Learning to Cluster with Auxiliary Tasks: A Semi-Supervised Approach

Jhosimar Arias Figueroa and Adín Ramírez Rivera
Institute of Computing, University of Campinas, Campinas, SP, Brazil
Email: jhosimar.figueroa@students.ic.unicamp.br, adin@ic.unicamp.br

***Abstract*—In this paper, we propose a model to learn a feature-category latent representation of the data, that is guided by a semi-supervised auxiliary task. The goal of this auxiliary task is to assign labels to unlabeled data and regularize the feature space. Our model is represented by a modified version of a Categorical Variational Autoencoder, i.e., a probabilistic generative model that approximates a categorical distribution with variational inference. We benefit from the autoencoder's architecture to learn powerful representations with Deep Neural Networks in an unsupervised way, and to optimize the model with semi-supervised tasks. We derived a loss function that integrates the probabilistic model with our auxiliary task to guide the learning process. Experimental results show the effectiveness of our method achieving more than 90% of clustering accuracy by using only 100 labeled examples. Moreover we show that the learned features have discriminative properties that can be used for classification.**

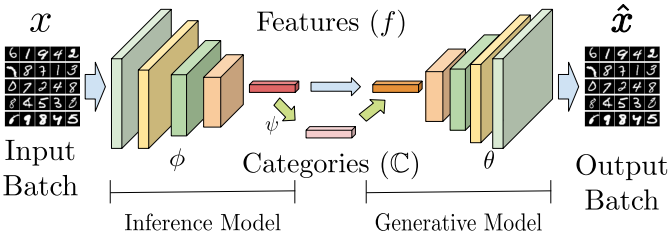


Fig. 1. Our proposed architecture based on the probabilistic model CatVAE [15], [16]. Our encoder (ϕ) transforms the data into a feature (f), that is converted into a category (C) by a set of layers (ψ). The decoder (θ) uses the latent variables (f and C) to reconstruct the image. Section III details the construction of the architecture through a probabilistic model.

I. Introduction

Clustering is an unsupervised problem, widely used in different applications, with the goal of finding hidden patterns or groupings within the data. Commonly, clustering methods are totally unsupervised, meaning that no labels are considered in the process. However, in many situations, a small amount of information is available that may help to guide the learning process by using both type of data, i.e., semi-supervised learning [1]–[5]. A similar approach called semi-supervised clustering [6], [7] aims to improve the performance of unsupervised clustering algorithms with limited amounts of supervision in the form of labels on the data or constraints.

One of the problems when working with semi-supervised learning is how to learn representations for both labeled and unlabeled data. In the last few years, Deep Neural Networks (DNNs) have emerged with promising results for different machine learning tasks, DNNs can help to learn good representations for the data. Normally, these representations can be learned in a supervised way by performing classification [8]–[10] and subsequently adopting the learned representations, or by extracting useful information from data in an unsupervised way [11], [12]. However these approaches do not consider an end-to-end learning framework, as they depend on learning the representations from data and then using these representations for other tasks, e.g., clustering.

The second problem that arises is the representation of clusters. Hence, Deep generative models [13], [14] have become popular due to their ability to learn distributions from data. By using Bayesian inference, researchers created probabilistic generative models which learned representations as distributions of the data in an end-to-end manner. Therefore, the clusters can be represented by a probability distribution, such as a categorical distribution [15], [16], or Mixture of Gaussians [17], [18]. In this work, we do not focus on generating images with the generative model, but rather use the learning process of these models to improve our representations. Despite having good representations for the data and clusters, an important problem remains: how to learn jointly from labeled and unlabeled data?

To address the aforementioned problems, we propose a modification on a generative model that transforms the data into a feature-category latent space (as shown in Fig. 1), and an end-to-end learning framework to teach the model to perform clustering by transferring the knowledge of an auxiliary clustering task. Our goal is to cluster the data by approximating it with a categorical distribution, thus our model is based on a Categorical Variational Autoencoder (CatVAE) [15], [16]. We use the labeled data to input knowledge about the existing clusters to the categorical latent variables, by transferring the information of the labeled data into the unlabeled data through an auxiliary clustering task during training.

Our learning framework uses the generative process to improve the learning through the different phases of our model, while regularizing the categories and the feature representations. Our two fold approach comprises, first, verifying that the learned categories match the labels on the data, and, secondly, by assigning the labels of the known data to the unknown ones that are closer in the feature space. Moreover, we minimize the distance of the features of the elements within

the same cluster to improve the representation and to ease the category learning phase.

The first part of our architecture (depicted in Fig. 1) is treated as an encoder where we obtain an embedding of the current batch. This network is also called inference network because it approximates the feature's posterior with variational inference. From the embedding we obtain an approximation of the categorical distribution. Jointly, the features and the categories are used as inputs to another network that acts as a decoder that tries to reconstruct the original image. In the literature, this network is also called a generative network, because it can generate images with samples from the categorical approximation and feature representations. Although, that is out of the scope of our current proposal.

To summarize, the contributions of our work are:

- a semi-supervised auxiliary task which aims to define the clustering assignment for the unlabeled data that can be used in conjunction with the labeled data;
- a regularization on the feature representations of the data, by minimizing the inter-cluster feature distances which helps to drive the learning process in this mixed tasks; and
- a loss function to guide the learning process based on our auxiliary task taking advantage of the generative model.

The remainder of this paper is organized as follows. We present a brief comparison against existing work to contextualize our proposal in Section II. In Section III, we formalize the definition of our probabilistic model by introducing our inference and generative model based on two latent variables that represent our clusters and features. Section IV presents our proposed semi-supervised auxiliary task comprising the aforementioned contributions of our work. Section V shows results verifying that our proposal gets at least 90% of performance. And, finally, we give our concluding remarks and future works in Section VI.

II. Related Work

Clustering algorithms are commonly related to unsupervised learning where no labeled data is available. There are several studies that combines clustering with DNNs [19], [20]. These works require to pre-train a DNN to obtain initial representations and then fine-tune the pre-trained network jointly with a clustering algorithm like K-means. Recent works utilize an end-to-end optimization [21], [22] involving DNNs and unified loss functions to guide the learning process.

Semi-supervised learning has been applied mostly to classification problems. However, they use different types of auxiliary tasks combined with DNNs (as we proposed in this paper). For instance, Weston et al. [1] improved supervised learning for deep architectures by adding an unsupervised embedding on any (or all) layers as an auxiliary task. Recently, Rasmus et al. [3] proposed a semi-supervised model based on ladder networks where the learning task is similar to that in denoising autoencoders but applied to every layer, not just the inputs.

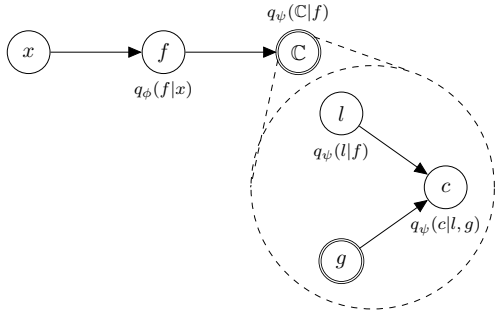
Besides of normal autoencoder-based architectures, other methods take advantage of probabilistic graphical models and Bayesian inference, such as Variational Autoencoders [13] (VAEs) which are generative models represented by an encoder that infers the posterior distribution in a latent space and a decoder which generates images by minimizing the reconstruction loss. Recent works proposed extensions and modifications over the VAE model by adding discrete variables and considering the importance of hierarchical latent variables. Kingma et al. [2] introduced a deep generative model for semi-supervised learning based on a VAE whose latent space is the joint distribution over data and labels. The problem with this model is the layer-wise pre-training which sometimes is costly because we require to train two networks, the first one for better feature representations (M1) and the second one for semi-supervised learning (M2). On the contrary, our approach does not require pre-training because our auxiliary task helps to improve the feature representations by considering the distances between elements of the same cluster.

Maaløe et al. [4] introduced the auxiliary deep generative model (ADGM) for semi-supervised learning which utilizes an extra set of auxiliary latent variables to improve the variational lower bound. Their model is an improvement to the model proposed by Kingma et al. [2] in the sense that it is possible to train the model in an end-to-end fashion with more than one stochastic variable, and without pre-training. Despite of the success of stochastic latent variables, their complexity increases when dealing with discrete variables because it is required to marginalize them out in order to backpropagate them. Unlike their work, we do not marginalize out the discrete variables when backpropagating instead we approximate a categorical distribution by adopting the Gumbel-Softmax/Concrete distribution proposed by Jang et al. [15] and Maddison et al. [16].

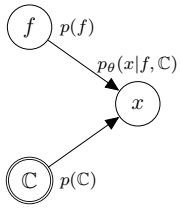
Some works have explored the combination of clustering with semi-supervised learning. For instance, Maaløe et al. [23] proposed the Cluster-aware Generative Model (CaGeM) based on a VAE and discrete variables. They showed that higher latent representations can create clusters using unlabeled information and their performance can be refined using additional labeled information. Compared to our approach, they focus on the improvement of generative performances rather than clustering. More recently Dizaji et al. [22] introduced the clustering model called Deep Embedded Regularized Clustering (DEPICT) based on stacked softmax layers on top of deep convolutional autoencoders. Unlike our approach, they use softmax classifiers for the clustering assignments and denoising autoencoders. We do not require a denoising and a stacked softmax representation to define our clusters because we take advantage of the generative process of our categorical distribution.

III. Probabilistic Model

In this section we define our probabilistic model based on a Variational Autoencoder [13] (VAE). Unlike this generative model, which approximates a Gaussian distribution, our model



(a) Inference Model $q(\mathbb{C}, f|x)$



(b) Generative Model $p(x|f, \mathbb{C})$

Fig. 2. The probabilistic graphical model of our proposed solution. (a) The inference is done from the data, x, to the features, f, and finally to the categories, $\mathbb{C}$, which are a discrete node. We re-parametrized the categories with a Gumbel-Softmax distribution, with g as a stochastic Gumbel distribution generator. (b) The generative model corresponds to our decoder that transforms the features, f, and the categories, $\mathbb{C}$, into the reconstructed data. (Double line nodes represent non-deterministic nodes.)

approximates a categorical distribution that fits better for clustering. This variation of the VAE is called a Categorical VAE (CatVAE) [15], [16].

We use two latent variables to model the data, the features, f, and the categories, $\mathbb{C}$, of the data, i.e., we have $p(x, f, \mathbb{C})$. Our assumption is that we can have a generative process, such that if we sample $f \sim p(f)$ and $\mathbb{C} \sim p(\mathbb{C})$, we can generate a new data point by $x \sim p(x|f, \mathbb{C})$—as shown in Fig. 2(b). This process is the main driver of our method. Since we are interested in clustering the data, we do not care about the generated data, but rather about the existing latent space for the categories.

Since we have a generative process, we also have the inverse inference process: $p(\mathbb{C}, f|x)$. We assume that the categories are dependent on the features, as shown in Fig. 2(a), thus,

$$p(\mathbb{C}, f|x) = p(\mathbb{C}|f)p(f|x). \tag{1}$$

Nevertheless, computing the latter distribution is intractable since the prior of x, $p(x)$, needs to be computed for the entire latent space. Thus, as the literature suggests, we approximate the distribution with variational parameters (i.e., families of distributions), such that

$$p(\mathbb{C}, f|x) \approx q_{\psi,\phi}(\mathbb{C}, f|x), \tag{2}$$

that is, we want to find the variational parameters ψ and ϕ that minimize the divergence between our variational posterior, $q_{\psi,\phi}(\mathbb{C}, f|x)$, and the true posterior, $p(\mathbb{C}, f|x)$, therefore we have

$$\arg\min_{\psi,\phi} \big\{ KL\big(q_{\psi,\phi}(\mathbb{C}, f|x)||p(\mathbb{C}, f|x)\big)\big\}, \tag{3}$$

we explain the solution of this divergence in Section IV-A.

Due to the assumptions given by (1), we can approximate the features as

$$p(f|x) \approx q_\phi(f|x). \tag{4}$$

This distribution allows us to compute our latent variables from the data, x. On the other hand, our categorical variable, $\mathbb{C}$, is a discrete node that represents a categorical distribution that we use to associate each cluster with each category. Nevertheless, this variable will not allow us to easily train our distributions as neural networks because we can not backpropagate through discrete nodes. Thus, we need to reformulate it into a deterministic path within the model. We use the reparametrization trick [13] by using a Gumbel-Softmax distribution [15], [16] to approximate our categorical distribution with a continuous one. Since, we have a categorical variable $\mathbb{C}$ with class probabilities π_i, we transform it to a continuous distribution, c, by

$$c = \text{one hot}\left(\arg\max_i \{g_i + \log \pi_i\}\right), \tag{5}$$

where one hot is an operator that generates a vector with a one in the given parameter and zeros elsewhere, and with independent draws $g_i \sim Gumbel(0, 1)$.[1] Nevertheless, we still cannot backpropagate through the $\arg\max$ operator, but we can replace it with softmax function, such as

$$c_i = \frac{\exp\left\{\left(\log(\pi_i) + g_i\right)/\tau\right\}}{\sum_{j=1}^{k} \exp\left\{\left(\log(\pi_j) + g_j\right)/\tau\right\}}, \tag{6}$$

for every i, and where k is the number of categories. The above conversion is known as the Gumbel-Softmax [15] or the Concrete [16] distribution trick. The softmax temperature τ controls the discreteness of (6). When $\tau \to 0$ the samples are identical to those generated by (5), they become one-hot. At higher temperatures $\tau \to \infty$ the samples are not longer one-hot and become uniform.

To obtain the probabilities π of a given feature, f, we learn them through another latent variable. As shown in Fig. 2(a), we compute another variational distribution $q_\psi(l|f)$, where l is a latent variable that holds such probabilities, and f is our feature. Since we are approximating this distribution with a neural network, we can obtain the $\log(\pi)$ directly, instead of the simple probabilities. Finally, we can approximate the inference from the features as

$$p(\mathbb{C}|f) \approx q_\psi(\mathbb{C}|f) \approx q_\psi(c|l, g)q_\psi(l|f), \tag{7}$$

where $q_\psi(c|l, g)$ is defined as a Gumbel-Softmax distribution, and when drawing c we compute (6).

We are implementing our distributions through neural networks; thus, the main distributions' parameters corresponds to the neural network's parameters that implements them, i.e., $q_\phi(f|x)$, $q_\psi(l|f)$, and $p_\theta(x|f, \mathbb{C})$—cf. Fig. 1 and Section V-A.

[1]The distribution $Gumbel(0, 1)$ can be sampled by drawing $u \sim Uniform(0, 1)$, and computing $g = -\log(-\log(u))$.

IV. Learning with an Auxiliary Task

In this section we introduce our main proposal which allows to jointly learn from both labeled and unlabeled data. To this end, we define two auxiliary tasks, the first one considers the feature representations of the data to learn cluster assignments, and the second one is a regularizer which minimizes the distances of the feature representations of the same cluster.

Our main objective for teaching our network is to improve the cluster representations. To achieve it we must maintain the reconstruction task that drives the distributions from which we are deriving the inference of the category space. On top of that, we must maintain the consistency of the label data we have, and to maximize the chance of assigning correct labels to our unlabeled data. The former is similar to performing a classification task on the labeled data, as we want to predict the correct label to each of them. The latter requires us to infer the labels for the data. To do so, we exploit the information of the existing ones to serve as representatives for each cluster, and we assign them the labels of their closest known neighbor. Finally, we need to minimize the differences of the feature representations of within the same cluster.

To achieve all those goals, we define a loss function that should be minimized as

$$\mathcal{L}_{total} = w_R\mathcal{L}_R + w_C\mathcal{L}_C + w_A\mathcal{L}_A + w_F\mathcal{L}_F, \tag{8}$$

where $\mathcal{L}_R$ is the reconstruction loss, $\mathcal{L}_C$ is the loss of the categorical distribution fit, $\mathcal{L}_A$ is the assignment loss of an element to a determined cluster, $\mathcal{L}_F$ is the feature loss which works as a regularizer for the feature representation in each cluster. We consider the normalized version of each loss in order to have values in the same range, in this way we can define the importance of each loss by assigning them some weights w_*. We explain the selection of these weights in Section V-C.

A. Categorical VAE Loss

The loss function of this type of generative models is derived with the help of variational inference. When working with variational models we are interested in the posterior $p(\mathbb{C}, f|x)$ because we want to learn a distribution from the observable variable x. The posterior is intractable because the normalization factor, $p(x)$, depends on the latent variables. We can approximate the posterior with a variational distribution $q_{\psi,\phi}(\mathbb{C}, f|x)$, cf. (2), that we want to find, by maximizing the Evidence Lower Bound (ELBO) [13], $\mathcal{L}(\psi, \phi)$, which is equivalent to solve (3). A lower bound for the intractable log-likelihood $\log p(x)$ obtained using Jensen's Inequality is

$$\log p(x) \geq E_{q_{\psi,\phi}(\mathbb{C},f|x)}\left[\log \frac{p(\mathbb{C}, f, x)}{q_{\psi,\phi}(\mathbb{C}, f|x)}\right] = \mathcal{L}_P, \tag{9}$$

where $q_{\psi,\phi}(\mathbb{C}, f|x)$ is our approximation of the inference, and $p(\mathbb{C}, f, x)$ is the joint probability between the likelihood (how likely our categorical variable $\mathbb{C}$ and features f are to generate data x) and a prior. We represent the approximate distribution $q_{\psi,\phi}(\mathbb{C}, f|x)$ with a deep neural network parameterized by ψ and ϕ (as an encoder, see inference model in Fig. 1).

The variational lower bound specified in (9) is defined as

$$\begin{aligned}\mathcal{L}_P =& E_{q_{\psi,\phi}(\mathbb{C},f|x)}\left[\log p(x|\mathbb{C}, f)\right] - \\ & KL\big(q_{\psi,\phi}\left(\mathbb{C}, f|x\right)||p\left(\mathbb{C}, f\right)\big),\end{aligned} \tag{10}$$

where the first term is the reconstruction loss ($\mathcal{L}_R$) which encourages the decoder to learn to reconstruct the data when using samples from the categorical latent space and feature representations. We approximate this loss with the normalized mean square error (MSE) defined by

$$\mathcal{L}_R = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|x_i|}(x_i - \hat{x_i})^2, \tag{11}$$

where N is the number of samples, $|x_i|$ is the size of the i-th input (e.g., number of pixels in an image) and $\hat{x}$ is the reconstruction of the input obtained from our network (as a decoder, see generative model in Fig. 1).

The second term of (10) is the distance between the joint distribution of our latent variables w.r.t. the given data and the prior. However, we are not interested in the joint distribution, but rather on a way to optimize each independent variable. Thus, by working the KL-divergence we obtain

$$\begin{aligned}\mathcal{L}_P =& E_{q_{\psi,\phi}(\mathbb{C},f|x)}\left[\log p(x|\mathbb{C}, f)\right] - \\ & E_{q_\phi(f|x)}\left[KL\left(q_\psi\left(\mathbb{C}|f\right)||p\left(\mathbb{C}\right)\right)\right] - \\ & KL\left(q_\phi\left(f|x\right)||p\left(f\right)\right),\end{aligned} \tag{12}$$

where the new terms represent the regularization of $\mathbb{C}$ and f, respectively. Since, we are not assuming any variational distribution for f, we will approximate it with the feature loss $\mathcal{L}_F$ (see Section IV-C). Note that for the $\mathbb{C}$'s regularizer we obtain the expected divergence over the possible features produced by our data, i.e., $f \sim q_\phi(f|x)$. Since we want to maximize $\mathcal{L}_P$, we need to minimize the expected KL-divergence, which is equivalent to minimize the KL-divergence alone. Hence, our categorical loss becomes

$$\mathcal{L}_C = KL\left(q_\psi\left(\mathbb{C}|f\right)||p\left(\mathbb{C}\right)\right) \approx KL\left(q_\psi\left(l|f\right)||p\left(l\right)\right). \tag{13}$$

This term can be interpreted as regularizing ψ, encouraging the approximate posterior $q_\psi(\mathbb{C}|f)$ to be close to the prior $p(\mathbb{C})$. However, as we can see in (7), we are required to approximate only the logits, $q_\psi(l|f)$, because the first term is already defined. The model proposed by Kingma et al. [13] specifies the prior as a standard normal distribution $\mathcal{N}(0, 1)$ because they consider a multivariate Gaussian for the approximate posterior. Unlike their model, ours uses categorical latent variables. We specify the prior as a standard uniform distribution $Uniform(0, 1)$ because initially all categories are equally likely to be chosen. Finally our normalized categorical loss is given by

$$\mathcal{L}_C = \frac{1}{N\log K}\sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\log\left(Kq_{ik}\right), \tag{14}$$

where N is the number of samples to cluster, K is the number of clusters, q_{ik} are the probabilities of the logits obtained with a softmax function and $N\log K$ is the normalization factor, allowing to obtain values between 0 and 1.

B. Assignment Loss

To verify the assignments of the network to the unlabeled data, we need to infer the more likely label for each unlabeled sample we are observing. To achieve that we exploit the information from the labeled data, and use them as representatives of their respective cluster to disperse their labels to the unlabeled samples. In order to assign a label for the unlabeled data, we use the nearest neighbor (1-NN) on the feature space, i.e., we compare the feature of the unlabeled samples with the labeled ones, and assign the label of the closest one. Note that any other clustering or assignment algorithm will work for this task, and we selected 1-NN for its simplicity. We rely on the fact that similar features learned by the neural network should belong to the same cluster.

Once we obtained the assignments for the unlabeled data, we consider a modification of the negative log-likelihood loss. Since log-likelihood loss maximizes the log-probability of the correct assignment, this loss is normally used in supervised learning. In our case, we have the assignments for the unlabeled data, plus the existing labels of the labeled data. One problem with this loss is that it only penalizes one category at a time, and it does not assign the correct label in case of a mistake. Our modification of the negative log-likelihood, besides trying to maximize the log-probability of the potential true category, it minimizes the log-probability of other categories

$$\mathcal{L}_A = \sum_{i=1}^{N}\sum_{k=1}^{K}\left[-\log P(c_{ik}|x_{ik}) + \sum_{j\neq k}\log P(c_{ij}|x_{ik})\right], \quad (15)$$

where c_{ik} is the assigned category to the data x_{ik}, and c_{ij} are the rest of categories, $P(c_*|x_*)$ values are given by (6). That is, the loss function penalizes the assignment if it is not correct by moving the wrong predictions away, and the correct ones closer. Note that (15) is not normalized, besides that, small values of $P(c_*|x_*)$ produce large values for $-\log P(c_*|x_*)$ and small values for $\log P(c_*|x_*)$, i.e., $\lim_{x\to 0}\log(x) = -\infty$. Thus, we normalize the values of this function independently with a hyperbolic function

$$\begin{aligned}\mathcal{L}_A = &\frac{1}{2N}\sum_{i=1}^{N}\sum_{k=1}^{K}\Bigg[\tanh\big(-\log P(c_{ik}|x_{ik})\big) + \\ &\sum_{j\neq k}^{K}\tanh\big(\log P(c_{ij}|x_{ik})\big) + 1\Bigg],\end{aligned} \quad (16)$$

where $2N$ is the normalization factor that allows values between 0 and 1, we add the value 1 in the last term because $\tanh$ produces values between -1 and 0 for negative values.

C. Feature Loss

In this section we introduce our second auxiliary task defined as a regularizer over the feature representations. Let $D(a, b)$ be a metric function which measures distances between a and a group of elements b in the feature space. In our experiments, we use $D = \ell_2$ distance, but any distance can be used. And given the features of all the data, we want the neural network to learn an embedding such that the distances between elements belonging to the same cluster are minimized. Therefore we want the feature representations of similar elements be as close as possible.

Since we have a subset of labeled data for each class, we are sure that their feature's distances have to be close. On the contrary, we cannot be sure for the unlabeled data. That is why we weight differently the importance of distances for the labeled and unlabeled data (through a parameter $\alpha \in [0, 1]$). Our feature loss function is defined as

$$\mathcal{L}_F = \frac{\alpha}{L}\sum_{i}^{L} D(f_i, r_i) + \frac{(1-\alpha)}{U}\sum_{j}^{U} D(f_j, r_j), \quad (17)$$

where L is the number of labeled samples, U is the number of unlabeled samples, f_i are the labeled data features, f_j correspond to unlabeled samples, and r_k are all the labeled features that correspond to the assigned cluster of f_k, i.e., $r_k \in C_k$, where C_k is the cluster that f_k belongs to. We normalize $D(f, r)$ in order to obtain values between 0 and 1

$$D(f, r) = \frac{1}{\sqrt{2}|r|}\sum_{l}^{|r|}\|f - r_l\|, \quad (18)$$

where $\sqrt{2}|r|$ is the normalization factor, $|r|$ is the number of labeled features of the assigned cluster of f and the value $\sqrt{2}$ is an upper bound of the ℓ_2 distance assuming that the features have positive values and unit norm, i.e., $\|f\| = 1$. We assure that our features, f, have these properties (see Section V-A1).

V. Experiments and Results

We evaluate the accuracy of our proposed method on the standard MNIST [24] classification data set which consists of 70000 images of size 28×28, and 10 classes representing digits. For our problem, we consider the number of classes as the number of clusters, and try to learn the clustering of the data, instead of simply classifying the data. We consider the standard split of the data set which consists of 60000 images for training and 10000 images for testing. From the training set, we randomly chose 100 labeled examples (we ensure that all classes have the same number of labeled images) that are fixed for the whole training phase. After that, we randomly split the 59900 training images left into 80% for training set and 20% for validation set, we perform the splits trying to maintain the classes balanced for both training and validation sets (we group by labels and obtain the desired percentage for each category). This set of images is our unlabeled data.

A. Proposed Network Architecture

We adopted the architecture proposed by Dilokthanakul et al. [17] with modifications in the input and output of the inference and generative networks. All the convolutional layers, except the output layer of the generator network, are followed by batch normalization [25] and rectified linear unit (ReLU) as non-linearity. We represent the convolutional layers as $(n@h \times w, s, p)$ where n is the number of filters, h and w are the height and width of each filter respectively, s is the

stride and p is the padding. In the following we detail the architecture of our networks.

1) Inference Network: This network is based on a convolutional encoder. It receives the input image (28×28), then the next layers are convolutional: $\{(16@6 \times 6, 1, 0)$, $(32@6 \times 6, 1, 0)$, $(64@4 \times 4, 2, 1)$, $(50@9 \times 9, 1, 0)\}$. At this point we have a vector of size (1×50), this size was chosen experimentally (see Section V-C), followed by a ReLU activation function in order to obtain positive values and then normalize it to have unit norm with a ℓ^2 normalization layer. This normalized vector is our feature representation f—we benefit from this representation in our feature loss (see Section IV-C). This layer is followed by three linear layers with ReLU activations in between: $\{Linear(50, 50)$, $Linear(50, 25)$, $Linear(25, k)\}$, the last layer outputs the logits l of size k used to approximate a categorical distribution, $q_\psi(l|f)$ by (7). For MNIST, we require 10 categories, thus $q_\psi(l|f) \approx Linear(25, 10)$. Then, we have another layer that represents the Gumbel-Softmax version, c, of our categories, $\mathbb{C}$, of size (1×10) by (6). The feature and categorical representations, f and c, respectively, will be passed as input to the generator network.

2) Generative Network: This network is based on a convolutional decoder. It receives the feature vector f of size (1×50) and a vector of probabilities, c, of size (1×10) obtained from the Gumbel-Softmax distribution. These two vectors are concatenated, (1×60) and fed to the generative network as our input. First, a linear layer is applied to obtain a vector with the same size of the feature representation f, $Linear(60, 50)$. Then, it is followed by the reverse version of the encoder: $\{(64@9 \times 9, 1, 0)$, $(32@4 \times 4, 2, 1)$, $(16@6 \times 6, 1, 0)$, $(1@6 \times 6, 1, 0)\}$. At this point we have a matrix with the same dimensions of our input image (28×28). We assume that the input image follows a Bernoulli distribution (values between 0 and 1). Hence, we use a *Sigmoid* layer as our final activation which outputs the reconstructed image.

B. Training

Our model uses the initialization defined by LeCun et al. [26]. For training we used Adam [27] optimizer with the default hyperparameters for 1st- and 2nd-order moments $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We iterate for 100 epochs, and in every epoch we consider a different random permutation of our data. We use a batch size of 200 images being 100 the labeled ones and the rest unlabeled.

C. Hyperparameters

An important step when training neural networks is the setup of hyperparameters. Since it is hard to do a full hyperparameters search due to the high dimensionality and prohibiting computational time, we chose our parameters independently. All the hyperparameters were found in the validation set. We started with the following setup: $\{f_{sz} = 200$, $\eta = 0.1$, $\eta_d = 0.5$, $\tau = 0.5$, $\alpha = 0.8$, $w_R = 1$, $w_C = 1$, $w_A = 1$, $w_F = 1\}$, where f_{sz} is the feature size, η is the learning rate, η_d is the learning rate decay, τ is the temperature used in Gumbel-Softmax, α is the importance of the labeled data in the regularization of distances, and w_* are the weights used in our loss function $\mathcal{L}$—cf. (8). We began varying the learning rate η, we tested with values (0.0001, 0.001, 0.01, 0.1) and found that 0.001 gives better results. Additionally we decay the learning rate by 0.5 every 50 epochs. For the temperature τ, used in Gumbel-Softmax, we tested values (0.5, 1, 1.5), and found the best result with a fixed temperature is 1. We tried to smooth the temperature from 1 to 0.5 without good results. For α we tested values (0.5, 0.6, 0.7, 0.8) and found that 0.6 is a good trade-off value that gives more importance to the embeddings of labeled data than the unlabeled ones, and simultaneously avoids over-fitting by regularizing it with the unlabeled data. The weights of our loss function were adapted separately. We found that changing w_R and w_F did not improve the results. Nevertheless, we obtained more than 90% accuracy by increasing the assignment loss, i.e., w_A. We tested several values (5, 10, 20, 30), and with all of them we outperformed our initial results. As shown in Fig. 3, without this weight the assignment loss does not help in the guidance of the loss function. Likewise, we improved the results by decreasing the weight of the categorical loss, i.e., w_C. We considered several values (0.1, 0.2, 0.3, 0.4), and found that with 0.1 we obtained similar results as those of increasing w_A, see Fig. 3. We tried to increase w_A and decrease w_C simultaneously, as both of them improve the results independently, without success. Finally, we set $w_A = 5$ for the assignment loss and $w_* = 1$ for the other weights, because we found that experimentally giving more weight to the assignment of cluster drives the process in a better way. We tested different sizes of feature representations, as shown in Fig. 4, for values between 50 and 300 we obtained the best results, thus we chose the size of 50 because of efficiency and performance. Our final setup is specified in Table I.

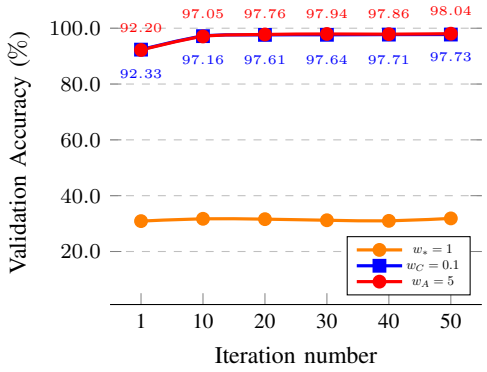


Fig. 3. Importance of weights (w_A, w_C) at different epochs.

TABLE I
SELECTED HYPERPARAMETERS OF OUR MODEL.

Hyperparameter	f_{sz}	η	η_d	τ	α	w_A
Value	50	0.001	0.5	1.0	0.6	5

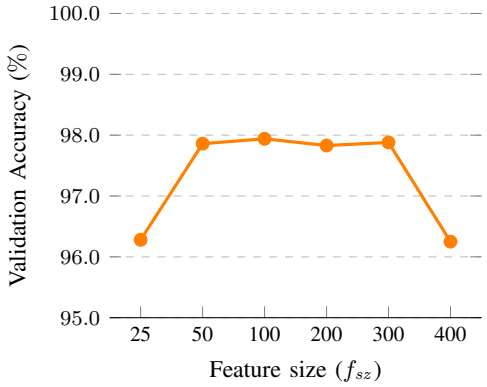


Fig. 4. Hyperparameter setup on feature representation size.

D. Results

In this section we present our results compared to the state of the art. While our main focus is clustering based on semi-supervised learning, we also show competitive results for semi-supervised classification. All the results of the related works are reported from the original papers. The character '-' means that results for that metric or setup were not executed. For all the experiments we did not perform any type of pre-processing over the images, we only normalize their pixel values to the range of $[0, 1]$. Our final results are performed using all the 60000 training samples with 5 random seeds for network initialization and data splits.

1) Clustering: We evaluated our clustering results with two clustering metrics commonly used in the literature, clustering accuracy (ACC) and normalized mutual information (NMI) [28]. Both metrics are in the range of $[0, 1]$, where larger values indicate more precise clustering results. For training, we considered 100 labeled examples evenly distributed across the categories. Note that the related works are fully unsupervised, even if the results are not comparable we want to show that our model is good for clustering and illustrate the results of similar tasks.

As we can see in Table II, unsupervised algorithms show promising results, most of them obtained more than 90% of performance. GMVAE [17] and VADE [18] are methods based on generative models and bayesian inference that are very related to us in terms of the algorithms and techniques employed. JULE-RC [21] and DEPICT [22] are models based on Deep Neural Networks, the former use a recurrent approach and the latter use denoising autoencoders. According to the results, generative bayesian models can be applied to clustering and show competitive results but they are outperformed by non-generative approaches. Our results show that by considering a small amount of labeled data, which commonly is easy to obtain, it is possible to get good clusters and improve the results. In Fig. 5, we can visualize the feature representations f of the MNIST test set, we used t-SNE [29] for the visualization. We can see how each digit was assigned to a different cluster.

2) Semi-supervised Classification: We evaluate our model for the task of classification, considering only related works which use deep neural networks as their learning representations for a fair comparison. Table III shows the classification error considering 100 labeled examples, and all the examples of the data (fully supervised). We chose these values because all the related works report their results with the same number of labeled examples. Despite the design of our model that is not built for classification, it achieves comparable results with the state of the art, outperforming different models such as EmbedCNN [1], which improves supervised learning by jointly learning an embedding task using unlabeled data, SWWAE [30], which uses unpooling layers in the decoder and trains jointly a supervised loss with reconstruction loss on each level of the network, and DEPICT [22], which employs denoising autoencoders with reconstruction losses in a similar way as SWWAE [30], and has comparable results with CatGAN [31] which is based on Adversarial Networks [14]. These results show that the feature representations learned by our model have good discriminative properties that is desired for the clustering embedding. The classification is a byproduct of the cluster definition within our model.

We believe that our model does not perform so well as the Ladder Network [3] due to the use of reconstruction losses on each level of the network, we have seen that other models, SWWAE [30] and DEPICT [22], use this type of regularization on each layer and showed that reconstructing from denoising layers improve the performance, besides that we have not tested and compared our model with natural images datasets. We leave these experimental tests for future works.

TABLE II
CLUSTERING PERFORMANCE, ACC AND NMI, ON MNIST OF DIFFERENT UNSUPERVISED ALGORITHMS

Model	NMI	ACC
GMVAE [17]	-	0.778
VADE [18]	-	0.945
JULE-SF [21]	0.876	0.940
JULE-RC [21]	0.915	0.961
DEPICT [22]	0.916	0.965
Proposed	0.954	0.983

Note that our results are not directly comparable with unsupervised methods. However, we want to show our model's clustering results.

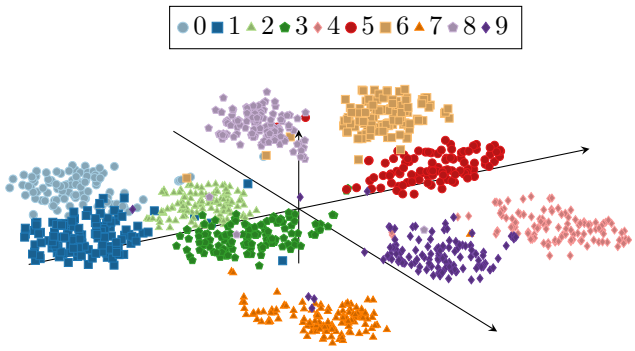


Fig. 5. Visualization of the feature representations of the MNIST test set with t-SNE [29], from 50 dimensions to 3. (View with 60° azimuth and 60° elevation.)

TABLE III
SEMI-SUPERVISED TEST ERROR (%) BENCHMARKS ON MNIST FOR 100 RANDOMLY AND EVENLY DISTRIBUTED LABELED DATA.

Model	With n labeled examples	
	100	All
EmbedCNN [1]	7.75	-
SWWAE [30]	8.71 ($\pm$ 0.34)	0.71
Small-CNN [3]	6.43 ($\pm$ 0.84)	0.36
DEPICT [22]	2.65 ($\pm$ 0.35)	-
Conv-CatGAN [31]	1.39 ($\pm$ 0.28)	0.48
Conv-Ladder τ-model [3]	0.89 ($\pm$ 0.50)	-
Proposed	1.67 ($\pm$ 0.18)	0.70

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a variation of the Categorical Variational Autoencoder, as well as a semi-supervised end-to-end learning framework that is based on an auxiliary clustering task. We assigned images to clusters, and regularizes the embedding of the feature space by minimizing the distance between similar feature representations, and by penalizing the cluster assignment with the aid of the known data assignments. Experimental results show that our approach can generate good clusters with only 100 labeled images, additionally the learned features have discriminative representations which can be employed in classification tasks.

Future work focuses on the evaluation of clustering algorithms (e.g., K-means, DBSCAN, agglomerative clustering, etc.) as part of our auxiliary task, which can yield a full unsupervised model. Furthermore improvements of our probabilistic generative model can be performed by using generative adversarial models.

ACKNOWLEDGMENT

This work was supported in part by CNPq-Brazil through Grant 132848/2015-5, and in part by the São Paulo Research Foundation (FAPESP) under Grant 2016/19947-6.

REFERENCES

[1] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," in *International Conference on Machine Learning (ICML)*, 2008.

[2] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[3] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko, "Semi-supervised learning with ladder networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[4] L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther, "Auxiliary deep generative models," in *International Conference on Machine Learning (ICML)*, 2016.

[5] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *arXiv preprint arXiv:1606.03498*, 2016.

[6] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl, "Constrained k-means clustering with background knowledge," in *International Conference on Machine Learning (ICML)*, 2001.

[7] S. Basu, A. Banerjee, and R. J. Mooney, "Semi-supervised clustering by seeding," in *International Conference on Machine Learning (ICML)*, 2002.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[9] C. Szegedy, W. L., Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," in *The Journal of Machine Learning Research*, 2010, pp. 3371–3408.

[12] A. Dosovitskiy, P. Fischer, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with exemplar convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 9–50.

[13] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[15] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations (ICLR)*, 2016.

[16] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *International Conference on Learning Representations (ICLR)*, 2016.

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

[18] Z. jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, "Variational deep embedding: A generative approach to clustering," in *arXiv preprint arXiv:1611.05148*, 2016.

[19] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International Conference on Machine Learning (ICML)*, 2016.

[20] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *International Conference on Machine Learning (ICML)*, 2017.

[21] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[22] K. G. Dizaji, A. Herandi, and H. Huang, "Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization," in *arXiv preprint arXiv:1704.06327*, 2017.

[23] L. Maaløe, M. Fraccaro, and O. Winther, "Semi-supervised generation with cluster-aware generative models," in *arXiv preprint arXiv:1704.00637*, 2017.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Learning Representations (ICLR)*, 2015.

[26] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*, 1998, pp. 9–50.

[27] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[28] W. Xu, X. Liu, and Y. Gong, "Document clustering based on non-negative matrix factorization," in *International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, 2003, pp. 267–273.

[29] L. v. d. Maaten and G. Hinton, "Visualizing high-dimensional data using t-sne," in *The Journal of Machine Learning Research*, 2008, pp. 2579–2605.

[30] J. Zhao, M. Mathieu, R. Goroshin, and Y. LeCun, "Stacked what-where auto-encoders," in *International Conference on Learning Representations (ICLR)*, 2016.

[31] J. T. Springenberg, "Unsupervised and semi-supervised learning with categorical generative adversarial networks," in *arXiv preprint arXiv:1511.06390*, 2015.