SHAPING LATENT REPRESENTATIONS USING SELF-ORGANIZING MAPS WITH RELEVANCE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent work indicates that Deep Clustering (DC) methods are a viable option for unsupervised representations learning of visual features. By combining representation learning and clustering, traditional approaches have been shown to build latent representations that capture essential features of the data while preserving topological characteristics. In this sense, models based on Self-Organizing Maps models with relevance learning (SOMRL) were considered as they perform well in clustering besides being able to create a map that learns the relevance of each input dimension for each cluster, preserving the original relations and topology of the data. We hypothesize that this type of model can produce a more intuitive and disentangled representation in the latent space by promoting smoother transitions between cluster points over time. This work proposes a representation learning framework that combines a new gradient-based SOMRL model and autoencoders. The SOMRL learns the relevance weights for each input dimension of each cluster. It creates a tendency to separate the information into subspaces. To achieve this, we designed a new loss function term that weighs these learned relevances and provides an estimated unsupervised error to be used in combination with a reconstruction loss. The model is evaluated in terms of clustering performance and quality of the learned representations and then compared with start-of-the-art models, showing competitive results.

1 INTRODUCTION

Previous research has shown the potential that Deep Neural Networks (DNN) have in building representations with good generalization power. For instance, Nanni et al. (2017) showed that representations built by Convolutional Neural Networks (CNN) are better than the state-of-art handcrafted features used for object classification. Medeiros et al. (2019), also showed that the representations learned by GoogleLeNet (Szegedy et al., 2015) can be used for the task of clustering objects in images for visual object recognition, achieving about 75-90% of agreement with human assigned labels in an unseen dataset. So, these learned representations have proved to be adequate for unsupervised tasks, especially when both tasks are optimized jointly somehow (Zhang et al., 2017; Nutakki et al., 2019).

Analogously, humans have the extraordinary ability to learn many different concepts and apply such knowledge in a variety of different tasks in a lifelong setting, as it is possible to see in infants that learn how to interact with objects in their environment from correlated visual input, with minimal external supervision or with no clear specification, sequentially, and many times, without any forgetting behavior (Rao et al., 2019).

Clustering is one of the most natural ways of summarizing and organizing data. In particular, the main objective of clustering is to separate data into groups of similar data points. Although there exist multiple successful approaches for clustering data with high-level features, clustering high-dimensional unstructured data such as images, text, and sound is a hard task. Techniques based on Deep Learning (DL) have been very successful in yielding good high-level representations for such type of data (Bengio et al., 2013; Aljalbout et al., 2018).

Some of the most successful approaches for producing representations from unlabeled data are Autoencoders (AE), Variational Autoencoders (VAE), and Generative Adversarial Networks (GAN). Moreover, these techniques can be applied in different ways, such as with self-labeling (Asano et al., 2019), Deep Clustering (DC) (Nutakki et al., 2019), or contrastive learning (Caron et al., 2020). This work focuses on DC, differing from conventional approaches in terms of the algorithmic structure, network architecture, loss function, and optimization method used for training.

Current DC approaches treat representation learning and clustering as a joint task and focus on learning representations that are clustering-friendly, i.e., that preserve the prior knowledge of cluster structure. It is typically performed by optimizing a loss function ($\mathcal{L}_c$), which can be seen as a clustering loss combined with a regular neural network loss ($\mathcal{L}_n$), such as the reconstruction loss of an AE (Ji et al., 2017), or the evidence lower bound (ELBO) loss of a VAE (Jiang et al., 2016).

In this work, we particularly focus on a family of clustering algorithms called Self-Organizing Map (SOM) (Kohonen, 1990). SOM-based models have proven to be suitable for clustering high-level features, such as in Braga & Bassani (2018). In this sense, our main contributions are:

- A novel gradient-based Self-Organizing Map with Relevance Learning (SOMRL) and time-varying structure.
- A novel framework that combines Self-Organizing Map with Relevance Learning (SOMRL) with AEs to learn, shape, and cluster good latent representations.
- A thorough empirical assessment of our propositions, in both quantitative and qualitative terms on benchmark image datasets.
- Results that showed to be comparable in terms of clustering performance with its competitors, and brought novel forms of interpretation of the latent space structure, concerning its relations and transitions.

2 Related Work

The *k-means* is one of the most popular clustering algorithms. Despite being proposed over 50 years ago, it is still widely used in a diverse range of applications (Jain, 2010; Yang et al., 2017; Nutakki et al., 2019). In particular, it includes the task of clustering latent space representations of models based on AE, VAE (Aljalbout et al., 2018), fast spectral clustering with AE (Nutakki et al., 2019), and many more.

However, over the last years, models based on SOM have shown to be superior in clustering than *k-means* (Bassani & Araujo, 2015; Medeiros et al., 2019), while also providing an interpretable topological structure of the data that *k-means* can not offer. SOM (Kohonen, 1990) is a biologically inspired unsupervised learning model that maps data from a higher-dimensional input space to a lower-dimensional output space while preserving the similarities and the topological relations found between points in the input space. It creates nodes (cluster prototypes) that can be seen as abstractions of the data and a simplified way of exhibiting information.

Aljalbout et al. (2018) have shown that combining clustering algorithms working in the latent space of AE can obtain a good clustering performance. Moreover, Aljalbout et al. (2018) and Nutakki et al. (2019) provided evidence that the most successful methods for clustering with deep neural networks follow the same principle of using the representations learned by a DNN as input for a specific clustering method. They also propose a taxonomy, in which our work is mostly related to the so-called joint training. We combine an AE with a gradient-based SOM with relevance learning.

To the best of our knowledge, only two models combine AE or generative models with SOM to learn interpretable and shape latent space representations. First, the Self Organizing Map Variational Autoencoder (SOM-VAE) (Fortuin et al., 2018) combines SOM, VAE, and probabilistic model. SOM-VAE is closely related to Vector Quantised-Variational Autoencoder (VQ-VAE) (Oord et al., 2017), which can be seen as a special case of its framework, and differ in certain implementation aspects, as for instance in parts of the loss function that are set to zero. Second, Deep Embedded Self-Organizing Map (DE-SOM) (Forest et al., 2019) combines an AE with traditional SOM from Kohonen (1990) with a Gaussian neighborhood function with exponential decay.

Moreover, it is important to highlight that both methods use a classical bidimensional SOM grid with fixed topology. Fixed topology maps are very useful for data visualization. However, in complex input spaces, a 2D grid can not represent the topology adequately, especially if clusters live in different subspaces or share the same characteristics in few input dimensions. This issue has been addressed by SOM-based models that present a time-varying structure (Araujo & Rego, 2013). These models learn a topology that best shapes the input data during the training process by trying to determine an optimal arrangement at the end. This approach relies on an incremental learning process, in which not only the number of clusters is found, but also the connections between cluster prototypes must be learned.

The time-varying model proposed in this work is based on Bassani & Araujo (2015) and is presented in the next section. Afterwards, we explain how SOMRL can combined with AEs to achieve a good clustering performance as well as to extract useful information about the learned representation.

3 Self-Organizing Map with Relevance Learning (SOMRL)

SOMRL is a SOM based on Local Adaptive Receptive Field Dimension Selective Self-organizing Map (LARFDSSOM) (Bassani & Araujo, 2015). The model introduces new strategies to improve clustering quality and to allow easy integration as a layer in DNN architectures. It works as a gradient-based model that deals with batches of samples and can be used to build more intuitive latent representations and discover patterns in the latent space as well as neighborhood relations between them. To achieve this, SOMRL introduces: 1) a new node update rule to prevent the map from collapsing to trivial solutions; 2) a node removal scheme (also suitable for online learning); 3) a dynamic way of discovering neighborhood relations; and 4) an implementation focused on parallelism, that avoids sequential operations that are usually performed by SOM-based methods. Moreover, the model inherits from its predecessor a time-varying structure, a local receptive field adapted for each node as a function of its local variance, and the ability to learn different relevances for each input dimension.

3.1 Nodes Structure

Basically, as in LARFDSSOM, each node j in the map assumes the role of a cluster prototype and is associated with three m-dimensional vectors, where m is the number of input dimensions. **Center Vector:** $\boldsymbol{c}_j = \{c_{ji}, i = 1, \cdots, m\}$ represents the prototype of each cluster j.

Relevance Vector: $\boldsymbol{\omega}_j = \{\omega_{ji}, i = 1, \cdots, m\}$, in which each component is a weighting factor within [0, 1], that represents the relevance that the node j applies to the i-th input dimension.

Distance Vector: $\boldsymbol{\delta}_j = \{\delta_{ji}, i = 1, \cdots, m\}$ stores a moving average of the observed absolute distance between the input patterns and the center vector. It is used only to update the relevance vector.

3.2 Competition

In SOMRL the nodes in the map compete to cluster the input patterns. Whenever a batch of samples is presented to the map, the activation, $ac(D_\omega(\boldsymbol{x}, \boldsymbol{c}_j), \boldsymbol{\omega}_j)$, of each node j to each sample $\boldsymbol{x}$ is computed as a radial basis function of a weighted distance $D_\omega(\boldsymbol{x}, \boldsymbol{c}_j)$ with the receptive field adjusted as a function of $\boldsymbol{\omega}_j$ (equation 1). The winner node for each sample is the most active, where $D_\omega(\boldsymbol{x}, \boldsymbol{c}_j)$ is a learned distance metric, weighted by the relevance vector $\boldsymbol{\omega}_j$ equation 2.

$$ac(D_\omega(\boldsymbol{x}, \boldsymbol{c}_j), \boldsymbol{\omega}_j) = \frac{\sum\limits_{i=1}^{m} \omega_{ji}}{\sum\limits_{i=1}^{m} \omega_{ji} + D_\omega(\boldsymbol{x}, \boldsymbol{c}_j) + \epsilon}. \tag{1}$$

$$D_\omega(\boldsymbol{x}, \boldsymbol{c}_j) = \sum_{i=1}^{m} \omega_{ji}(x_i - w_{ji})^2. \tag{2}$$

SOMRL finds the winner node and the activation *intensities* of each node in the map. These *intensities* also play an important role in the definition and update of the neighbors.

3.3 Node Insertion and Update

In SOMRL, the samples (assumed to be in $[\mathbf{0}, \mathbf{1}]$ interval) are processed in batches of arbitrary sizes. After the competition, the samples of the batch associated with each winner node are grouped, and the average position of the group is computed. This average is then used as an input pattern $\boldsymbol{x}$ to update the map as follows:

First. For samples that the most active node presents an activation below a certain threshold parameter at, the model creates a new node j with its center $\boldsymbol{c}_j$ initialized as $\boldsymbol{x}$, the relevance vector $\boldsymbol{\omega}_j$ initialized with ones, and the distance vector $\boldsymbol{\delta}_j$ with zeros.

Second. Winner nodes and neighbors with activation above at are updated towards $\boldsymbol{x}$, as per equation 3.

$$\boldsymbol{c}_j(n+1) = \boldsymbol{c}_j(n) + lr[(\boldsymbol{x} - \boldsymbol{c}_j(n)) - \rho \text{ repel}(\overline{\boldsymbol{c}}(n) - \boldsymbol{c}_j(n))], \tag{3}$$

where lr is the learning rate which is maximum for the winner and smaller for the neighbors (see Section 3.5), $\overline{\boldsymbol{c}}$ is the geometric center (average) of the nodes in the map, and $\rho \in [0, 1]$ is an update rate that controls how the node will be repelled from $\overline{\boldsymbol{c}}$, according to a repel function equation 4. This term prevents the map from collapsing to a trivial solution without limiting too much the exploration capabilities since, after a certain distance, this factor is zero. Note that equation 4 behaves like an inverted ReLu, and $\max$ is applied to each dimension.

$$\text{repel}(\mathbf{d}) = \max(\mathbf{1} - \mathbf{2} * \mathbf{d}, \mathbf{0}). \tag{4}$$

Third. In order to update the relevance vector $\boldsymbol{\delta}_j$, first, the average distance of each node to the input patterns it clusters is estimated. This is done by computing a moving average of the observed distance between the input pattern and the current center vector:

$$\boldsymbol{\delta}_j(n+1) = (1 - \beta * lr)\boldsymbol{\delta}_j(n) + (\beta * lr)|\boldsymbol{x} - \boldsymbol{c}_j(n+1)|, \tag{5}$$

where lr is the learning rate used in equation 3, $\beta \in]0, 1]$ controls the rate of change of the moving average, and the operator $|\cdot|$ denotes the absolute value applied to the components of the vector.

Fourth. Each component ω_{ji} of the relevance vector is calculated by an inverse logistic function of the distances δ_{ji}, as per equation 6.

$$\omega_{ji} = \begin{cases} \frac{1}{1+\exp\left(\frac{\hat{\delta}_{ji\text{mean}} - \hat{\delta}_{ji}}{s(\hat{\delta}_{ji\text{max}} - \hat{\delta}_{ji\text{min}})}\right)} & \text{if } \hat{\delta}_{ji\text{min}} \neq \hat{\delta}_{ji\text{max}} \\ 1 & \text{otherwise,} \end{cases} \tag{6}$$

where $s > 0$ controls the slope of the logistic function. The relevances go to zero for dimensions with distances close to the maximum $\delta_{ji\text{max}}$, whereas in the other dimensions, they are set within [0, 1].

3.4 Node Removal

In SOMRL, a concept of life is introduced to each node. They begin at 100% and lose their vitality by a parameter factor $ld \in [0\%, 100\%]$ every time they do not win a competition, i.e., present an activation that is not the highest. Therefore, whenever a node achieves a life value of 0, it is removed from the map. Alternatively, the life of a node is restored to 100% when it wins a competition.

3.5 Neighborhood

The neighborhood of SOMRL is not fixed but defined dynamically at each update step as a function of the nodes activation rank, from the highest to the lowest. Therefore the neighbors of the winner node are also updated towards the input pattern, but with smaller learning rates since it decays exponentially as a function of the position of the neighbor in the rank, as $h(r) = exp(-\frac{r}{\gamma})$, where r is the rank, and γ a parameter that controls the decay rate.

Before updating the nodes, the final learning rate is computed by $lr * h(r)$. Notice that the most active node will be fully updated according to lr since $h(r)$ is 1 for rank 0 and decays as the rank increases and activation decreases. Therefore, as the neighborhood of SOMRL is not related to the nodes themselves but to the input patterns presented to the model, a sample-driven approach.

3.6 Connecting the Pieces: SOMRL Algorithm

The SOMRL training procedure is described in Algorithm 1. Note that the variable n is used to control the current number of nodes in the map, and n_{max} defines its limit. It is used solely for implementation purposes, once we have to define a fixed shape for the matrices and vectors to avoid concatenation operations, that leads to a loss of performance. The operations to control the active nodes is performed by using logical masks. We refer to it in Appendix B.

At the end of the training, we compute the activations among the remaining nodes to define connection and relations between them, and, thus, the final topology of the map. The nodes that are mutually activated, above a_t threshold, are connected.

Algorithm 1: SOMRL Forward Pass

```
Input : Batch of Input Patterns $\boldsymbol{x}$
Require : Initialize parameters $a_t$, $lr$, $\rho$, $ld$, $\beta$, $s$, $n_{max}$
Function SOMRL($\boldsymbol{x}$):
    if map is empty then
        Initialize the map with one node with $\boldsymbol{c}_j$ initialized at $\overline{\boldsymbol{x}}$, $\boldsymbol{\omega}_j \leftarrow \mathbf{1}$, $\boldsymbol{\delta}_j \leftarrow \mathbf{0}$, life$_j \leftarrow 1$; Set the number of nodes $n \leftarrow 1$;
    else
        Compute the activations $\mathbf{A}$ of all nodes for each $\boldsymbol{x}_i \in \boldsymbol{x}$ equation 1;
        Find the winners $s_i$ with the highest activation for each $\boldsymbol{x}_i \in \boldsymbol{x}$;
        forall $s_i \in \boldsymbol{s} \mid A_{i,s_i} < a_t$ and $n < n_{max}$ do
            Create a new node $j$ and set: $\boldsymbol{c}_j \leftarrow \overline{\boldsymbol{x}}_{s_i}$, $\boldsymbol{\omega}_j \leftarrow \mathbf{1}$, $\boldsymbol{\delta}_j \leftarrow \mathbf{0}$, life$_j \leftarrow 1$;
            Set $n \leftarrow n + 1$;
        forall $s_i \in \boldsymbol{s} \mid A_{i,s_i} \geq a_t$ do
            Compute the learning rates $\boldsymbol{lr_n}$ of the neighbors by ranking $\mathbf{A}_i$ (Section 3.5);
            Update the winner nodes and its neighbors towards $\overline{\boldsymbol{x}}_{s_i}$ with $lr$ and $\boldsymbol{lr_n}$, respectively:
            - Update the distance vectors $\boldsymbol{\delta}_{s_i}$ and $\boldsymbol{\delta_n}$ equation 5;
            - Update the relevance vectors $\boldsymbol{\omega}_{s_i}$ and $\boldsymbol{\omega_n}$ equation 6;
            - Update the center vectors $\boldsymbol{c}_{s_i}$ and $\boldsymbol{c_n}$ equation 3;
            Decrement the life of all nodes $j \notin \boldsymbol{s}$ to life$_j \leftarrow$ life$_j - ld$;
        Remove all nodes $j$ with life$_j$ < 0; Decrement $n$ according to number of removed nodes;
```

4 Deep Clustering SOMRL (DC-SOMRL): Complete Framework

Our proposed framework[1], combines AE with our new gradient-based SOM with relevance learning and time-varying structure, SOMRL. A schematic overview of this arrangement is presented in figure 1. It not otherwise noted, the AE is architecture is based on Xie et al. (2016).

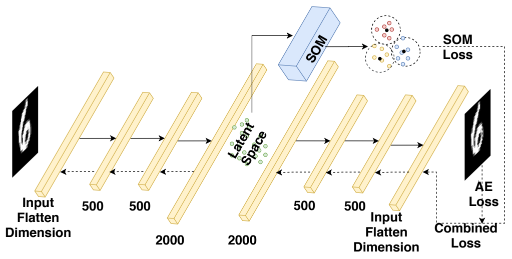


Figure 1: Combined Architecture Proposed

Lets denote the encoder and decoder parameters as θ_e and θ_d, respectively. An input $\boldsymbol{x} \in \mathbb{R}^d$ is mapped to a latent space encoding $\boldsymbol{z}_e \in \mathbb{R}^m$. Bassani & Araujo (2015) have shown that it is easier to adjust the parameters of SOM when the input dimensions are scaled to the [0, 1] interval. To maintain this behavior, $\boldsymbol{z}_e$ is computed by $\boldsymbol{z}_e = sigmoid(f_{\theta_e}(\boldsymbol{x}))$. The encoding is then fed as input to a forward pass of SOMRL and a decoder pass of the AE architecture. At this point, a reconstruction $\hat{x}$ of the input can be computed as $\hat{\boldsymbol{x}} = g_{\theta_d}(\boldsymbol{z}_e)$. This component is used to calculate the reconstruction

[1]Complete source code provided as supplementary material. It will be made public after acceptance.

loss $\mathcal{L}_{\mathrm{r}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2$. It is well-know that AE can minimize the reconstruction error ensuring that hidden units capture the most relevant aspects of the data (Murphy, 2012).

In SOMRL, a competition establhishes the winner node as the most active for the latent representation in $\boldsymbol{z}_e$, when it is fed as input, as per equation 1. Then, the winner and its neighbors move towards the input by updating the center vectors, and the new relevance vectors are estimated. Notice that when there is no node sufficiently activated, according to a threshold parameter, SOMRL will create and insert a new node into the map at the position of the input pattern. At the end of this process, the winner prototype for $\boldsymbol{z}_e$, $\boldsymbol{c}_z$ is used to compute the clustering loss, defined as $\mathcal{L}_c(\boldsymbol{z}_e) = D_\omega(\boldsymbol{z}_e, \boldsymbol{c}_z)$, that is the distance weighted by the relevances found by SOMRL equation 2. It is expected that $\mathcal{L}_c$ creates a tendency to approximate encodings to prototypes. Due to the latent space constraints (all dimensions between [0, 1]), and the fast convergence characteristics of SOMRL, even with a small number of samples, $\mathcal{L}_c$ will frequently converge to small values in few epochs.

To achieve the main challenge of optimizing our complete framework, but more precisely the representations (prototypes) learned by our SOMRL, the AE is learned jointly with the SOMRL. To do so, we add the $\mathcal{L}_c$, weighted by an α, to the $\mathcal{L}_{\mathrm{r}}$ in order to add the gradient information about $\boldsymbol{z}_e$ with respect to the SOMRL state. The complete loss function to be minimized is given as follows:

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}, \mathbf{z}_e) = \mathcal{L}_{\mathrm{r}}(\mathbf{x}, \hat{\mathbf{x}}) + \alpha \mathcal{L}_c(\mathbf{z}_e). \tag{7}$$

5 Experiments

We conducted experiments on MNIST handwritten digits (LeCun et al., 1998) and Fashion-MNIST article images (Xiao et al., 2017). For all experiments, the same architecture was used, and the results were evaluated from both quantitative and qualitative perspectives. For implementation details and further information, we refer to Appendix B. We present an extensive methodology to illustrate how we achieved our results and to allow a better picture of our final remarks.

5.1 Adjusting the Hyperparameters of the Model

We begin by analyzing how our method behaves and how each of its hyperparameters impacts the outcome. To do so, we conducted an experiment varying the values within a defined range and sampling them according to a Latin Hypercube Sampling (LHS) (Helton et al., 2005). It is a statistical method for generating a random sample of values from a multidimensional distribution. In this sense, we gathered 30 different parameter settings, i.e., the range of each parameter was divided into 30 intervals of equal probability, resulting in a random selection of a single value from each interval. LHS ensures that each component is represented in a fully stratified manner, no matter the importance that it might have. The ranges used for our method can be found in Appendix A.1.

The results of the experiment in terms of Purity and Normalized Mutual Information (NMI) were used to define the best parameter set over these 30 runs for both datasets (we refer to Appendix A.1 for detailed information). Then, it is fixed for the rest of the experiments. We quantitatively compare our model against k-means (Lloyd, 1982), SOM-VAE (Fortuin et al., 2018), and DE-SOM (Forest et al., 2019). Similar approaches were conducted to the other models to control their parameters.

5.2 Clustering Performance

After adjusting the parameters of the model, we run a comparison with the other models on the standard MNIST and Fashion-MNIST test sets. The results are shown in Table 1. We found that our method achieves competitive results concerning its competitors. It is important to mention that we do not aim at outperforming other models in terms of metric value, instead build robust representations with meaningful properties, which will be further explored qualitatively. We highlight that all compared models used the same number of clusters. However, due to the time-varying feature of our SOMRL, we can not specify any number of clusters. Note that the parameter n_{max} is only used for implementation purposes, as argued in Appendix B.

As discussed in Appendix C, NMI is a more balanced measure for clustering performance than purity due to the penalty term for the number of clusters. Purity may lead to a scenario in which results

[2]The authors did not provide standard deviation and the public source code was not possible to be ran

Table 1: Clustering performance of SOMRL and some baselines on MNIST and Fashion-MNIST in terms of Purity and NMI. The values are the means of 10 runs ± the respective standard deviations. Each method used 64 embeddings/clusters, except SOMRL, due to its time-varying structure.

Method	MNIST		Fashion-MNIST	
	Purity	NMI	Purity	NMI
k-means	0.791 ± 0.005	0.537 ± 0.001	0.703 ± 0.002	0.492 ± 0.001
SOM-VAE	0.868 ± 0.003	0.595 ± 0.002	0.739 ± 0.002	0.520 ± 0.002
DE-SOM[2]	0.939	0.657	0.752	0.538
SOMRL	0.884 ± 0.014	0.521 ± 0.007	0.679 ± 0.009	0.404 ± 0.005

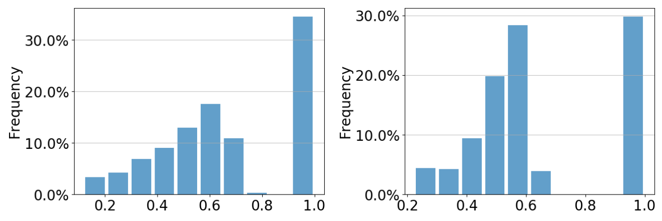
that present a high number of clusters are rewarded in detriment of more meaningful representations. Moreover, other models are originally trained over 10,000 epochs. However, DC-SOMRL is consistently able to achieve the reported results in less than 30 epochs (Appendix A.2).

5.3 Latent Representations

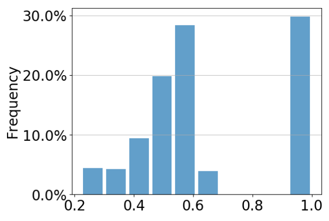
In order to assess whether our model can learn interpretable representations, we analyse the relevances learned by the model. Note that SOMRL is able to find samples that may belong to more than one cluster. It is a consequence of taking into account different subsets of the input dimensions, according to their relevances for what the prototype is trying to represent. A good interpretable behavior happens when the prototype does not represent the raw class itself, but specific characteristics. So, their composition may form the respective class. We expect that in these situations, the relevances of the important features may become high, whereas the irrelevant ones become low. In this case, we hypothesize that changes to these irrelevant features will not degrade the main characteristics of the prototype when it is decoded. Analogously, when an important feature is changed, we expect that the prototype loses its properties, and a mischaracterization occurs.

5.3.1 Learned Relevances

In an effort towards understanding the relevances learned by the model, we analyse the distribution of relevance values found among all prototypes. figure 2 presents this information. Note that in this particular case, the model considered 35% of the dimensions as highly important (figure 2(a), values close to 1.0). Fashion-MNIST (figure 2(b)) presented 30% of highly important dimensions, and 30% of moderately important dimensions.



(a) MNIST



(b) Fashion-MNIST

Figure 2: Histograms of relevances (relevance values on the x-axes) for MNIST and Fashion-MNIST.

5.3.2 Effect of the Relevances

In order to demonstrate the importance of the relevant and irrelevant dimension to the reconstructions, we start from an example prototype (figure 3(a)) and disturb the dimensions with low relevance values, such as <0.3. In this case, we expect that changes in these dimensions of the latent space will not cause a strong impact on the reconstructed images. figure 3(b) shows that the prototype retains its main characteristics. If we expand the perturbation to more relevant dimensions, such as <0.4 (figure 3(c)), we start seeing an impact on the reconstruction. If we push this threshold to dimensions with relevance <0.5 (figure 3(d)), the effect is evident. It degrades, as we expected to see, though it still is recognizable as the digit 4. In opposition, if we disturb the most relevant dimensions for this particular node (>0.6) the degradation takes place in a sense the node loses its original characteristic (figure 3(e)). So, the relevance values found by the model are indeed meaningful. Intuitively, the contrary statement also showed to be true. Low values of relevance means, in fact, unimportant dimensions for the prototype at hand.

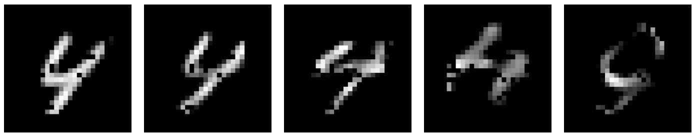

(a) Prototype (b) < 0.3 (c) < 0.4 (d) < 0.5 (e) > 0.6

Figure 3: The decoded prototype on the left (a). Perturbation in dimensions with relevances values less then 0.3 (b), 0.4 (c), and 0.5(d) is inserted. For small values, the prototype does not lose its characteristics. However, when the relevance value increases, the prototype starts to degrade. When only relevant dimensions are changed (e), the node loses its original characteristic.

If we further explore the highest relevances, it is possible to highlight some interesting behaviors. figure 4 shows the pictures generated by decoding the prototype after varying a specific relevant dimension on the latent space. In this particular case, the learned factors relate to an important characteristic that differentiate the numbers 0 and 8, which is the constraint in the middle. If we decrease, it becomes a zero, if we increase, it becomes an eight, in between it is similar to a 3. The proximity between these categories can also be observed in figure 5(a). So, the model succeeds in representing specific characteristics of images on its relevant dimensions.

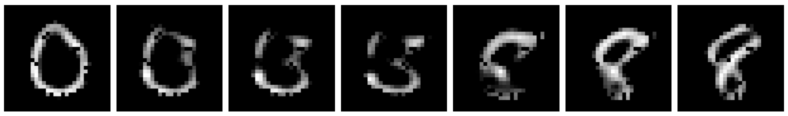
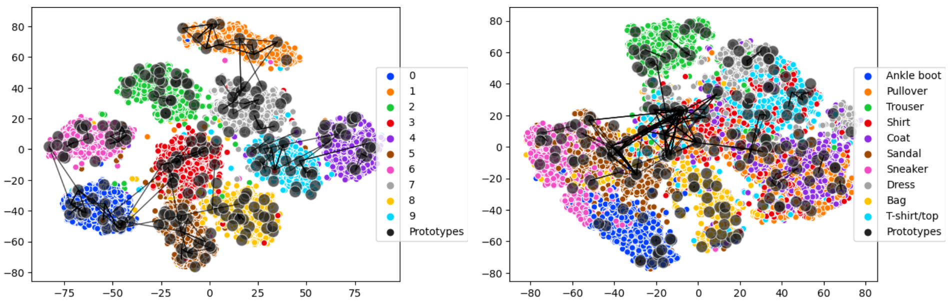


Figure 4: Latent factors learned on MNIST: Transition from 0 to 3, and to 8 obtained by changing only relevant features in the latent space.

We also discussed in Appendix A.3 some compositionality characteristics of our model. Some clusters combine their characteristics to define certain classes. We fully explored those behaviors of the union of clusters to assess what they form when posed together. We also analyzed the intersection between clusters. For instance, what is the intersection between an zero and an eight? Our model was able to find these relations in a very comprehensive manner.

5.4 Neighborhood and Topological Relations



(a) MNIST Test Data (b) Fashion-MNIST Test Data

Figure 5: t-SNE for MNIST and Fashion-MNIST Test Data alongside decoded SOMRL Prototypes.

To illustrate the topological structure in the latent space, we present the t-SNE projection (Maaten & Hinton, 2008) of the encoded samples alongside the prototypes found by the model (figure 5). t-SNE was chosen due to the fact that our map does not present a 2D disposal grid, as the related competitors. The edges between prototypes represent topological neighborhood found, i.e. connected prototypes are considered somehow similar by the model. Notice that the model was able to create at least one cluster for each class, connecting more densely prototypes of a same class, and the connections between nodes in regions of different classes usually make sense in a semantic perspective (e.g., the number 9 shares similarities with 4 and 7, number 7 with 1, relations between Sandal and Sneaker, Pullover and Coat, and so on). These interesting results allows to observe characteristics of prototypes found and features shared by prototypes of different categories.

6 Conclusion

In this article, we present a model combining AE with a customized SOM (SOMRL). Our complete framework shows a novel way of learning a time-varying structure map with relevance learning and a dynamic topology that is more suitable for the problem of clustering features in evolving latent spaces. Although in the quantitative analysis (Purity and NMI) it did not present the best results, it is yet competitive with previous models and provides additional interesting characteristics.

The prototypes identified represent variations frequently observed in the input data. For instance, the different ways of writing a digit. It can also bring insights about similarities between different categories or feature representations and which dimensions of the latent space capture then in relation to the prototypical representations found. Also, the learned neighborhood leads to smoother regions of transition between categories in the latent space.

An analysis of the impact of the loss produced by the model to the obtained latent space is important for future work as well as finding better ways to guide the latent space. We believe, though, that this was an important step in this direction. Therefore, we consider the proposed framework as a useful tool for promoting the formation of more meaningful representations as well as analyzing them.

References

Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, Maximilian Strobel, and Daniel Cremers. Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*, 2018.

Aluizio FR Araujo and Renata LME Rego. Self-organizing maps with a time-varying structure. *ACM Computing Surveys (CSUR)*, 46(1):1–38, 2013.

Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. Self-labelling via simultaneous clustering and representation learning. *arXiv preprint arXiv:1911.05371*, 2019.

Hansenclever F Bassani and Aluizio FR Araujo. Dimension selective self-organizing maps with time-varying structure for subspace and projected clustering. *IEEE transactions on neural networks and learning systems*, 26(3):458–471, 2015.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Pedro HM Braga and Hansenclever F Bassani. A semi-supervised self-organizing map for clustering and classification. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2018.

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *arXiv preprint arXiv:2006.09882*, 2020.

Florent Forest, Mustapha Lebbah, Hanane Azzag, and Jérôme Lacaille. Deep architectures for joint clustering and visualization with self-organizing maps. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 105–116. Springer, 2019.

Vincent Fortuin, Matthias Hüser, Francesco Locatello, Heiko Strathmann, and Gunnar Rätsch. Som-vae: Interpretable discrete representation learning on time series. *arXiv preprint arXiv:1806.02199*, 2018.

Jon C Helton, FJ Davis, and Jay D Johnson. A comparison of uncertainty and sensitivity analysis results obtained with random and latin hypercube sampling. *Reliability Engineering & System Safety*, 89(3):305–330, 2005.

Ronald L Iman and Jon C Helton. An investigation of uncertainty and sensitivity analysis techniques for computer models. *Risk Analysis*, 8(1):71–90, 1988.

Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

Pan Ji, Tong Zhang, Hongdong Li, Mathieu Salzmann, and Ian Reid. Deep subspace clustering networks. In *Advances in Neural Information Processing Systems*, pp. 24–33, 2017.

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

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

Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2): 129–137, 1982.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Heitor R Medeiros, Felipe DB de Oliveira, Hansenclever F Bassani, and Aluizio FR Araujo. Dynamic topology and relevance learning som-based algorithm for image clustering tasks. *Computer Vision and Image Understanding*, 179:19–30, 2019.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Loris Nanni, Stefano Ghidoni, and Sheryl Brahnam. Handcrafted vs. non-handcrafted features for computer vision classification. *Pattern Recognition*, 71:158–172, 2017.

Gopi Chand Nutakki, Behnoush Abdollahi, Wenlong Sun, and Olfa Nasraoui. An introduction to deep clustering. In *Clustering Methods for Big Data Analytics*, pp. 73–89. Springer, 2019.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pp. 6306–6315, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. Continual unsupervised representation learning. In *Advances in Neural Information Processing Systems*, pp. 7645–7655, 2019.

Andrea Saltelli, Karen Chan, E Marian Scott, et al. *Sensitivity analysis*, volume 1. Wiley New York, 2000.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pp. 478–487, 2016.

Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3861–3870. JMLR. org, 2017.

Dejiao Zhang, Yifan Sun, Brian Eriksson, and Laura Balzano. Deep unsupervised clustering using mixture of autoencoders. *arXiv preprint arXiv:1712.07788*, 2017.

APPENDIX

A ADDITIONAL EXPERIMENTS

For all the experiments, the models were trained exclusively on the training set of MNIST and Fashion-MNIST, and evaluated on the test set of each one.

Moreover, the MNIST dataset was normalized according to the mean and standard deviation of the training set. After this pre-processing step, samples become centered in zero with a standard deviation of one. For Fashion-MNIST, we employed the common normalization of using 0.5 for mean and standard deviation.

A.1 PARAMETER ANALYSIS

First, it is important to start discussing the meaning of each parameter of the model. They are given as follows:

Batch Size, **Epochs**, and **SOM learning rate** (lr), by intuition, do not need further explanation. For fair comparisons with state of the art competitors, we fixed the batch size to 256. Moreover, the number of epochs was defined at first hand to range from 10 to 1,000. Nonetheless, an extensive empirical analysis showed that 30 epochs are more than enough for the model to stabilizes. For instance, our competitors used 10.000 epochs on their experimental setup. We refer to Appendix A.2 for further discussion.

Clustering Loss weighting factor (α). It is the weighting factor that penalizes the clustering loss. It is an empirical value that is used to control the influence (or contribution) of the clustering loss to the final loss.

SOMRL input dimensions. It is the number of dimensions given as input to the model. It defines the dimensionality of the latent space after the input is encoded.

Activation threshold (a_t). Activation threshold. During training, if the activation of the winner node is below this level, a new node is inserted to define a new cluster.

Repel rate (ρ). Controls the negative push rate $w.r.t.$ the geometric center of the map. It is employed when updating the center vectors equation 3 and avoids the model from collapsing to a trivial solution. Moreover, this term facilitates the nodes to be safely distant from each other.

Life decay (ld). It is the life decreasing rate that is applied when a node does not win a competition. Whenever the life of a node reaches zero, the node is removed from the map.

Relevance rate (β). Rate of change of the moving average used to compute the relevance vector. Higher values make the nodes to adapt faster to the relevant dimensions. Too high values provoke instability. Lower values produce a smoother adaptation.

Neighborhood decay (γ). This parameter controls how strong will be the update of neighbors towards the input pattern. The winners are fully updated according to the lr, but the neighbors are updated with a lower factor, according to their activation. Distant nodes have a low activation and are barely influenced.

Relevance smoothness (s). It represents the slope of the logistic function when calculating relevances. Values close to zero, produce a sharp slope. As the value increases, the slope becomes less prominent and all relevances tend to be similar. Values higher than 1.0 result in similar relevances values around 0.5 for all dimensions.

Max number of nodes (n_{max}). This value should be higher than the number of expected clusters in the dataset to allow exploration of the search space. It also may be used to prevents trivial solutions, suchlike one cluster for each data point, and to control memory usage. It is important to mention that this parameter, when other parameters are set correctly it should not influence significantly the outcome due to the time-varying topology of the model that inserts and removes nodes when it is necessary during the self-organizing process.

Sampling Parameters. Table 2 shows the range of parameters used for studies. The values were sampled according to a LHS (Helton et al., 2005), as mentioned in Section 5.1. For initial ranges, the model was executed 30 times 30 different parameter settings. For adjusted ranges, the model was run 10 times with 10 different sets of parameters. Finally, for the final round of experiments, we fixed the best values varying the seed over 10 runs to extract the mean and standard deviation.

Table 2: Parameter values starting from initial ranges, then an intermediate step of Adjusted Ranges, and finally, the best values for each dataset.

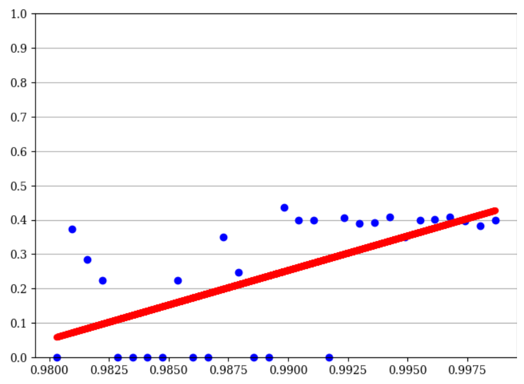
Parameters	Initial Ranges		Adjusted Ranges		**Best Values**	
	min	**max**	**min**	**max**	**MNIST**	**Fashion-MNIST**
Batch Size	16	512	256	-	256	256
Epochs	10	1,000	30	-	30	30
$\mathcal{L}_c$ weighting factor (α)	0.01	1.00	0.01	0.15	0.1145	0.024
SOM input dimensions	10	50	20	30	22	22
Activation threshold (a_t)	0.95	0.999	0.99	0.999	0.99425	0.99885
SOM learning rate (lr)	0.001	0.3	0.05	0.15	0.1426	0.1383
Repel rate (ρ)	0.0	1.0	0.5	1.0	1.0	0.575
Life decay (ld)	0.01	0.6	0.1	0.5	0.36	0.46
Relevance rate (β)	0.01	0.5	01	0.15	0.067	0.0195
Neighborhood decay (γ)	0.17	7.0	2.0	6.0	1.382	4.916
Relevance smoothness (s)	0.01	0.7	0.4	0.7	0.426	0.658
Max number of nodes (n_{max})	20	300	50	250	200	200

After the initial runs, a sensitivity analysis was performed, as in Iman & Helton (1988) and Saltelli et al. (2000). They both show that given the probabilistic basis of LHS, it can provide direct estimates for the Cumulative Distribution Function (CDF) and variance of models. It is done by pairing the obtained results measure and the parameter values sampled from the LHS distribution used to obtain each of the results. Also, it is possible to use this in combination with a linear regression model to get intuitions about the trends of the obtained results as a function of the parameter value used. Such an analysis can be done for each parameter in order to draw a better understanding of their impact and influence on the models.

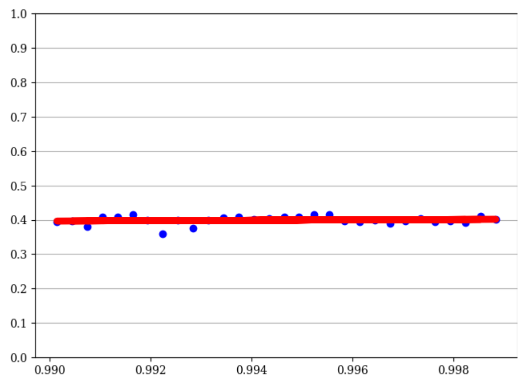
The sensitivity analysis showed that only a_t presented a high impact on the results. Remark that a_t is an exponential parameter, in which even smaller changes outside the ranges can cause instability. It depends on the dataset. All the other parameters presented marginal impacts on the outcome. figure 6 shows its behavior for both FashionMNIST and MNIST in terms of NMI. figure 6(a) and figure 6(c) shows the intervals of a_t between 0.95 and 0.999, and between 0.98 and 0.999, respectively. It shows a great trend to higher values. So, if we adjust the values to the proper ranges, i.e., from 0.99 to 0.99, the performance stabilizes. Note that for the second run, after the adjustments, DC-SOMRL was executed only 10 times with 10 different parameters. Finally, it is possible to select the best parameters as fixed values, as given by Table 2, to perform further comparisons with other models. This behavior is exactly the same for the Purity metric. The model usually ends with a number of nodes that vary from 60 up to 80.

A.2 Fast Convergence

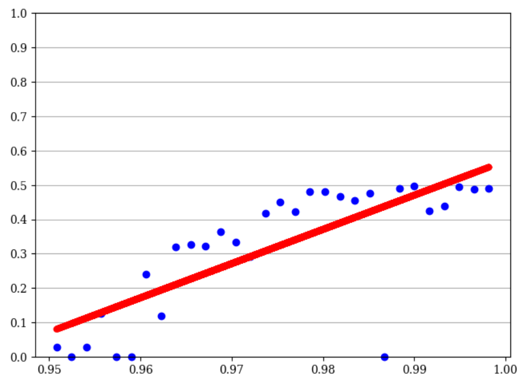
DC-SOMRL showed to be capable of achieving good results very rapidly. In each DC-SOMRL forward step, we analyzed the Purity and NMI metrics over validation data. It is composed of 10 samples of each class on the test set. The purpose is to analyze the performance of the model without any compromise nor significant add of time to the execution of the models. It is not used during the training. It also allows us to see convergence trends at the same time that gives some insights about how we could adjust some parameters and provides a better picture of what is happening during the learning process. In figure 7, the visualization of the mentioned metrics per each step of the model is illustrated. Notice that good metrics values are reached at an early stage of training and do not degrade afterwards.



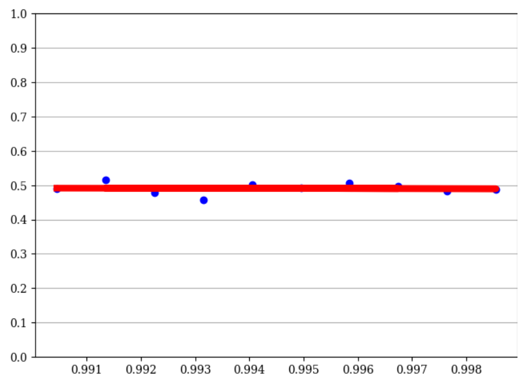
(a) a_t [0.98, 0.999]: Fashion-MNIST



(b) a_t [0.99, 0.999]: Fashion-MNIST



(c) a_t [0.95, 0.999]: MNIST



(d) a_t [0.99, 0.999]: MNIST

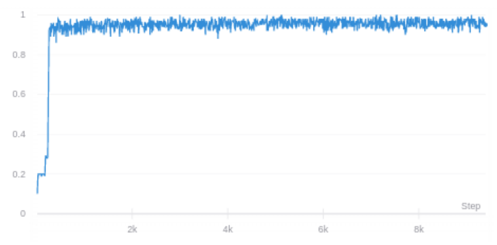
Figure 6: Scatter plots of the NMI obtained with SOMRL as a function of its parameter a_t for the datasets on MNIST and Fashion-MNIST varying the range from [0.95, 0.999] to [0.99, 0.999], and from [0.98, 0.999] to [0.99, 0.999], respectively

A.3 Undestanding Prototypes

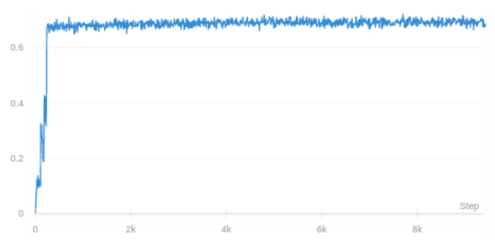
To illustrate some interpretations that DC-SOMRL has learned w.r.t. input classes, we implemented a function for the intersection of clusters. First, the prototypes in the latent space are decoded. Then, a confusion matrix is computed to help us at identifying clusters that may represent the same classes. After that, some of them are manually selected for the study. For instance, figure 8 and figure 9, show the intersection of clusters representing the number 0 and the number 4, respectively. It is possible to observe that our prototypes identify the number 0 basically by the style of rounded curves at the top and bottom of the number. So, the model created abstractions of different styles of writing a 0 in different prototypes. In figure 9, a similar behavior is shown. However, distinguished interpretations of a number 4 are given by the style of the middle line. The importance of such an experiment implies on the fact we can visualize what the model is learning and why sometimes it creates more clusters than expected to represent the correspondent manifold of a class in the latent space. To address this process of sub clustering or to analyze the power cluster, we could utilize a k-means as a post-processing step to cluster similar prototypes, or add a pruning phase after the self-organization step.

A.4 Prototypes and Closest Samples

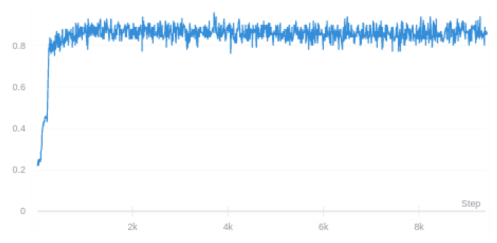
We consider that a qualitative analysis of the similarities between the prototypes created during training phase of DC-SOMRL, and the samples from the test dataset (never seen before) could also instigate good interpretations. This experiment demonstrates the capacity of the model to create prototypes close to the mean of the samples distribution. To this extent, the euclidean distances between decoded prototypes and input patterns were computed and the 10 closest samples for



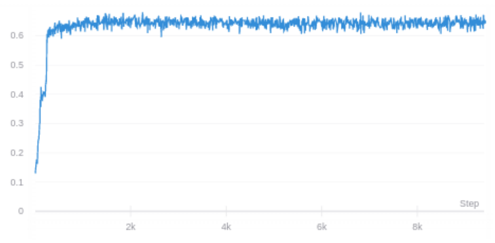
(a) MNIST: Purity x DC-SOMRL



(b) MNIST NMI x DC-SOMRL Forward step



(c) Fashion-MNIST: Purity x DC-SOMRL



(d) Fashion-MNIST: NMI x DC-SOMRL

Figure 7: Plots of the Purity and the NMI obtained with the DC-SOMRL over each batch forward step for the validation datasets on MNIST and Fashion-MNIST.

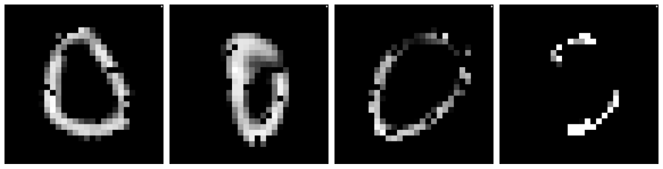

(a) Prototype 3 (b) Prototype 49 (c) Prototype 9 (d) Intersection

Figure 8: Reconstruction of prototypes representing the number 0 and their intersection.

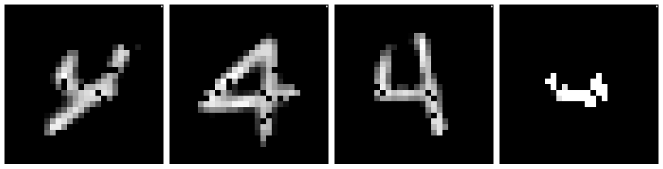

(a) Prototype 36 (b) Prototype 44 (c) Prototype 51 (d) Intersection

Figure 9: Prototypes representing the number 4 and their intersection.

each prototype were selected. figure 10 illustrates the outcome. On the left column, four different prototypes, two representations of the number 2, and two representations of the number 2 and two representing the clusters of number 7. On the right, the ten closest samples of each corresponding prototype.

Now, it is possible to expand this interpretation and see in figure 10 two different ways of writing the numbers 2 and 7, respectively.

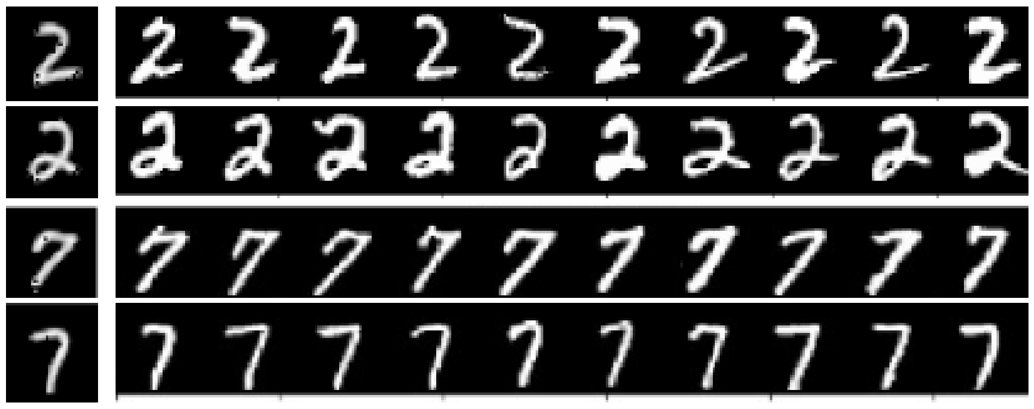


Figure 10: On the left, a column with the prototypes. On the right, the top ten samples closest to the prototypes.

B Implementation Details

In addition to the considerations from Section 3.6 and Section A, the model was implemented in PyTorch[3] (Paszke et al., 2019), and the AE used in DC-SOMRL was optimized using Adam (Kingma & Ba, 2014).

The implementation was focused entirely on optimization. Traditional implementations of SOM performs sequential steps over loops. We avoided these situations by using logical masks of tensors and matrix operations. The masks are also used to control the active nodes in the map and its related parameters (PyTorch Tensors — vector and matrices) that we must operate at each forward pass. The code became a bit harder to interpret but improved significantly in terms of performance. So, we could take compelling advantages of Graphics Processing Units (GPU) in a completely new fashion for a SOM model. We also extensively used Weights & Biases (Biewald, 2020) when executing our experiments.

For external competitors, we used the hyperparameters and code from their respective publication, when applicable. For those in which the code was not possible to be executed due to technical issues, we referred to the results presented in the respective papers.

C Performance Measures

Given the nature of our task, we need an evaluation measurement that reflects the real quality of our model. For this extent, we decided to use two extensively used metrics in the literature: Purity and NMI. It is important to emphasize that if any of these two metrics are used in isolation, it will lack a real evaluation of performance. For instance, Purity has a very simple interpretation. It has some shortcomings and may mislead a realistic evaluation. A high-value of Purity is not meaningful information. If the models converge to a trivial solution by finding one cluster for each input pattern, it will yield 1.0 of Purity, which is, indeed, not very informative. That is why we must use another measure that applies a penalty to the number of clusters. The NMI is also chosen for this reason.

NMI captures information about class labels even if it is spread in more than one cluster. Its formula is given by equation 8.

$$NMI(Y,C) = \frac{I(Y,C)}{\frac{1}{2}\big[H(Y) + H(C)\big]} \tag{8}$$

where Y is the class labels, C is the cluster labels, H(·) is the entropy, and I(Y, C) is the mutual information between Y and C.

[3]The source code is available in the supplementary materials.

For mathematical clarification, Purity is a homogeneity metric expressed by equation 9.

$$Purity = \frac{1}{N} \sum_{i=1}^{k} max_j \left| c_i \cap t_j \right|, \tag{9}$$

where N is the number of samples, $c_i \in C$ is the cluster, and t_j is the assigned cluster label.