	5.980 ± 0.038	**0.610 ± 0.034**
Method	Fashion-MNIST			
	Pur	NMI	QE	TE
DESOM	.751 ± .009	.541 ± .004	**4.756 ± 0.009**	**0.537 ± 0.029**
ConvDESOM	**.758 ± .004**	**.546 ± .002**	4.777 ± 0.017	**0.538 ± 0.045**

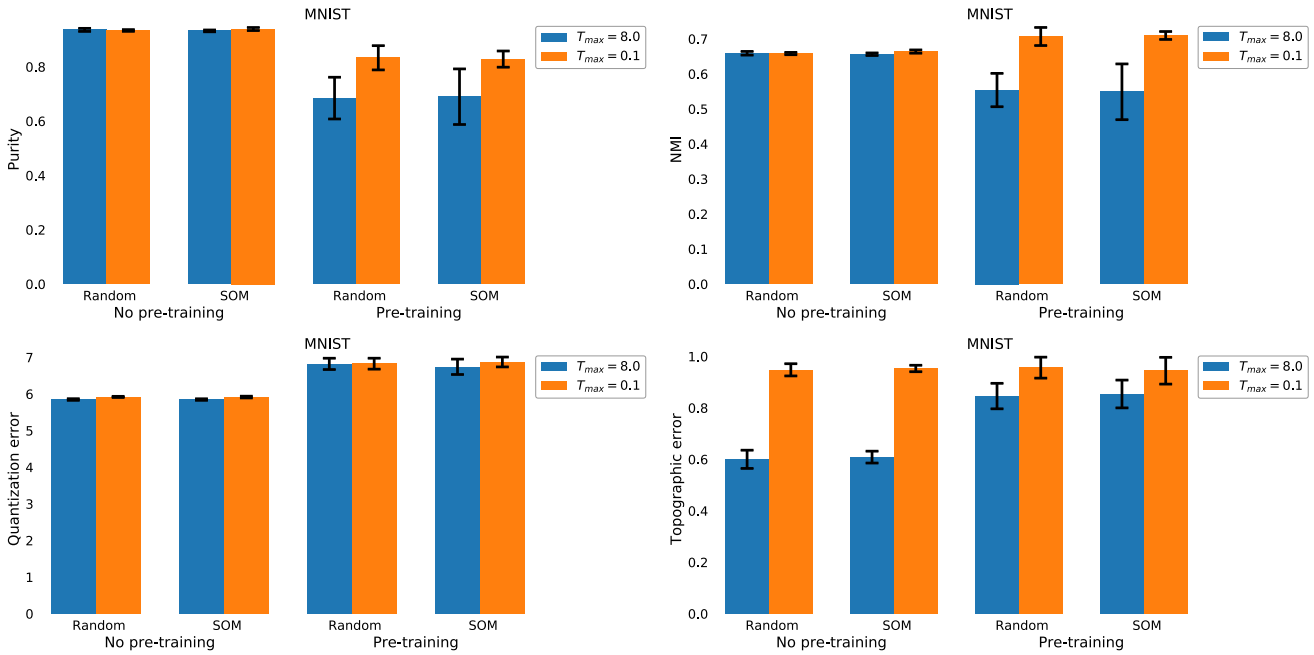

Fig. 5 Performance metrics (purity, NMI, quantization and topographic errors) with different combinations of pretraining (with or without) and initialization (random or SOM)

representations. Thus, we might try to use a very small initial temperature (e.g., $T_{max} = 0.1$). We try following initial temperatures:

- $T_{max} = 0.1$ (local finetuning)
- $T_{max} = 8.0$ (self-organization across the entire map size)

To summarize, this results in eight combinations of possible initializations. Some results are displayed in Fig. 5. As for all other experiments, we performed 10 runs for meaningful means and standard deviations. The first observation is that SOM initialization has no effect at all on the final results in terms of clustering (purity and NMI) or SOM quality (quantization and topographic errors). Second, AE pretraining deteriorates the model's performance. The only improvement is the NMI when $T_{max} = 0.1$, but a small initial temperature naturally leads to a higher topographic error, as the prototypes are locally finetuned and global topology is not preserved well. It suggests that the best solution is found when reconstruction and SOM loss are optimized jointly from the beginning. The lowest topographic error is achieved by random AE and SOM initialization with $T_{max} = 8.0$, which is the setting we use throughout all other experiments. To conclude, initialization and pretraining do not lead to performance improvements; in our benchmarks, none will be used. This allows to cut training time drastically, as AE pretraining time has the same order of magnitude DESOM full joint training (about the half, see Table 12 in the last paragraph).

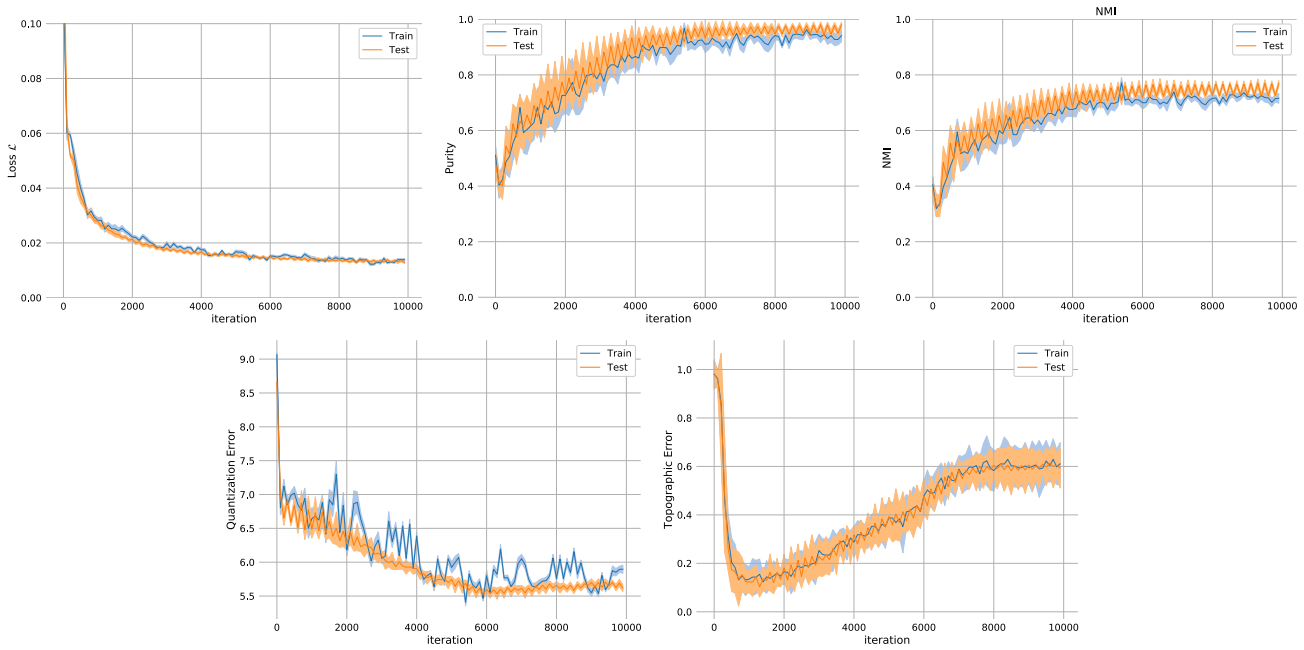

Fig. 6 Learning curves for loss, purity, NMI, quantization and topographic error for MNIST

4.3 Training parameters and learning dynamics

4.3.1 Learning curves

Learning curves representing the evolution of losses and metrics during training on the training and test set (see Fig. 6) show that our model converges, and does not overfit. To preserve space, curves are only included for the MNIST dataset, and for $\gamma = 10^{-3}$, unless specified. An interesting behavior is that of topographic error, which first rapidly decreases, but then increases back until convergence (when temperature T reaches T_{min}). It shows the trade-off between self-organization and the autoencoder's reconstruction quality (very low at the beginning) as well as the resulting clustering quality. A practitioner could thus choose to use an early stopping strategy, to obtain a lower topographic error, but it would harm the quality of representations and clustering. For this reason, we choose to train until convergence in this work.

4.3.2 Latent space evolution

We visualize the evolution of latent space during training using the UMAP (Uniform Manifold Approximation and Projection) dimensionality reduction technique [35]. We choose this method instead of the widely used t-SNE projection, because it runs orders of magnitude faster (the MNIST test set with 10000 points is projected in a few seconds only, compared with several minutes for t-SNE), and it is also able to effectively visualize the local and global structures of the data distribution. Figure 7 displays the MNIST test set and DESOM map after 0, 10, 20 and 40 epochs. Points are colored according to their target class (digit). We clearly see that the DESOM objective (with SOM regularization) pushes points together to form clusters, and that map prototypes are self-organized in latent space. Note that we cannot interpret the SOM grid topology because of the UMAP projection, but we can still note that "similar" digits have strong connections: 4 and 9, 1 and 7, etc.

The visualizations on Fig. 8 show the joint representation learning and self-organization: epoch after epoch, the reconstruction quality of the prototypes improves (upper part of the figure), and well-organized regions are emerging on the map, with samples of the same class gathering in the same units (bottom part).

SOM update interval When using hybrid loss functions composed of terms with different optimization dynamics, it is common to update each term at different intervals. To prevent one term to prevail too much on the other, it may be optimized less frequently than the other. In our case, the SOM term is optimized faster, thus we tried to update it only every 10 or 100 SGD steps, while the reconstruction loss is updated at every step. However, no impact was observed.

*Gamma γ parameter*The gamma hyperparameter controls the relative weight of reconstruction and SOM in the optimization of the DESOM loss function. Its influence on training can be visualized on Fig. 9, representing $\mathcal{L}$ (total loss), $\mathcal{L}_{\text{R}}$ and $\mathcal{L}_{\text{SOM}}$ learning curves for different values of

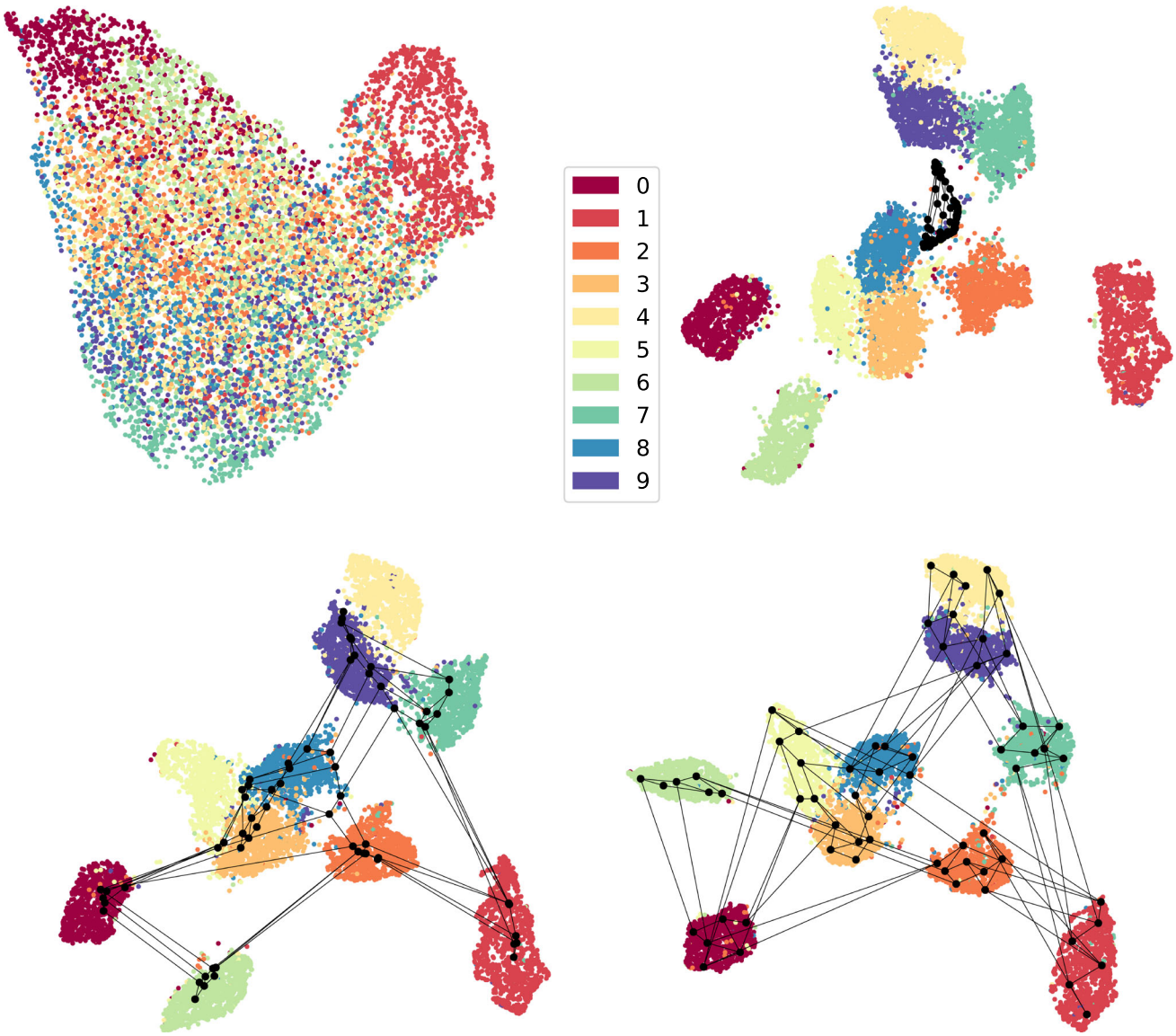

Fig. 7 UMAP visualization of latent space after 0, 10, 20 and 40 training epochs

gamma. We see that the SOM loss has a higher amplitude, hence using $\gamma < 1$.

If its value is too high, $\mathcal{L}_{\text{SOM}}$ is optimized too quickly, leaving a higher reconstruction loss. The comparison of both loss terms on Fig. 10 makes clear the trade-off on γ, leading to different final values for each term. As we have seen previously, $\gamma = 10^{-3}$ is an optimal value across datasets in terms of clustering quality, as we can see on Fig. 11 which presents the learning curves of purity and NMI for different values of gamma.

Batch size Training a SOM using minibatch stochastic gradient descent (here with the Adam optimizer) is unusual as it corresponds neither to the original stochastic Kohonen algorithm nor the batch version. However, it produces good results, even better than the standard SOM training (this was found in [19] and [17]). Let us denote n_b the batch size, i.e., the number of samples in each minibatch. The Kohonen algorithm would correspond to $n_b = 1$, and the batch algorithm to $n_b = N$ with N the total number of training samples.

Throughout our experiments, we use $n_b = 256$, as it is a common practice in deep learning to use the largest possible batch size in order to exploit GPU parallelization and accelerate training (the limit being graphics memory). In this experiment, we studied the impact of batch size on DESOM training, and made n_b vary in powers of two from 16 to 256. To be comparable, we adapted the number of iterations to keep the overall number of epochs constant (i.e., 10000 iterations for $n_b = 256$, 20000 for $n_b = 128$, etc.). Final external clustering metrics on the test set are

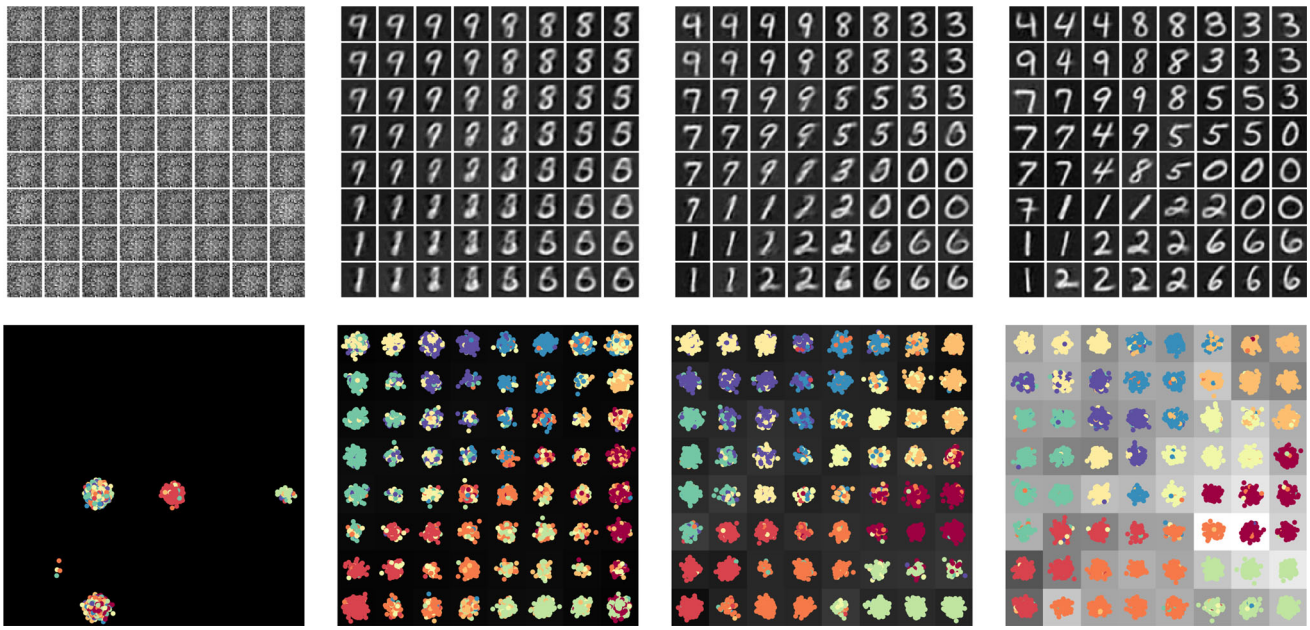

Fig. 8 (Top) decoded prototypes (bottom) samples projected on U-matrix and colored by class after 0, 10, 20 and 40 training epochs

presented in Fig. 12, and are clearly in favor of using the largest possible batch size.

On the other hand, SOM quality metrics such as quantization, topographic and combined errors are not sensitive to the batch size (see Fig. 13).

A large batch size is preferable as it improves both the solution and training speed due to parallelization.

4.4 Prototype image sharpness

As explained in the previous section, in case of image datasets, it is important for visualization that the SOM prototypes are realistic and not suffering from the blurring that can be caused by averaging. It is clear that *k*-means and standard SOM will produce a certain amount of blur, because they compute a (weighted) average of input images in original space. DESOM also computes a weighted average, but in the latent space of autoencoders. Our intuition is that the reconstructed prototype images would not suffer as much from this issue, because autoencoders learn a *flattened* latent space where linear operations (addition, interpolation, etc.) are meaningful (for example, [6] use linear interpolation and extrapolation for data augmentation and produce realistic outputs).

In Fig. 14, we represented an example of a prototype image of a 5 digit from the MNIST dataset, obtained by four different models. First, a standard SOM applied on the raw pixels; then, a SOM applied on the encoded dataset, using a standard autoencoder (AE+SOM); thirdly, the DESOM model; and finally, ConvDESOM. This basic visual inspection confirms that the prototypes learned by the deep models have better quality and are less blurred than the standard SOM, and with no surprise, the best visual quality is obtained with the convolutional variant ConvDESOM.

In order to quantify this "blurriness" or, equivalently, the "sharpness" of prototype images compared with the original samples, we introduce the prototype sharpness ratio (PSR) in Appendix 1. A good score should be very close to 1. Prototype sharpness ratio scores on benchmark image datasets for SOM, DESOM, ConvDESOM and variants are presented in Table 10, results section. On MNIST, SOM obtains a PSR of 0.720, while DESOM and ConvDESOM obtain, respectively, 1.030 and 1.045.

5 Experimental results

This section is dedicated to evaluation and comparison of various clustering and SOM-based methods in a large benchmark, in terms of clustering, visualization and classification performance. As previously, all experiments are run 10 times to obtain meaningful means and standard deviations, and use the standard train/test split (see Table 3), always reporting results on the test set (unless specified otherwise). On the four benchmark datasets, we evaluate different aspects and tasks. First, we assess clustering quality with respect to external labels, using purity and NMI. Then, we measure clustering and self-organization through SOM's quality indices in original and latent space. More qualitatively, we also assess the visual quality of the obtained maps, using the reconstructed prototypes.

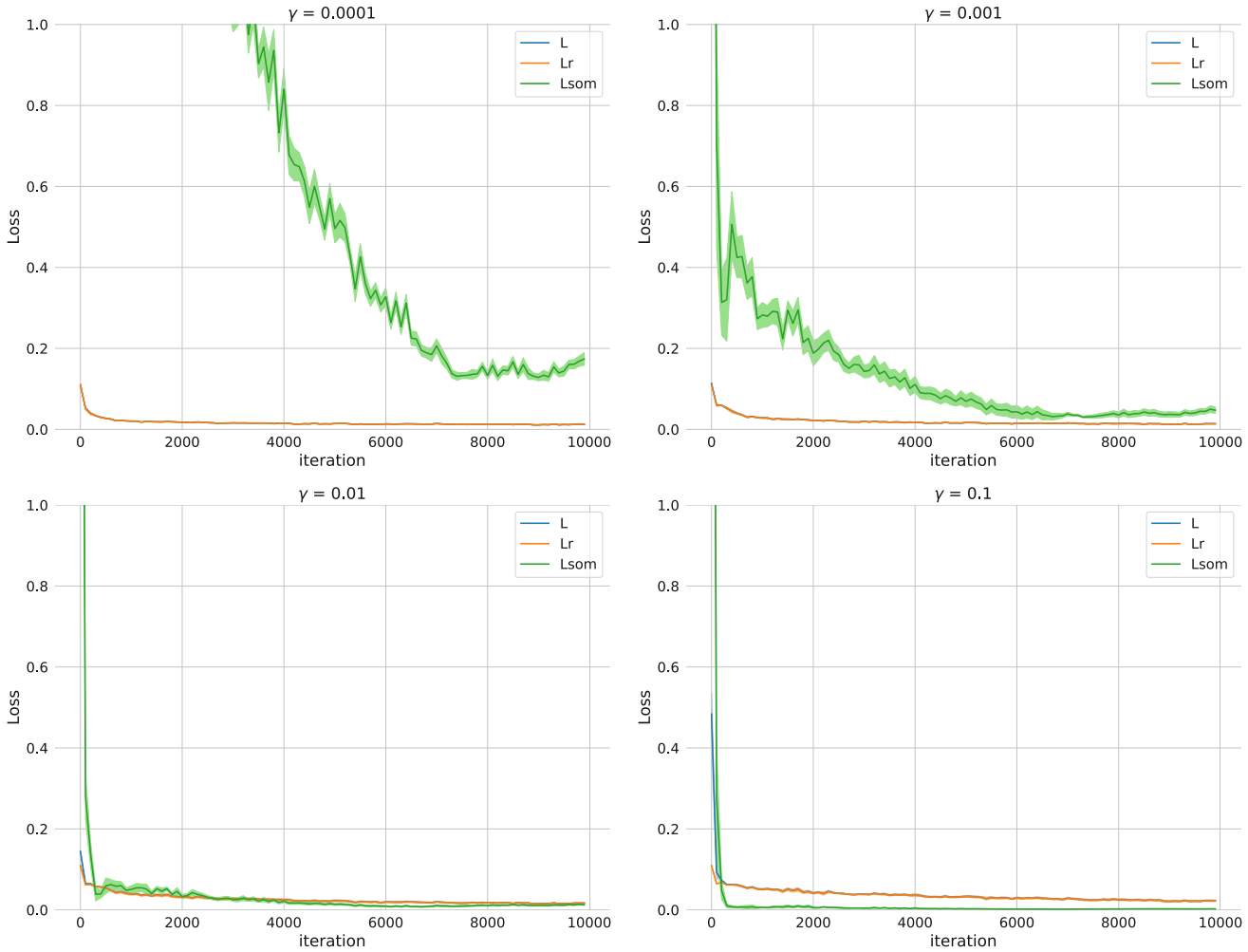

Fig. 9 Evolution of losses for different values of gamma

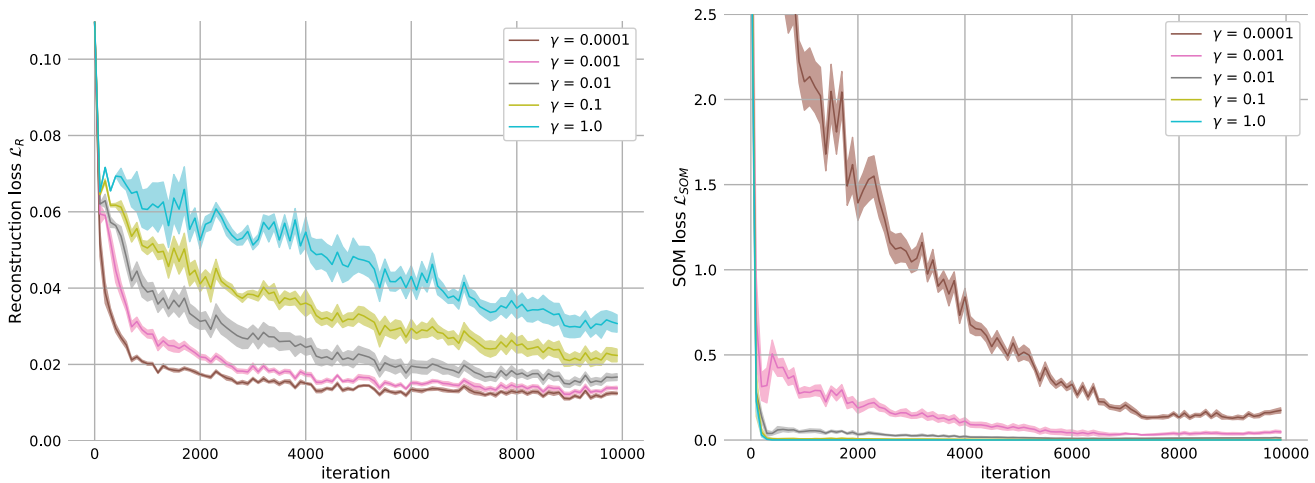

Fig. 10 Evolution of reconstruction loss and SOM loss for different values of the gamma hyperparameter, which trades off between optimizing the reconstruction and SOM loss functions.

Lastly, we compare different methods on a classification task, where the goal is to discriminate classes when the number of clusters equals the number of target classes, using unsupervised clustering accuracy.

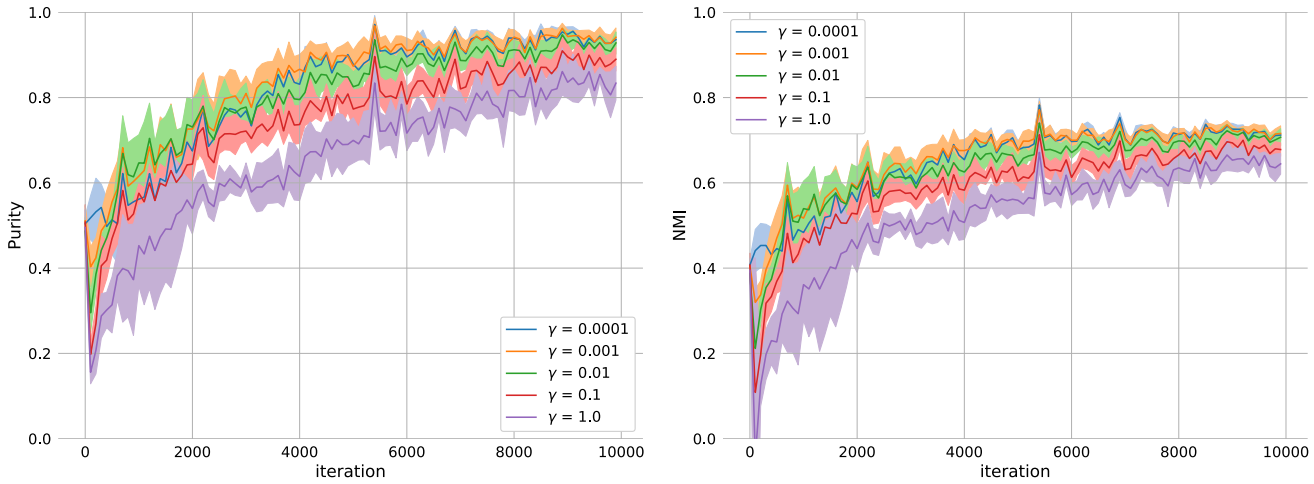

Fig. 11 Evolution of clustering quality (purity and NMI) for different values of the gamma hyperparameter. Small values lead to better clustering, but at cost of topology

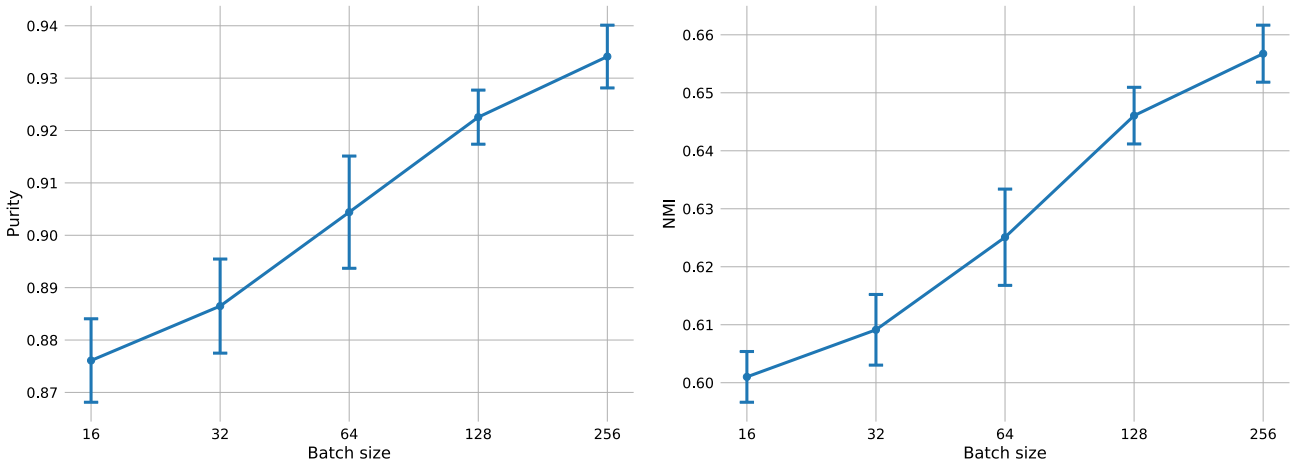

Fig. 12 Purity and NMI for different batch sizes. Larger batch sizes improve clustering results overall

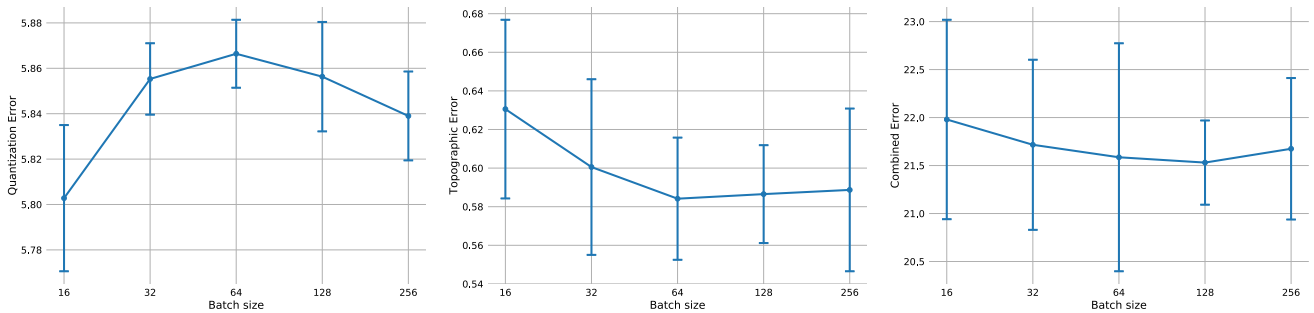

Fig. 13 Quantization, topographic and combined errors for different batch sizes. Medium values (64 or 128) seem preferable, but difference is negligible

5.1 Compared methods

The benchmarks compare following SOM-based methods:

- **SOM**: our implementation of a SOM in Keras (equivalent to DESOM with identity encoder and decoder) and trained by SGD.
- **AE+SOM**: two-stage approach, where SOM is learned on the encoded dataset using a pretrained autoencoder,

Fig. 14 Examples of (reconstructed) prototype images for SOM, AE+SOM, DESOM and ConvDESOM (from left to right). With standard SOM, images are blurry due to averaging in original space. The best visual quality is obtained by ConvDESOM

resulting in the DESOM model without joint optimization of the AE and SOM.

- **DESOM-AE+SOM**: same approach, but using the AE from a pretrained DESOM model, to study the impact of the SOM-guided regularization.
- **DESOM**: Deep Embedded SOM with joint representation learning and self-organization.

The maps are always 8×8-sized, excepted in the classification task, where we also compare a one-dimensional #classes $\times$ 1 map. The AE has always the same architecture as in previous experiments, i.e., [500, 500, 2000, 10] fully connected symmetric, except in ConvDESOM. We also include k-means (from scikit-learn module), AE+k-means and DESOM-AE+k-means, with 64 prototypes, as a baseline, even if it cannot be directly compared because it lacks self-organization.

When using SOM, practitioners often need interpretable high-level regions on the map, that discriminate between different classes or behaviors of the studied phenomenon. A common technique is to apply a subsequent clustering on the obtained prototypes, in order to reduce the number of clusters. They will often use a hierarchical clustering (HC) algorithm, as in [12]. Thus, we evaluate the class discrimination power of the previously listed models, but without removing the topology constraint between the prototypes. In order to do so, the number of clusters must be reduced to the number of target classes. We compare three different methods:

- Perform k-means clustering on the map prototypes with k = #classes.
- Perform Ward hierarchical clustering (HC) on the map prototypes with #classes clusters.
- Directly train a one-dimensional, wire-shaped map of size #classes $\times$1.

On this task, we measure purity, NMI and unsupervised clustering accuracy, the unsupervised counterpart of supervised classification accuracy.

Results are put in perspective with baseline state-of-the-art SOM-based models (SOM-VAE [19], DPSOM [34]) and deep clustering models (k-means, DEC, IDEC [20], DCN [50], DKM [11] and VaDE [24]), keeping in mind that these are pure clustering models and do not produce a

Table 7 Clustering performance of k-means and SOM-based models according to purity and NMI. Best performance among SOM-based models in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t test)

Method	MNIST		Fashion-MNIST	
	Pur	NMI	Pur	NMI
k-means ($k = 64$)	.845 ± .011	.581 ± .006	.718 ± .006	.514 ± .002
AE+k-means	.946 ± .004	.672 ± .005	.764 ± .005	.548 ± .003
DESOM-AE+k-means	.932 ± .005	.656 ± .006	.754 ± .008	.542 ± .003
SOM (8×8)	.832 ± .010	.576 ± .005	.712 ± .002	.513 ± .002
AE+SOM	**.935 ± .005**	**.666 ± .005**	**.758 ± .007**	**.542 ± .005**
DESOM-AE+SOM	**.933 ± .005**	.655 ± .005	**.756 ± .007**	**.543 ± .003**
DESOM (8×8)	**.934 ± .004**	.658 ± .004	**.751 ± .009**	**.541 ± .004**
SOM-VAE (8×8) [19]	.868 ± .003	.595 ± .002	.739 ± .002	.520 ± .002
DPSOM (8×8) [34]	.964 ± .001	.705 ± .001	.764 ± .003	.571 ± .001
Method	USPS		REUTERS-10k	
	Pur	NMI	Pur	NMI
k-means ($k = 64$)	.858 ± .007	.598 ± .004	.895 ± .007	.439 ± .007
AE+k-means	.874 ± .007	.611 ± .006	.856 ± .016	.392 ± .014
DESOM-AE+k-means	.858 ± .010	.592 ± .009	.798 ± .028	.360 ± .018
SOM (8×8)	**.848 ± .009**	.595 ± .007	.554 ± .046	.225 ± .063
AE+SOM	**.849 ± .015**	**.611 ± .010**	.782 ± .020	.323 ± .018
DESOM-AE+SOM	**.852 ± .007**	.589 ± .007	**.799 ± .021**	**.355 ± .017**
DESOM (8×8)	**.857 ± .011**	.592 ± .010	**.808 ± .017**	**.364 ± .011**

self-organized map. Values are taken directly from the papers (references in the results Tables 7 and 11), and were in some cases reported for the entire dataset, and not only the hold-out test set.

5.2 Clustering benchmark

Results of the clustering benchmark are displayed in Table 7. The first statement that is not surprising but clearly confirmed here, is that reducing dimensionality with neural networks improves clustering quality, also in the case of SOM. If we compare SOM vs. AE+SOM, every time there is a considerable performance gain.

Overall, AE+SOM (two-stage training) and DESOM (joint training) have similar performance across the first three datasets, with an advantage for AE+SOM. However, DESOM outperforms AE+SOM by a fair margin (approx. +3% in purity and +10% in NMI) on REUTERS-10k, where the AE alone struggles to find good representations for the high-dimensional text data (the AE even decreases performance for *k*-means). Thus, the joint training of DESOM does not bring consistent quantitative benefits in terms of purity or NMI, as seen in deep clustering approaches, but it is at least close to the two-stage approach and much faster to train (no pretraining). Using the AE of DESOM in a two-stage approach yields similar or lower scores than joint training with DESOM. On Reuters-10k, we see that DESOM-AE has learned a better code than the standard AE.

As noted by [19] and in our previous work [17], the SOM trained by SGD with Adam achieves much higher clustering quality than standard SOM implementations, and benefits from GPU acceleration. Thus, we did not include another SOM implementation in this work.

Comparing with other SOM-based models, DESOM consistently outperforms SOM-VAE on MNIST and Fashion-MNIST according to purity and NMI, but is second to the very recent DPSOM model, achieving state-of-the-art results on both datasets. However, it is difficult to directly compare with these models, because they use a variational autoencoder (VAE), and the authors do not measure other metrics than purity and NMI (e.g., topographic organization). We believe the superiority of DPSOM is partly due to using a VAE. Its regularization of the latent space through a Gaussian prior leads to extracting better representations, reflected by the class labels used by purity and NMI, as well as better generalization (preventing overfitting on training samples). Then, they also optimize a soft clustering loss (similar to DEC [49]), in addition to the SOM loss (with only 4 neighbors), while we directly optimize SOM distortion with a Gaussian neighborhood. This also helps pure clustering quality. However, authors do not evaluate other aspects such as SOM quality (topographic organization).

Tables 8 and 9 present quantization, topographic and combined errors as well as topographic product in original and latent space for all datasets. In original space, comparison between SOM and DESOM shows that generally DESOM obtain inferior quantization and topology (excepted on Reuters-10k). Joint representation learning implies a trade-off on SOM training: deep representations enable learning meaningful clusters with respect to latent factors (e.g., classes), but has an impact on the self-organization of SOM. The same metrics can be defined in latent space (see Appendix 1), to compare the AE+SOM, DESOM-AE+SOM and DESOM approaches. Here, results clearly point toward the advantage of the regularized, SOM-friendly latent space, because quantization and topology are by far superior with DESOM's autoencoder than with the non-regularized AE. Latent quantization, topographic and combined errors are similar for DESOM-AE+SOM and DESOM, but much higher for AE+SOM. Latent topographic products are closer to zero for every dataset, meaning that the map is less stretched or distorted in DESOM's latent space.

5.3 Visualization

In this paragraph, we visualize and compare maps of image datasets obtained by SOM, DESOM and ConvDESOM. For image datasets, we can directly visualize the prototypes or reconstructed prototypes using the decoder. Maps for MNIST and Fashion-MNIST are shown on Figs. 15 and 16. We clearly see the regions corresponding to different classes and smooth transitions between them. The advantage of fitting the map in latent space instead of the original high-dimensional space is visible when comparing the prototypes learned by SOM and DESOM. With SOM, they are very blurred (caused by averaging of data vectors in original space), compared with the decoded prototypes of DESOM. A few SOM prototypes are not realistic samples; however, the topographic organization looks smoother with SOM (this was quantified in previous paragraphs). Visually, the best map is obtained with ConvDESOM, as the reconstruction quality is higher.

In order to assess quantitatively the visual quality of prototypes, we measure prototype sharpness ratio (PSR) for each model (see Table 10). A value close to 1 indicates sharpness close to the original samples, whereas a lower value points toward blurriness. SOM prototypes are blurry and obtain a low PSR, around 0.7. Models equipped with an autoencoder obtain scores close to 1 and statistically equivalent, with an advantage for DESOM.

Visualizations of larger maps are available in Appendix 2.

Table 8 Comparison between SOM and DESOM using internal quality indices in original space. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t test)

Method	MNIST			
	QE	TE	CE	TP
SOM (8×8)	**5.345 ± 0.004**	**0.518 ± 0.037**	**15.78 ± 0.662**	**−0.073 ± 0.004**
DESOM (8×8)	5.848 ± 0.016	0.597 ± 0.033	21.74 ± 0.643	-0.104 ± 0.004
Method	Fashion-MNIST			
	QE	TE	CE	TP
SOM (8×8)	**4.537 ± 0.008**	**0.477 ± 0.037**	**12.40 ± 0.654**	**−0.026 ± 0.007**
DESOM (8×8)	4.755 ± 0.007	0.536 ± 0.035	15.22 ± 0.941	-0.046 ± 0.005
Method	USPS			
	QE	TE	CE	TP
SOM (8×8)	**3.693 ± 0.005**	**0.474 ± 0.025**	**10.35 ± 0.378**	**−0.055 ± 0.007**
DESOM (8×8)	4.025 ± 0.018	0.556 ± 0.041	14.62 ± 0.649	-0.082 ± 0.005
Method	Reuters-10k			
	QE	TE	CE	TP
SOM (8×8)	42.70 ± 0.108	**0.595 ± 0.311**	**102.4 ± 21.26**	-0.206 ± 0.015
DESOM (8×8)	**41.81 ± 0.102**	**0.754 ± 0.072**	**113.8 ± 7.927**	**−0.147 ± 0.005**

5.4 Classification benchmark

The classification task consists in discriminating classes when the number of clusters equals the number of target classes. The number of clusters is reduced by applying a subsequent k-means or hierarchical clustering on top of the 8×8 map prototypes. Purity, NMI and unsupervised clustering accuracy results are displayed in Table 11. The HC approach, often used by practitioners, is more efficient than the k-means approach in most cases.

On MNIST and Fashion-MNIST, the DESOM+HC approach consistently outperforms every other method on all three metrics. It obtains 81% accuracy on MNIST, which challenges deep clustering methods without self-organization constraints, such as DEC [49]. The AE+SOM+HC method is also very competitive. On Fashion-MNIST, the DESOM-based methods are clearly superior. Generally, reducing dimensionality with an AE greatly improves results, but USPS is an exception with SOM+HC performing better than AE+SOM+HC. We assume this is due to the relatively low dimension of this dataset (256). However, DESOM+HC still achieves the best unsupervised clustering accuracy, which is the most important metric for this task. Lastly, on Reuters-10k the DESOM-based approaches again produce the best classification results. This time, DESOM+k-means performs best, but DESOM+HC is statistically equivalent.

These results demonstrate that DESOM's self-organizing map prior has enabled to learn a *SOM-friendly* representation that improves classification accuracy when classifying the map prototypes with HC or k-means. DESOM performs best on all four datasets is terms of purity, NMI and accuracy, the only exception being NMI on USPS.

The one-dimensional DESOM achieves decent results and even the best on Reuters-10k, showing that the very high-dimensional TF-IDF features benefit from joint learning. It even outperforms k-means, which has no topology constraint. However, such a map provides much less information on the data distribution and topology than a larger two-dimensional map.

5.5 Training time

We report training times of the compared methods for MNIST in Table 12. All models were trained on a RTX2080 GPU card in our lab, with a batch size of 256. Autoencoder end-to-end pretraining for 100 epochs lasts 2 mins. The training time of SOM (our SGD-based Keras implementation) is also 2 mins for 10000 iterations, giving a total training time of about 4 mins for the two-stage AE+SOM approach. This is identical to DESOM, which trains in 4 mins. If AE pretraining was necessary for DESOM, training time would jump to a total of 6 minutes,

Table 9 Comparison between SOM, AE+SOM and DESOM using internal quality indices in latent space. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t test)

Method	MNIST			
	$\hat{\text{QE}}$	$\hat{\text{TE}}$	$\hat{\text{CE}}$	$\hat{\text{TP}}$
AE+SOM (8 × 8)	1.231 ± 0.029	**0.510 ± 0.047**	4.429 ± 0.287	−0.066 ± 0.002
DESOM-AE+SOM (8 × 8)	**0.205 ± 0.008**	**0.514 ± 0.031**	**0.713 ± 0.017**	**−0.055 ± 0.003**
DESOM (8 × 8)	**0.205 ± 0.008**	**0.534 ± 0.026**	**0.727 ± 0.058**	**−0.057 ± 0.005**
Method	Fashion-MNIST			
	$\hat{\text{QE}}$	$\hat{\text{TE}}$	$\hat{\text{CE}}$	$\hat{\text{TP}}$
AE+SOM (8 × 8)	0.960 ± 0.026	**0.532 ± 0.022**	3.973 ± 0.250	−0.059 ± 0.007
DESOM-AE+SOM (8 × 8)	**0.166 ± 0.003**	0.572 ± 0.037	**0.664 ± 0.041**	**−0.044 ± 0.003**
DESOM (8 × 8)	**0.167 ± 0.003**	**0.556 ± 0.035**	**0.661 ± 0.036**	**−0.045 ± 0.005**
Method	USPS			
	$\hat{\text{QE}}$	$\hat{\text{TE}}$	$\hat{\text{CE}}$	$\hat{\text{TP}}$
AE+SOM (8 × 8)	3.926 ± 0.151	0.689 ± 0.104	19.88 ± 2.737	−0.098 ± 0.007
DESOM-AE+SOM (8 × 8)	**0.278 ± 0.009**	**0.554 ± 0.033**	**1.174 ± 0.085**	**−0.065 ± 0.005**
DESOM (8 × 8)	**0.280 ± 0.007**	**0.563 ± 0.031**	**1.184 ± 0.037**	**−0.069 ± 0.003**
Method	Reuters-10k			
	$\hat{\text{QE}}$	$\hat{\text{TE}}$	$\hat{\text{CE}}$	$\hat{\text{TP}}$
AE+SOM (8 × 8)	30.00 ± 0.867	0.934 ± 0.017	270.7 ± 13.72	−0.146 ± 0.010
DESOM-AE+SOM (8 × 8)	**0.527 ± 0.017**	**0.710 ± 0.035**	**3.391 ± 0.401**	**−0.071 ± 0.007**
DESOM (8 × 8)	**0.524 ± 0.015**	**0.696 ± 0.065**	**3.102 ± 0.289**	**−0.069 ± 0.004**

a +50% increase. Overall, these SGD-based, GPU-accelerated methods are orders of magnitudes faster than standard SOM implementations that cannot handle datasets of this size in a reasonable time. Finally, we think that these low training times make SOM and DESOM very effective tools for surveying large, high-dimensional datasets.

6 Reproducibility

Reproducibility is of utmost importance in machine learning research. We took special care that all our experiments and benchmarks are fully reproducible. First, the implementation of the model is open source and available online at https://github.com/FlorentF9/DESOM. It uses the popular framework Keras. The repository provides instructions on how to use the model with the datasets used in our experiments, and external datasets also can be used with minimal effort, in order to apply our model on various use cases. All the hyperparameters and knobs of the model have default values, but they can all be tuned and are not hard-coded. To reproduce experiments, we have created a Linux shell script for each experiment and benchmark that automatically lauches the training procedures with the right parameters. Results of each experiment and run are saved to disk into plain files, and can be aggregated using a provided script to obtain averages and standard deviations. This allows to reproduce any experiment or benchmark by running a single shell script. In addition, the performance metrics are also open source, available at https://github.com/FlorentF9/SOMperf, making sure that the exact values reported in the tables of this paper can be reproduced. Finally, our experiments generated a vast amount of graphs and tables (for every experiment, metric, hyperparameter, dataset, etc.); only a few of them are included and discussed in this paper, but we make them all available in the companion repository.

7 Conclusion and perspectives

DESOM is an unsupervised learning algorithm that jointly trains an autoencoder and the code vectors of a self-organizing map in a continuous latent space in order to survey, cluster and visualize large, high-dimensional datasets. It is one of the first members of what we could call the *deep SOM* family, along with several other recent concurrent works. Joint optimization allows to integrate

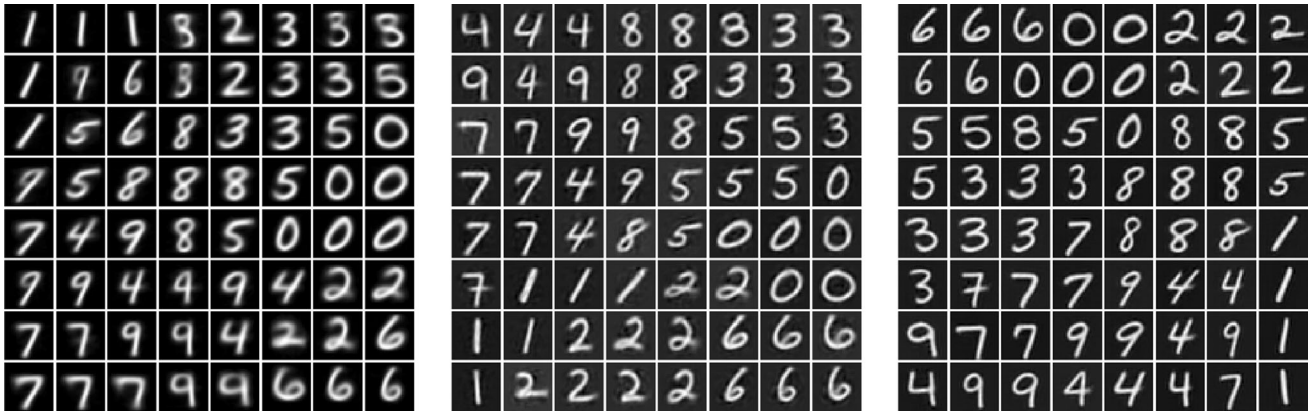

Fig. 15 Prototypes (decoded) visualized on SOM, DESOM and ConvDESOM maps for MNIST

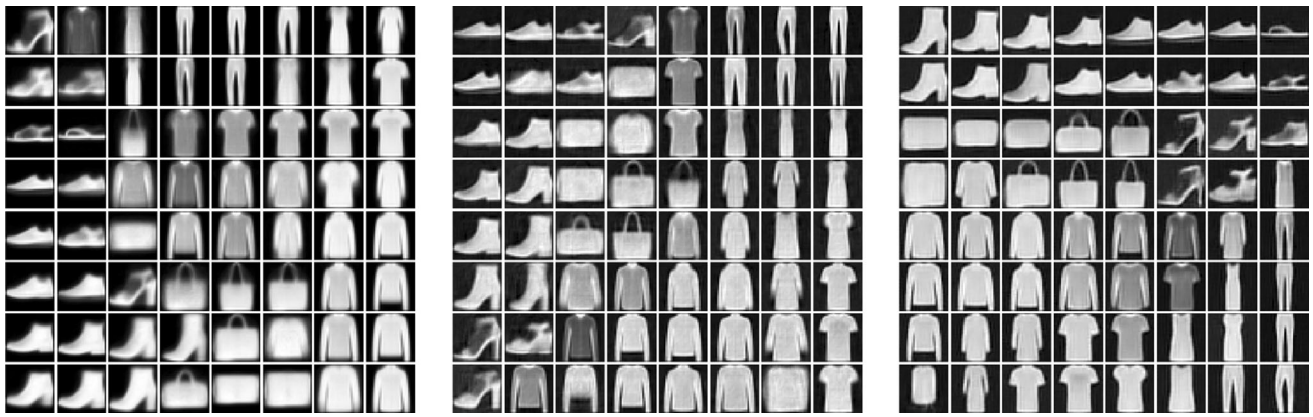

Fig. 16 Prototypes (decoded) visualized on SOM, DESOM and ConvDESOM maps for Fashion-MNIST

Table 10 Prototype sharpness ratio (PSR). Closest to 1 is best. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value $>$ 0.05 after pairwise t test)

Method	MNIST	Fashion-MNIST	USPS
SOM (8×8)	0.720 ± 0.005	0.703 ± 0.005	0.769 ± 0.005
AE+SOM (8×8)	1.083 ± 0.012	$\underline{\mathbf{0.834 \pm 0.011}}$	0.777 ± 0.016
DESOM-AE+SOM (8×8)	$\mathbf{1.034 \pm 0.011}$	$\mathbf{0.825 \pm 0.010}$	$\mathbf{1.041 \pm 0.021}$
DESOM (8×8)	$\underline{\mathbf{1.030 \pm 0.005}}$	$\mathbf{0.832 \pm 0.013}$	$\underline{\mathbf{1.027 \pm 0.020}}$
ConvDESOM (8×8)	$\mathbf{1.045 \pm 0.027}$	$\mathbf{0.829 \pm 0.009}$	-

dimensionality reduction to SOM learning, and to seek a *SOM-friendly* latent space that improves the performance of SOM. The goal of this paper is to extend and provide further insights to the previous conference papers. The model is governed by several hyperparameters impacting training and performance, in particular the trade-off between clustering and self-organization. This work presented experiments and visualizations in order to understand these effects. In our experiments, we found that reducing dimension with an autoencoder vastly improves SOM clustering with respect to latent factors of variation. We also found that the learned latent space does not in general improve quantitative clustering quality compared with the representations produced by a pure autoencoder model; however, DESOM results are on par with the two-stage approaches (AE+SOM), while requiring no pre-training at all and thus cutting down training time, which is an important criterion when quickly exploring large datasets. Training time is only a few minutes on medium-sized benchmark datasets. On a classification task where the goal

Table 11 Classification performance when number of clusters equals number of classes. Best performance among SOM-based models in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t test)

Method	MNIST			Fashion-MNIST		
	Pur	NMI	Acc	Pur	NMI	Acc
k-means (k =#classes)	.591 ± .026	.501 ± .020	.533 ± .038	.583 ± .018	.513 ± .012	.549 ± .040
AE+k-means	.820 ± .019	.754 ± .013	.801 ± .027	.544 ± .010	.524 ± .012	.489 ± .017
DESOM-AE+k-means	.770 ± .025	.701 ± .018	.744 ± .045	.625 ± .014	.590 ± .009	.586 ± .014
DEC [20]	–	.837	.866	–	–	–
IDEC [20]	–	.867	.881	–	–	–
DCN [50]	–	.810	.830	–	–	–
DKM [11]	–	.796 ± .009	.840 ± .022	–	–	–
VaDE [24]	–	–	.945	–	–	–
SOM (8 × 8)+KM	.559 ± .053	.485 ± .041	.510 ± .071	.538 ± .023	.499 ± .024	.498 ± .035
SOM (8 × 8)+HC	.641 ± .030	.612 ± .027	.598 ± .035	.550 ± .032	.536 ± .022	.491 ± .041
AE+SOM+KM	.773 ± .051	.728 ± .032	.728 ± .073	.485 ± .041	.461 ± .032	.439 ± .048
AE+SOM+HC	**.822 ± .024**	**.788 ± .030**	**.791 ± .026**	.528 ± .029	.550 ± .026	.480 ± .032
DESOM-AE+SOM+KM	.743 ± .049	.690 ± .033	.720 ± .057	**.588 ± .024**	.583 ± .012	.535 ± .033
DESOM-AE+SOM+HC	.747 ± .041	.681 ± .036	.721 ± .056	**.598 ± .042**	**.586 ± .036**	**.553 ± .047**
DESOM (8 × 8)+KM	.751 ± .048	.696 ± .036	.717 ± .065	**.587 ± .044**	.582 ± .025	**.536 ± .053**
DESOM (8 × 8)+HC	**.824 ± .024**	**.793 ± .025**	**.810 ± .032**	**.613 ± .035**	**.604 ± .019**	**.571 ± .036**
DESOM (#classes × 1)	.790 ± .017	.720 ± .020	.779 ± .033	.563 ± .019	.553 ± .011	.546 ± .025
	USPS			REUTERS-10k		
Method	Pur	NMI	Acc	Pur	NMI	Acc
k-means (k = #classes)	.703 ± .026	.585 ± .019	.660 ± .032	.647 ± .082	.373 ± .085	.589 ± .096
AE+k-means	.720 ± .033	.604 ± .026	.680 ± .063	.589 ± .044	.235 ± .046	.538 ± .041
DESOM-AE+k-means	.698 ± .027	.575 ± .024	.648 ± .032	.615 ± .067	.257 ± .062	.533 ± .069
DEC [20]	–	.753	.741	–	.498	.737
IDEC [20]	–	.785	.761	–	.498	.756
DCN [50]	–	–	–	–	.760	.800
DKM [11]	–	.776 ± .011	.757 ± .013	–	.331 ± .049	.583 ± .038
VaDE [24]	–	–	–	–	–	.798
SOM (8 × 8)+KM	.659 ± .029	.571 ± .016	.629 ± .043	.443 ± .070	.107 ± .099	.434 ± .072
SOM (8 × 8)+HC	**.711 ± .019**	**.650 ± .014**	.666 ± .024	.444 ± .028	.105 ± .041	.439 ± .030
AE+SOM+KM	.615 ± .027	.540 ± .027	.585 ± .049	.456 ± .058	.115 ± .067	.415 ± .042
AE+SOM+HC	**.673 ± .056**	.591 ± .043	**.649 ± .070**	.469 ± .057	.128 ± .058	.441 ± .071
DESOM-AE+SOM+KM	.639 ± .063	.545 ± .048	.621 ± .067	**.571 ± .050**	**.206 ± .050**	**.478 ± .061**
DESOM-AE+SOM+HC	.631 ± .041	.540 ± .035	.610 ± .040	**.536 ± .063**	**.197 ± .061**	**.467 ± .065**
DESOM (8 × 8)+KM	.636 ± .043	.543 ± .032	.611 ± .058	**.573 ± .059**	**.210 ± .063**	**.510 ± .073**
DESOM (8 × 8)+HC	**.710 ± .023**	.633 ± .017	**.698 ± .030**	**.521 ± .079**	**.186 ± .095**	**.486 ± .094**
DESOM (#classes × 1)	.683 ± .035	.556 ± .033	.649 ± .038	.661 ± .039	.322 ± .041	.596 ± .045

is to discriminate between target classes, after post-clustering the SOM code vectors, DESOM consistently outperforms comparable models.

Not every aspect has been tackled in this work. We conducted experiments using a fully connected or convolutional autoencoder network, but an extension to sequences with a recurrent autoencoder is possible. In addition, the model could be extended to the variational or adversarial frameworks (VAE or GAN), which could improve the quality of learned representations and provide us with a generative model. We also did not try different SOM neighborhood functions and radius decays, assuming

Table 12 Training times on MNIST (60000 samples, batch size 256)

Model	Average training time
AE pretraining (100 epochs)	2 min
SOM (10000 iterations)	2 min
AE+SOM (10000 iterations)	4 min
DESOM (10000 iterations)	4 min

that they are only related to the SOM learning and should not fundamentally change the properties of DESOM. However, this cannot be excluded due to the interaction between SOM loss and reconstruction loss. All these perspectives are left as future work.

Appendix 1: performance evaluation

Quantitative evaluation of self-organizing maps is not as straightforward as for supervised classification tasks. To assess and compare the performance of models, we implemented and evaluated a collection of metrics that have been developed in related literature. This literature spans almost 30 years of history, and implementations are not easy to find. Thus, we implemented these metrics and provide them as an open-source library, SOMperf[6] [18]. First, SOM performance metrics can be categorized into two families:

1. Clustering metrics. Any clustering quality measure that relies solely on the prototype vectors and not on their topological organization. This encompasses all quality indices used in clustering literature (e.g., purity, normalized mutual information (NMI), Rand index, etc.).
2. Topographic metrics. Under this term, we coin quality measures that, on the contrary, assess the topological organization of the model. Some indices also evaluate the clustering quality, but we call a metrics topographic as soon as it incorporates the map topology.

On another level, we can also classify them into two well-known families, depending on the use or not of ground-truth label knowledge:

1. Internal indices, using only intrinsic properties of the model and the data.
2. External indices, relying on external ground-truth class labels to evaluate results, as in supervised classification.

For instance, quantization error falls into the clustering metric category (as it measures how the SOM cluster centers fit the data distribution, without using any topology information) and is an internal indices (not depending on external labels). On the other side, the Class Scatter Index (introduced in [10]) is a topographic metric and an external indices, as it measures how ground-truth class labels are organized into groups of neighboring map units.

Clustering metrics

Internal indices

Quantization error Quantization error is the average error made by projecting data on the SOM, as measured by Euclidean distance, i.e., the mean Euclidean distance between a data sample and its best-matching unit. It can be measured in the original space, using the prototypes reconstructed by the decoder (QE), or in latent space ($\hat{\text{QE}}$), introducing the notations $\tilde{b_i}$ for the best-matching unit of data point $\mathbf{x}_i$ in original space, and b_i for the best-matching unit of $\mathbf{z}_i$ in latent space.

$$\text{QE}(\{\tilde{\mathbf{m}}_k\}, \mathbb{X}) = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{x}_i - \tilde{\mathbf{m}}_{b_i}||_2$$

$$\hat{\text{QE}}(\{\mathbf{m}_k\}, \mathbb{Z}) = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{z}_i - \mathbf{m}_{b_i}||_2$$

Prototype sharpness ratio In the case of images, we want the prototype images to be realistic and as *sharp* as the original images. When visualizing a self-organized map of an image dataset, we usually simply visualize the image corresponding to each prototype vector. If they are blurry (due to the averaging induced by the SOM algorithm), the visualization will be of poor quality, even if the quantization error is low (because QE is only an average Euclidean distance). We have chosen a very simplistic way to compute the sharpness of an image, which is the average norm of pixel gradients, measured in two dimensions. The sharpness of a SOM is defined as the average sharpness of its prototypes. This sharpness measure can then be compared with the average sharpness of images in the original dataset. We introduce the *prototype sharpness ratio* (PSR), defined as follows:

$$\text{PSR}(\{\tilde{\mathbf{m}}_k\}, \mathbb{X}) = \frac{\text{average prototype sharpness}}{\text{average dataset sharpness}} = \frac{\frac{1}{K}\sum_{k=1}^{K} ||\nabla_{2\text{D}}\tilde{\mathbf{m}}_k||_2^2}{\frac{1}{N}\sum_{i=1}^{N} ||\nabla_{2\text{D}}\mathbf{x}_i||_2^2}$$

A score lower than 1 means that the prototypes are in average blurrier than the original images; on the contrary, if it is larger than 1, they are less blurry (i.e., more crisp or noisy) than the originals. The closer the PSR is to 1, the better is the score.

[6] https://github.com/FlorentF9/SOMperf.

External indices

A clustering with K clusters is described by the sets of data points belonging to each cluster, noted $\mathbf{Q} = \{Q_k\}, k = 1 \ldots K$. In order to define the external clustering criteria, we assume that labels are associated with each data point, corresponding to a set of C different classes. We note $\mathbf{Y} = \{Y_j\}, j = 1 \ldots C$ the sets of elements belonging to each class.

Purity Purity is one of the most commonly used external quality indices. It measures the purity of clusters with respect to ground-truth class labels. To compute the purity of a clustering $\mathbf{Q}$, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned points and dividing by the total number of points. Formally:

$$\mathrm{Pur}(\mathbf{Q}, \mathbf{Y}) = \frac{1}{N} \sum_{k=1}^{K} \max_{j=1 \ldots C} |Q_k \cap Y_j| \tag{9}$$

High purity is easy to achieve when the number of data points per cluster is small; in particular, purity is equal to 1 if all points get their own cluster. Thus, purity cannot be used to trade off the validity of the clustering against the number of clusters.

Normalized mutual information Normalized mutual information (NMI) is also one of the most widespread external clustering indices:

$$\mathrm{NMI}(\mathbf{Q}, \mathbf{Y}) = \frac{I(\mathbf{Q}, \mathbf{Y})}{\frac{1}{2}(H(\mathbf{Q}) + H(\mathbf{Y}))}$$

where I is mutual information:

$$\begin{aligned} I(\mathbf{Q}, \mathbf{Y}) &= \sum_k \sum_j P(Q_k \cap Y_j) \log \frac{P(Q_k \cap Y_j)}{P(Q_k) P(Y_j)} \\ &= \sum_k \sum_j \frac{|Q_k \cap Y_j|}{N} \log \frac{N |Q_k \cap Y_j|}{|Q_k| |Y_j|} \end{aligned}$$

and H is entropy:

$$H(\mathbf{Q}) = -\sum_k P(Q_k) \log P(Q_k) = -\sum_k \frac{|Q_k|}{N} \log \frac{|Q_k|}{N}$$

Unsupervised clustering accuracy Accuracy is the most common metric used in supervised classification: it corresponds to the number of examples that are assigned to the correct class divided by the total number of examples in the dataset. It can also be used in clustering (i.e., unsupervised classification) as an external quality measure if labels are available and if the number of clusters is equal to the number of classes. It consists in the accuracy of the resulting classification using the best one-to-one mapping between clusters and class labels. Denoting this mapping by m, the expression of unsupervised clustering accuracy is:

$$\mathrm{Acc}(\mathbf{Q}, \mathbf{Y}) = \frac{1}{N} \max_m \sum_{k=1}^{K} |Q_k \cap Y_{m(k)}| \tag{10}$$

The best mapping can be found using the Hungarian assignment algorithm, also known as the Kuhn–Munkres algorithm.

Topographic metrics

Distortion Distortion is the loss function minimized by the SOM learning algorithm. It is similar to the within-cluster sum of squared errors (WSSE) minimized by k-means, but with an additional topology constraint introduced by the neighborhood function. It is calculated by the sum of squared Euclidean distances between each map prototype and data sample, weighted by the neighborhood function according to the distance between the sample and the prototype. As quantization error, it can be measured in the original data space or in latent space:

$$\mathrm{D}(\{\tilde{\mathbf{m}}_k\}, \mathbb{X}, T) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{K}^T\left(\delta(\tilde{b_i}, k)\right) ||\mathbf{x}_i - \tilde{\mathbf{m}}_k||_2^2$$

$$\hat{\mathrm{D}}(\{\mathbf{m}_k\}, \mathbb{Z}, T) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{K}^T(\delta(b_i, k)) ||\mathbf{z}_i - \mathbf{m}_k||_2^2$$

In DESOM, the SOM loss corresponds to latent distortion.

Topographic error Topographic error assesses the self-organization of a SOM model. It is calculated as the fraction of samples whose best and second-best matching units are not neighbors on the map. In other words, this error quantifies the smoothness of projections on the self-organized map. Using the notations $\tilde{b_i^k}$ and b_i^k for the k-th best-matching units of $\mathbf{x}_i$ and $\mathbf{z}_i$ in original and latent space, respectively, we define topographic error TE and latent topographic error $\hat{\mathrm{TE}}$:

$$\mathrm{TE}(\{\tilde{\mathbf{m}}_k\}, \mathbb{X}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\delta(\tilde{b_i^1}, \tilde{b_i^2}) > 1}$$

$$\hat{\mathrm{TE}}(\{\mathbf{m}_k\}, \mathbb{Z}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\delta(b_i^1, b_i^2) > 1}$$

Combined error Combined error [25] is an error measure that combines and extends quantification and topographic errors. Its computation is more complex than the previous indices. For a given data sample $\mathbf{x}_i$, we first compute its two best matching units $\tilde{b_i^1}$ and $\tilde{b_i^2}$. Then, we compute a sum of Euclidean distances from $\mathbf{x}_i$ to the second BMU's prototype vector $\mathbf{m}_{\tilde{b_i^2}}$, starting with the distance from $\mathbf{x}_i$ to $\mathbf{m}_{\tilde{b_i^1}}$, and thereafter following a shortest path until $\mathbf{m}_{\tilde{b_i^2}}$,

going through only neighboring units on the map. Finally, the combined error (CE) is the average of this distance over the input samples.

Let p be a path on the map of length $P \geq 1$, from $p(0) = \tilde{b}_i^1$ to $p(L) = \tilde{b}_i^2$, such that $p(k)$ and $p(k+1)$ must be neighbors for $k = 0 \ldots P-1$. The distance along the shortest path on the map is computed as:

$$\mathrm{CE}_i = ||\mathbf{x}_i - \tilde{\mathbf{m}}_{\tilde{b}_i^1}||_2^2 + \min_p \sum_{k=0}^{P-1} ||\tilde{\mathbf{m}}_{p(k+1)} - \tilde{\mathbf{m}}_{p(k)}||_2^2$$

Finally, the combined error is:

$$\mathrm{CE}(\{\tilde{\mathbf{m}}_k\}, \mathbb{X}) = \frac{1}{N} \sum_{i=1}^{N} \mathrm{CE}_i$$

As usual, we also define combined error in latent space:

$$\hat{\mathrm{CE}}_i = ||\mathbf{z}_i - \mathbf{m}_{b_i^1}||_2^2 + \min_p \sum_{k=0}^{L-1} ||\mathbf{m}_{p(k+1)} - \mathbf{m}_{p(k)}||_2^2$$

$$\hat{\mathrm{CE}}(\{\mathbf{m}_k\}, \mathbb{Z}) = \frac{1}{N} \sum_{i=1}^{N} \hat{\mathrm{CE}}_i$$

Topographic product The topographic product (TP) [3] measures the preservation of neighborhood relations between vector space and the map. It depends only on the prototype vectors and map topology, and is able to indicate whether the dimension of the map is appropriate to fit the dataset, or if it introduced neighborhood violations.

We will note d the Euclidean distance in vector space, and δ the topographic distance on the map. The computation of TP starts by defining two ratios between the distance of a prototype j to its k-th nearest neighbor on the map $n_k^\delta(j)$, and to its k-th nearest neighbor in vector space $n_k^d(j)$:

$$Q_1(j,k) = \frac{d\left(\tilde{\mathbf{m}}_j, \tilde{\mathbf{m}}_{n_k^\delta(j)}\right)}{d\left(\tilde{\mathbf{m}}_j, \tilde{\mathbf{m}}_{n_k^d(j)}\right)}, \; Q_2(j,k) = \frac{\delta\left(j, n_k^\delta(j)\right)}{\delta\left(j, n_k^d(j)\right)}$$

Naturally, we always have $Q_1 \geq 1$ and $Q_2 \leq 1$. The ratios are combined into a product in order to obtain a symmetric measure and mitigate local magnification factors:

$$P_3(j,k) = \left[\prod_{l=1}^{k} Q_1(j,l) Q_2(j,l)\right]^{\frac{1}{2k}}$$

Finally, the topographic product is obtained by taking the logarithm and averaging over all map units and neighborhood orders:

$$\mathrm{TP} = \frac{1}{K(K-1)} \sum_{j=1}^{K} \sum_{k=1}^{K-1} \log P_3(j,k)$$

$\mathrm{TP} < 0$ indicates the map dimension is too low to correctly represent the dataset; $\mathrm{TP} = 0$ means the dimension is adequate; and $\mathrm{TP} > 0$ indicates a dimension too high and neighborhood violations. A latent topographic product $\hat{\mathrm{TP}}$ can be defined in exactly the same manner, only replacing the prototypes $\tilde{\mathbf{m}}_j$ by their latent counterparts $\mathbf{m}_j$.

Appendix 2: map visualizations

See Fig. 17.

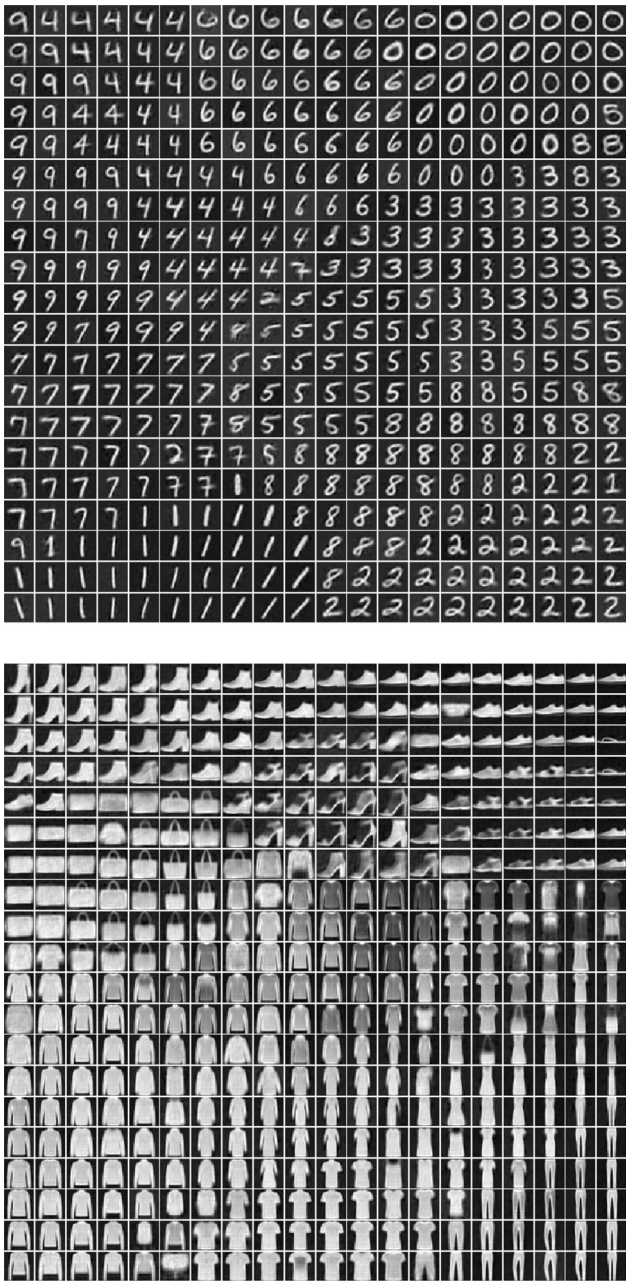

Fig. 17 20-by-20 DESOM maps of MNIST and Fashion-MNIST

Acknowledgements This research was funded by the French agency for research and technology (ANRT) through the CIFRE Grant 2017/1279 and by Safran Aircraft Engines (Safran group).

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Aljalbout E, Golkov V, Siddiqui Y, and Cremers D (2018) Clustering with Deep Learning: Taxonomy and New Methods. arXiv:1801.07648
2. Arpit D, Zhou Y, Ngo H, Govindaraju V (2016) Why regularized auto-encoders learn sparse representation?. In: International Conference on Machine Learning (ICML) 1:211–223
3. Bauer HU, Pawelzik K, Geisel T (1992) A topographic product for the optimization of self-organizing feature maps. NIPS 4:1141–1147
4. Carniel R, Jolly AD, Barbui L (2013) Analysis of phreatic events at Ruapehu volcano, New Zealand using a new SOM approach. J Volcanol Geotherm Res 254:69–79. https://doi.org/10.1016/j.jvolgeores.2012.12.026
5. Côme E, Cottrell M, Verleysen M, Lacaille J (2011) Aircraft engine fleet monitoring using Self-Organizing Maps and Edit Distance. In: International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM), pp 298–307
6. Devries T and GW Taylor (2017) Dataset augmentation in feature space. In ICLR Workshop
7. Diday E, Simon JC (1976) Clustering analysis. Springer, Berlin, Heidelberg, pp 47–94. https://doi.org/10.1007/978-3-642-96303-2_3
8. Dilokthanakul N, Mediano PA, Garnelo M, Lee MC, Salimbeni H, Arulkumaran K, and Shanahan M. (2017) Deep unsupervised clustering with Gaussian mixture variational autoencoders
9. Ghasedi Dizaji K, Herandi A, Deng C, Cai W, and Huang H (2017) Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In: ICCV, pp. 5747–5756
10. Elend L, Kramer O (2019) Self-organizing maps with convolutional layers. In: International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)
11. Fard MM, Thonet T, and Gaussier E (2018) Deep k-Means: jointly clustering with k-means and learning representations. arXiv:1806.10069
12. Faure C, Olteanu M, Bardet JM, Lacaille J (2017) Using self-organizing maps for clustering and labelling aircraft engine data phases. In: international workshop on self-organizing maps and learning vector quantization, clustering and data visualization (WSOM)
13. Ferles C, Papanikolaou Y, Naidoo KJ (2018) Denoising autoencoder self-organizing map (DASOM). Neural Netw 105:112–131. https://doi.org/10.1016/j.neunet.2018.04.016
14. Forest F, Cochard Q, Noyer C, Cabut A, Joncour M, Lacaille J, Lebbah M, and Azzag H (2020) Large-scale vibration monitoring of aircraft engines from operational data using self-organized Models. In: annual conference of the PHM society
15. Forest F, Lacaille J, Lebbah M, and Azzag H.(2018) A generic and scalable pipeline for large-scale analytics of continuous aircraft engine data. In: IEEE international conference on big data
16. Forest F, Lebbah M, Azzag H, and Lacaille J (2019) Deep architectures for joint clustering and visualization with self-organizing maps. In: PAKDD workshop on learning data representations for clustering (LDRC)
17. Florent F, Lebbah M, Hanane A, Lacaille J (2019) Deep embedded SOM: joint representation learning and self-organization. In: European symposium on artificial neural networks, computational intelligence and machine learning (ESANN)
18. Florent F, Lebbah M, Hanane A, and Lacaille J (2020) A survey and implementation of performance metrics for self-organized maps. arXiv:2011.05847
19. Fortuin V, Hüser M, Locatello F, Strathmann H, Rätsch G (2019) SOM-VAE: interpretable discrete representation learning on time series
20. Guo X, L Gao, X Liu, and J Yin (2017) Improved deep embedded clustering with local structure preservation. In: international joint conference on artificial intelligence (IJCAI), pp. 1753–1759
21. X Guo, X Liu, E Zhu, and J Yin (2017) Deep clustering with convolutional autoencoders. In: ICONIP
22. W Harchaoui, PA Mattei, A Alamansa, and C Bouveyron (2018) Wasserstein adversarial mixture clustering. https://hal.archives-ouvertes.fr/hal-01827775/
23. Hinton GE, Salakhutdinov R (2006) Reducing the dimensionality of data with neural networks. Science 313:504–507
24. Z Jiang, Y Zheng, H Tan, B Tang, and H Zhou (2017) Variational deep embedding : an unsupervised and generative approach to clustering. In: international joint conference on artificial intelligence (IJCAI), pp. 1965–1972
25. Kaski S, Lagus K (1996) Comparing self-organizing maps
26. DP. Kingma and M. Welling (2014) Stochastic gradient VB and the variational auto-encoder. In: international conference on learning representations (ICLR) arXiv:1312.6114
27. Kohonen T (1982) Self-organized formation of topologically correct feature maps. Biol Cybern 43(1):59–69
28. Kohonen Teuvo (1990) The self-organizing map. Proc IEEE 78:1464–1480
29. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
30. Lewis DD, Yang Y, Rose TG, Li F (2004) RCV1: a new benchmark collection for text categorization research. J Mach Learn Res 5:361–397
31. Liu Z, Cao J, Chen S, Lu Y, Tan F (2020) Visualization analysis of seismic facies based on deep embedded SOM. IEEE Geosci Remote Sens Lett 18(8):1491–1495
32. Q Ma, J Zheng, S Li, and GW Cottrell (2019) Learning representations for time series clustering. In: NeurIPS
33. P Madaan and A Maiti (2019) Deep mean shift clustering. PhD thesis, Indraprastha Institute of Information Technology
34. L Manduchi, M Hüser, G Rätsch, and V Fortuin (2020) DPSOM: deep probabilistic clustering with self-organizing maps. arXiv:1910.01590
35. L McInnes, J Healy, and J Melville (2018) UMAP: uniform manifold approximation and projection for dimension reduction. arXiv:1802.03426
36. HR Medeiros, PHM Braga, and HF Bassani (2020) Deep clustering self-organizing maps with relevance learning. In: ICML LatinX in AI Research Workshop
37. Min E, Guo X, Liu Q, Zhang G, Cui J, Long J (2018) A survey of clustering with deep learning: from the perspective of network architecture. IEEE Access 6:39501–39514
38. S Mukherjee, H Asnani, E Lin, and S Kannan (2019) ClusterGAN: latent space clustering in generative adversarial networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, 33:4610–4617. https://aaai.org/ojs/index.php/AAAI/article/view/4385

39. Andrew Ng (2011) Sparse autoencoder. Technical report, Stanford University, https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf
40. M Pesteie, P Abolmaesumi, and R Rohling (2018) Deep neural maps. In: ICML workshop. arXiv:1810.07291
41. DJ Rezende, S Mohamed, and D Wierstra (2014) Stochastic backpropagation and approximate inference in deep generative models. In: International Conference on Machine Learning (ICML), 4:3057–3070
42. Song C, Huang Y, Liu F, Wang Z, Wang L (2014) Deep auto-encoder based clustering. Intell Data Anal 18(6):65–76
43. Ullah A, Haydarov K, Haq IUI, Muhammad K, Rho S, Lee M, Baik SW (2020) Deep learning assisted buildings energy consumption profiling using smart meter data. Sensors 20(3):873
44. A van den Oord, O Vinyals, and K Kavukcuoglu (2017) Neural discrete representation learning. In: NIPS. arXiv:1711.00937
45. T. Villmann, M. Biehl, A. Villmann, and S. Saralajew (2017) Fusion of deep learning architectures, multilayer feedforward networks and learning vector quantizers for deep classification learning. In: international workshop on self-organizing maps and learning vector quantization, clustering and data visualization (WSOM)
46. Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol PA (2010) Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res 11:3371–3408
47. H Wu and M Flierl (2020) Vector quantization-based regularization for autoencoders. In: Proceedings of the AAAI conference on artificial intelligence, arXiv:1905.11062
48. H Xiao, K Rasul, and R Vollgraf (2017) Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747
49. J Xie, R Girshick, and A Farhadi (2016) Unsupervised deep embedding for clustering analysis. In: International conference on machine learning (ICML), vol. 48, arXiv:1511.06335
50. B Yang, X Fu, ND Sidiropoulos, and M Hong (2017) Towards K-means-friendly spaces: simultaneous deep learning and clustering. In: International Conference on Machine Learning (ICML), arXiv:1610.04794
51. D Zhu, T Han, L Zhou, X Yang, and YN Wu (2019) Deep unsupervised clustering with clustered generator model. arXiv:1911.08459

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com