Clustering Meets Implicit Generative Models: A Case For VAEs

Anonymous Authors[1]

Abstract

TODO

1. Introduction

In recent years, generative models have attracted significant attention which resulted in many success stories in machine learning. Generative models are trained from unlabelled data and are capable of generating samples which resemble the ones from the training distribution. This task is of crucial importance for the machine learning community as it is a fundamental component of unsupervised learning. Two of the most prominent representative are Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) and Variational Autoencoders (Kingma & Welling, 2013). Both approaches aim at minimizing the discrepancy between the true data distribution and the one learned by the model. The model distribution is typically parametrized with a neural network which transforms random vectors into samples in the space of the training data (for example a picture space). Variational Autencoders are based on an elegant bayesian interpretation. They maximize the log likelihood and are able to perform efficient approximate inference on probabilistic models with continuous latent variables and intractable posterior. Furthermore, they come with an encoder network which maps data points to the latent space. On the other hand, VAEs are known to produce blurry samples when applied to natural images. GANs take a completely different approach, relying on adversarial training. This resulted in impressive empirically results. On the contrary, GANs are notoriously hard to train and suffer from the *mode collapse* problem. Indeed, if the data distribution lays outside the class of functions that the generator network can learn the latter will ignore portions of the former and focus on the parts which can be approximated well given its limited capacity. An enormous effort have been made by the community to tackle this problem and several different approaches were proposed. The most relevant to our setting is the work of (Tolstikhin et al., 2017) who proposed to train multiple generators, one after the others. As long as each generator collapse on a different mode, and given a large enough number of generators, one can approximate the whole data distribution by combining them. Several works followed up, trying to avoid the sequential training of the generator, most notably (Hoang et al., 2017) proposed to train multiple generators in parallel using adversarial training and a classifier to help them separate to different modes. Instead, Variational Autoencoders stands at the other extreme suffering infinite loss if they do not model the whole support of the data distribution. We aim at bridging this gap developing a general approach to train multiple generators in parallel which focus on different parts of the training distribution. We particularize this framework for VAEs. As a consequence, each VAE will be able to collapse on some mode while the mixture of generators (decoders) will still approximate the data distribution.

Borrowing ideas from both the clustering and the causality literature we assume that the data was generated by *independent mechanisms*. This assumption states that the each random variable in the system is independent from the others given its parents. In other words, the generative process is composed of separate modules that do not inform nor interfere with each other. Furthermore, any knowledge of one of these modules do not bring any knowledge about the others. In this setting, each training point was generated by an unknown mechanism and we get to observe only a mixture of these realizations. Recovering the mechanism is a hard and ill posed inverse problem. A critical assumption is that the true causal conditional distribution is the simplest among all the possible factorizations of the random variables. Simplicity can be measured with the *Kolmogorov Complexity* (Janzing & Scholkopf, 2010). Inspired by these notions, we aim at approximating the data distribution by dividing it in different components with non overlapping support. Then, each part can be approximated by an independent generative model. In this way, we allow VAEs to learn only a fraction of the data distribution. As a consequence, they are able to model each part of the distribution better than a single model trying to approximate the whole data distribution. While the optimal split of the data distribution is unknown, we rely on a competitive procedure based on a set of discriminators to separate the data into the different

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

mechanisms. Each training point will be won by the model which generates the most similar samples according to the discriminators. Intuitively, a model that approximates the true causal mechanism will be easier to learn. Therefore, we exploit the implicit regularization given by the parametric form the networks to split the data in a way which is simple for them to learn.

In this paper we design a training procedure to split the data distribution into components which are approximated by independent generative models. This approach has several benefits:

- We provide a clear algorithmic framework for training mixtures of generative models. We target the most abstract case of minimizing a general f-divergence using a mixture of generators. Furthermore, we show convergence of the training procedure to the minimizer of the empirical estimate of the divergence
- We particularize our framework to mixtures of VAEs. By doing so, we allow them to collapse on separate modes of the distribution. As a consequence, we force the VAEs to produce the samples strictly inside the support of the data distribution.
- We present a clustering interpretation of our approach in non metric spaces and recover k-means as a special case.
- We provide empirical evidence that shows that the training procedure do split the training distribution into semantically different components which also increase the log likelihood estimate.

2. Generative Mixtures: Problem Setting

Let $\mathbf{X}$ be a dataset composed of N samples x from the data space $\mathcal{X}$. Furthermore, let P_d be an unknown data distribution defined over the data space $\mathcal{X}$ with support X to be approximated with an easy to sample distribution P_{model}. The goal of generative density estimation is to have the samples from both distributions to look alike. This is typically formulated as some optimization problem minimizing the disagreement between the true data distribution P_d and the approximating one P_{model}.

To measure such disagreement it is nowadays common practice to use a f-divergence:

$$D_f(Q\|P) := \int_{x\in\mathcal{X}} f\left(\frac{dQ}{dP}(x)\right) dP(x) \qquad (1)$$

where $f(1) = 0$ and f is convex. $D_f(Q|P)$ is *jointly convex* in P and Q (Nowozin et al., 2016). Therefore, the goal is to solve the following optimization problem:

$$\min_{P_{model}\in\mathbb{P}} D_f(P_{model}|P_d)$$

Unfortunately, P_d is unknown and only an empirical estimate of D_f is available through the samples in the training set $\mathbf{X}$. While this procedure it at the heart of the adversarial training of GANs (Goodfellow et al., 2014), VAEs (Kingma & Welling, 2013) minimize a variational bound on $D^{KL}(P_{model}|P_d)$ which is just another of the different divergences which can be written in the form of (1) with an appropriate choice of f (Nowozin et al., 2016). We assume that the true natural generating procedure for the data is composed by independent mechanisms which generate $\mathbf{X}$ according to some distribution P_d^j (Schölkopf et al., 2012; Peters et al., 2017). Therefore, we consider P_{model} as a mixture of distributions $P \in \mathbb{P}$:

$$P_{model} = \sum_{j=1}^{k} \alpha_j P_{g_j} \qquad (2)$$

each of them specialized on one of the generating mechanisms. Training mixtures of experts with boosting algorithms like the one in (Tolstikhin et al., 2017) has favorable optimization properties. In particular, adding components to a mixture is a convex optimization problem. On the other hand, in the context of deep learning, sequential training comes at a great cost in terms of time. Indeed, the sequential nature of boosting-like algorithms requires that each model is fully trained before the subsequent ones start training and is never changed afterwards. Then, the subsequent models are trained to fit the parts of space that the former models could not capture well. As a consequence, there is no guarantee that any of the models would focus on a mode. Indeed, each generator will try to cover the whole residual of the data distribution which is not yet covered by the current mixture.

As opposed to (Tolstikhin et al., 2017), instead of training the mixture of generative models sequentially we train each of them at the same time. By doing so, we loose the convexity of the objective, but, if one is able to decouple the training procedure of each model, each model can be trained in parallel.

3. Training Independent Generative Models

Our approach is informally depicted in Algorithm 1. Borrowing ideas from the clustering literature, at each iteration,

Algorithm 1 Mixture training

1: **init** K generative models g_j, $c_j^{(0)}$
2: **for** $t = 0 \dots T$
3: $\quad \min_{P_{g_j}} D_f\left(P_{g_j} \| c_j^{(t)}(P_d)\right)$ for every g_j in parallel.
4: $\quad$ Update $c_j^{(t+1)}(\mathbf{x})$ for every g_j
5: **end for**

each generative model g_j is trained on a different portion data distribution obtained using a partition function c_j. We assume that each point in $\mathcal{X}$ can be generated by exactly one mechanism. This translates to the assumption that there exists an unknown partition function $c^\star$ that divides the support of P_d into non-overlapping areas which corresponds to the different generating mechanisms. Intuitively, our training procedure is inspired by the EM algorithm. We first train the models (E-step) and then we approximate the posterior distribution (M-step). As in k-means clustering, we have a degenerate assignment, in which each training example is assigned to the model with largest posterior.

Formally, let us consider K assignment functions $c_j(\mathbf{x}) : \mathcal{X} \to \{0, 1\}$. We assume that for any $\mathbf{x} \in \mathcal{X}$ there is a unique j such that $c_j(\mathbf{x}) = 1$ and it is zero for all the others $c_{[K]\setminus j}$. Let us now use the c_j to partition the support X of dP_d: First of all, let us define:

$$c_j(dP_d) := dP_d^j(\mathbf{x}) := \begin{cases} \frac{dP_d(\mathbf{x})}{\int_{\mathsf{X}} dP_d(\mathbf{x}) c_j(\mathbf{x})}, & \text{if } c_j(\mathbf{x}) = 1 \\ 0, & \text{otherwise} \end{cases}$$

Trivially all the followings hold from the definition of c_j:

- $\int_{\mathsf{X}^j} dP_d^j(\mathbf{x}) = 1$ where X^j is the support of dP_d^j
- $\int_{\mathsf{X}} dP_d^j(\mathbf{x}) = 1$ as $dP_d^j(\mathbf{x})$ is zero outside its support
- $\cup_{j\in[K]} \mathsf{X}^j = \mathsf{X}$ and $\mathsf{X}^j \cap \mathsf{X}^l = \emptyset \; \forall \, l \neq j$
- $\sum_{j\in[K]} dP_d^j(\mathbf{x}) = dP_d(\mathbf{x})$ as the supports are all disjoint

For a given c_j we define the weighting α_j as the ratio between the measure of X^j and X.

We now rewrite the f-divergence minimization as:

$$\min_{\substack{c_j, P_{g_j} \\ j\in[K]}} \sum_j \alpha_j D_f(P_{g_j} | P_d^j) \tag{3}$$

Lemma 1. *The objective of the optimization problem in Equation* (3) *is an upper bound of the f-divergence for a mixture model.*

Proof. By definition of the model we write the f-divergence as:

$$D_f(P_{model} \| P_d) = D_f(\sum_{j=1}^k \alpha_j P_{g_j} \| P_d)$$

Now, we have that α_j is the ratio between the measure of the support X^j and X. Since $\mathsf{X}^j \cap \mathsf{X}^k = \emptyset$ for $j \neq k$, we can write:

$$D_f(\sum_{j=1}^k \alpha_j P_{g_j} \| P_d) = D_f(\sum_{j=1}^k \alpha_j P_{g_j} \| \sum_{j=1}^k \alpha_j P_d^j)$$

Joint convexity of D_f concludes the proof. □

Since each term in the sum in Equation (3) is independent, each generative model can be trained independently to approximate:

$$dP_d^{(t)}(\mathbf{x}|g_j) := c_j^{(t)}(dP_d(\mathbf{x}))$$

Intuitively, our goal is that the mixture of P_{g_j}s resembles as much as possible P_d. Therefore, at iteration t, we estimate $c_j^{(t+1)}(\mathbf{x})$ such that the now fixed $P_{g_j}^{(t)}$ is as close as possible to $c_j(P_d)$. In other words:

$$\min_{c_j} \alpha_j D_f \left(P_{g_j}^{(t)} \| c_j(P_d) \right) \tag{4}$$

which can be explained as finding the parts of the X that contains samples that looks similar to the ones produced by $P_{g_j}^{(t)}$. In order to evaluate the similarity between points in X and in the support of our approximation of the j-th generating mechanism we train a discriminator to distinguish samples from a g_j and samples drown from P_d, for which we have that:

$$D_{g_j}^{(t)}(x_i) \approx \frac{dP_d(x_i)}{dP_d(x_i) + dP_{g_j}^{(t)}(x_i)}. \tag{5}$$

After training the classifier, we can rewrite Equation (5) as:

$$dP_{g_j}^{(t)}(x_i) \approx dP_d(x_i) \frac{1 - D_{g_j}^{(t)}(x_i)}{D_{g_j}^{(t)}(x_i)}$$

Note that this relation makes sense only computed on the training points. Indeed, each generator is not perfect, and it might happen that its support is not completely overlapping with X (if not disjoint). On the other hand, in the regions of the space which are not in X, we trivially have that both $dP_d(x_i) = 0$ and $D_j(x_i) = 0$. Furthermore, X is not known and we can only empirically estimate dP_d on the points in the training set $\mathbf{X}$. Therefore, we approximate $dP_{g_j}^{(t)}(x_i)$ as the empirical estimate over the training set:

$$dP_{g_j}^{(t)}(x_i) \approx \frac{1}{Z} \frac{1 - D_{g_j}^{(t)}(x_i)}{D_{g_j}^{(t)}(x_i)}$$

where $Z = \sum_{x\in\mathbf{X}} \frac{1 - D_{g_j}^{(t)}(x_i)}{D_{g_j}^{(t)}(x_i)}$. We now assign each training point to the mechanism j that generates the most similar samples, i.e. the probability of sampling x_i from g_j is larger than the one of sampling it from other mechanisms g_k. If the support of dP_g and dP_d do not coincide we are essentially projecting dP_{g_j} onto the support of the data distribution before maximizing. This yields a bias in the assignment, favoring models with small overlap with the data distribution. Consider an indicator set function $\delta_{\mathsf{X}}(x) = 1$ if $x \in \mathsf{X}$. Then, we estimate $\frac{dP_{g_j}}{\int_{\mathsf{G}_j} \delta_{\mathsf{X}}(x) dP_{g_j}}$. So

we lift the probability of generating x by a factor which is larger the less the support of the generator intersect with X. While this seems counterintuitive it has important benefits during training. In particular, this procedure helps weaker generators which are specialized in a limited portion of X to compete with the strongest ones which are doing a relatively good job over the whole data distribution. In our setting indeed, it is undesirable that a single model covers the whole data distribution as no disentanglement between the generating mechanisms would happen. The model, using VAEs decoders as generators, is depicted in Figure 1

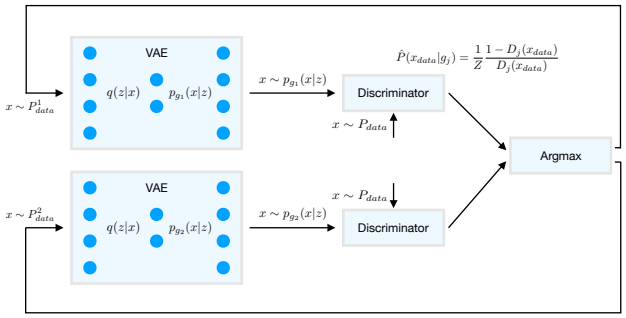

Figure 1: Pipeline

4. Convergence

Let us now fix a particular assignment $c_j = c_j^{(t)}$. As a consequence we write the corresponding portion of the data distribution as $P_d^{j^{(t)}}$. The optimization problem we need to solve then becomes:

$$\min_{\substack{P_{g_j} \\ j\in[K]}} \alpha_j \sum_j D_f(P_{g_j}|P_d^{j^{(t)}})$$

As every term in the sum is independent from each other, we can minimize them separately, which results in training K generative models approximating $dP_d^{j^{(t)}}$.
After training each model in parallel, we fix the generators and re-compute the assignment function based on how realistic are the samples produced by each model. Concretely:

$$c_j^{(t+1)}(\mathbf{x}_i) = \begin{cases} 1, & \text{if } j = \arg\max_l dP_{g_l}^{(t+1)}(\mathbf{x}_i) \\ 0, & \text{otherwise} \end{cases}$$

where $dP_{g_l}^{(t+1)}(\mathbf{x}_i)$ can be approximated by training a classifier between dP_{g_j} and dP_d as in Equation (5) .

Theorem 2. *This choice of c_j minimizes the empirical estimate of the loss in Equation* (3) *for a fixed set of generators with support contained in X and a fixed perfect discriminator.*

Proof. We want to minimize:

$$\sum_j \alpha_j D_f\left(P_{g_j}^{(t)} \| c_j(P_d)\right)$$

which we rewrite using the properties of c and definition of α_j as:

$$\sum_j \alpha_j D_f\left(P_{g_j}^{(t)} \| c_j(P_d)\right) = \sum_j \int_{\mathsf{X}} dP_d c_j \left(f\left(\frac{dP_{g_j}^{(t)}}{dP_d}\right)\right) \tag{6}$$

We can push the finite sum inside the integral, therefore, for each $x \in \mathsf{X}$ we need to find the j that maximize $f\left(\frac{dP_{g_j}^{(t)}}{dP_d}\right)$. Now we turn to the empirical estimate of the loss and exchange the sums:

$$\sum_{x\in\mathbf{X}} \sum_j \frac{1}{N} c_j \left(f\left(dP_{g_j}^{(t)}(x)N\right)\right)$$

Therefore, due to the definition of c_j, we need to find for every x the index j that maximize $dP_{g_j}^{(t)}(x)$ □

Corollary 3. *Since each step at least decreases the objective it holds that:*

$$F(c^{(t)}, P_g^{(t)}) \geq F(c^{(t)}, P_g^{(t+1)}) \geq F(c^{(t+1)}, P_g^{(t+1)}) \geq 0$$

where F is the empirical loss. Therefore, the algorithm converges.

Theorem 4. *The described algorithm converges in finitely many iterations if the non convex optimization problem*

$$\min_{P_{g_j}} F(c_j^{(t)}, P_{g_j})$$

has finitely many local minima with different function value.

Proof. There are only N^K possible assignments for the points in the training set. For each possible assignment there are finitely many solutions of $\min_{P_{g_j}} F(c_j^{(t)}, P_{g_j})$ but they all have to decrease the objective. We need to show that the same pair of assignment function and generator cannot happen twice unless the algorithm has converged. In other words, the sequence of $c^{(t)}$ and $P_g^{(t)}$ does not contain loops. Assume $c_j^{(t+k)} = c_j^{(t)}$ and $P_g^{(t+k)} = P_g^{(t)}$ with $k > 1$ but the algorithm did not converge, therefore there is at least a single $k' \in [1, k-1]$ s.t $F(c_j^{(t+k')}, P_g^{(t+k')}) < F(c_j^{(t+k'-1)}, P_g^{(t+k'-1)})$. This trivially contradicts Corollary 3 as $F(c_j^{(t+k)}, P_g^{(t+k)}) = F(c_j^{(t)}, P_g^{(t)})$. □

Note that convergence in finitely many iteration is guaranteed only if the minimization wrt the generators has only finitely many local minima with different function value.

5. kVAEs

We now particularize the framework using VAEs. One of the reasons to prefer VAEs instead of GANs in this framework is that the adversarial training, with a target that constantly

changes, is very unstable and if a single GAN fails the whole model fails. Another reason to use VAE is to better understand their blurriness. Indeed, one of the explanations for the sharpness of the images generated by GANs is that they tend to collapse on a small number of modes from which the model is able to sample points which looks very realistic. In practice, it can happen that a generative model is not expressive enough to perfectly model a complex distribution. In such a case GANs will focus on a part of the data distribution and ignore the parts which are too difficult to approximate. VAEs instead try to cover the whole data distribution as a consequence of minimizing the KL divergence. Indeed, the vae objective is:

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}\sim P_d}\left[\mathbb{E}_{\mathbf{z}\sim Q}\log P_g(\mathbf{x}|\mathbf{z}) - D^{KL}(Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z}))\right] \tag{7}$$

and after the reparametrization trick with gaussian encoder:

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}\sim P_d}\Big[\mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}\log P_g(\mathbf{x}|\mathbf{z}=\mu(\mathbf{x})+\Sigma^{\frac{1}{2}}(\mathbf{x})\epsilon) - D^{KL}(Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z}))\Big], \tag{8}$$

see (Kingma & Welling, 2013). The aim is to maximize this loss wrt the parameters of the encoders and the generator (decoder). If $P_g(\mathbf{x}|\mathbf{z}) = 0$ for a point in the data distribution the VAE incurs in infinite loss. Therefore, VAEs when used as generative models produce *bridges* between the modes as one can see in Figure 2. While we do not claim that the *only* reason for blurriness are samples out of the support of the data distribution, we argue that it is unlikely that these samples are visually realistic. Furthermore, if one wants to claim that the model approximate the data distribution well, there should be an incentive for the support to be as close as possible to X. In particular, if X is disjoint, P_{model} should also have disjoint support.

In Figure 2 is depicted a VAE trained for 100 epochs whose capacity is the same of the one used in our experimental evaluation.

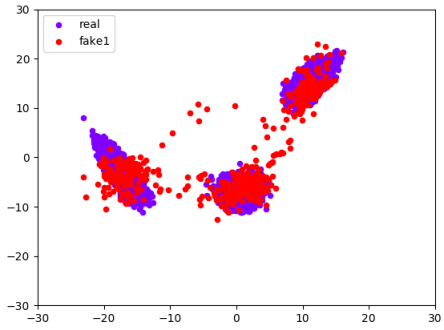

Figure 2: VAE producing samples outside the data distribution

6. Clustering Interpretation

The general approach we introduced is inspired by clustering literature (Aggarwal & Reddy, 2013). In this section we revisit classical clustering notions under the lights of our framework. In the generative interpretation of clustering, one assumes that the data was generated from each centroid by sampling gaussian noise vector. We can say that $x = c+\epsilon$ where c is the centroid. This formulation naturally yields an euclidean cost for the cluster assignment when decoupling the data between the different centroids. Unfortunately, the euclidean distance is known not to be a good metric in a non-euclidean space such as the pixel space. Our goal is to find a clustering of the data across the generating mechanism in a setting in which the metric of the space is not known. We now show how to recover the k-means clustering from our framework. Assume that the data is generated by a mixture of gaussians. We can lower bound the log likelihood of the data using a variational bound:

$$\log(P(X)) \geq \sum_n \sum_{z_i} q_n(z_i)\log\left(\frac{P(x_n, z_i)}{q_n(z_i)}\right)$$

where q is the variational distribution. One can then simply rewrite $P(x_n, z_i) = P(x_n|z_i)p(z_i)$. Then, for a gaussian mixture model one parametrize $P(x_n|z_i)$ with a gaussian distribution. If the gaussian is isometric, when the covariance vanishes one obtains that $q_n(z_i)$, the variational approximation of the posterior, degenerates to a hard assignment. Instead of approximating the generative model with a gaussian distribution, we parametrize $P(x_n|z_i)$ with an implicit generative model from which is easy to sample. If P_{g_i} is the decoder of a variational autoencoder, we can obtain the k-means algorithm by assuming that the mean of the encoder is constant and independent of x. The latent space of the variational autoencoder has the same dimensionality of the input but is independent from the input. Then we have a gaussian decoder with no covariance. The result of this procedure is that when training the autencoder with the reparametrization trick one has to minimize:

$$\mathbb{E}_{x\sim c_j(P_d)} - \log P_g(x|z=\mu) = \frac{1}{2}\|x-\mu\|^2$$

The only parameters to optimize are the biases of the encoder, i.e. μ for the assigned points. Then, using EM, we compute the update for the (degenerate) variational distribution:

$$q_n(z_i = 1) = \lim_{\sigma\to 0}\frac{\alpha e^{-\|x_n-\mu_i\|/2\sigma}}{\sum_j \alpha_j e^{-\|x_n-\mu_j\|/2\sigma}}$$

And recalling that $\log P_{g_i} = -\|x_n-\mu_i\|^2/2$ we notice that our assignment update corresponds in this case to maximizing the posterior assignment as typically done in k-means. Instead of using the autencoder loss, we estimate P_{g_j} using a discriminator to account for the fact that we might not have a clear notion of distance. When testing the model (i.e. generating samples from the centroids) we simply input values from a normal distribution to the gaussian decoder and $x = \mu + \epsilon$.

7. Related Work

8. Experimental Proof of Concept

In this section we evaluate the proposed framework. We want to test if the model can successfully learn disentangled generative mechanisms for the data. **FL:** add more on the goal of the experiments

8.1. Synthetic Data

We generate synthetic data in 2 dimension by first sampling 64000 points from a gaussian distribution and then we skew the second dimension x_2 with the non linear transformation $x_2 = x_2 + 0.04 * x_1^2 - 100 * 0.04$. This ensures that each mode is sufficiently complex so that the VAEs can not perfectly learn the data distribution by encoding each mode on a different dimension of the latent space. We use a small and standard architecture for the VAE: a neural network with two hidden layers with 500 units each as both decoder, encoder and discriminator. We use a 5 dimensional latent space and assume a gaussian encoder. At each iteration, we train each VAE for 10 epochs on a split of the dataset (VAEs are pretrained uniformly on the dataset) and the classifier is trained for 2 epochs for the first two experiments. We use adam with step size 0.005, $\beta = 0.5$ batch size 32 and batch norm with decay 0.9. We perform 3 different experiments, 3 modes (and 3 VAEs), 5 modes (and 5 VAEs) in which one mode is actually composed of two separate modes which are very close and 9 modes (with 9 VAEs). We run the algorithm for 100 iterations. We note that in the first two experiments each model perfectly cover a single mode, even in the case in which there is a mode which is distributed differently. In the last experiments, each mode contains significantly fewer points than the previous ones. In this case, we first train each VAE for 1000 epochs. We note that even after training for so long the models perform very poorly, because the data is far too complex for such simple models. Then, we run our algorithm on the pretrained models and each model is reatrained for 10 epochs on the split given by the classifier. Surprisingly, after only 10 iterations of the algorithm the model splits covering only limited parts of the data distribution. We still observe that some model still try to approximate multiple modes yielding samples outside the support of the data distribution. Note that the only incentive for the model to split is the competitive procedure which we use to assign the data to each VAE. **FL:** comment the log likelihood

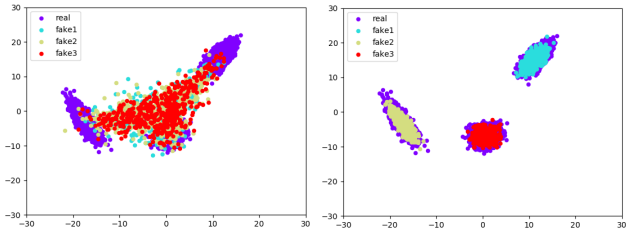

Figure 3: Synthetic data experiment, 3 modes, uniform training and after 100 iterations of the algorithm.

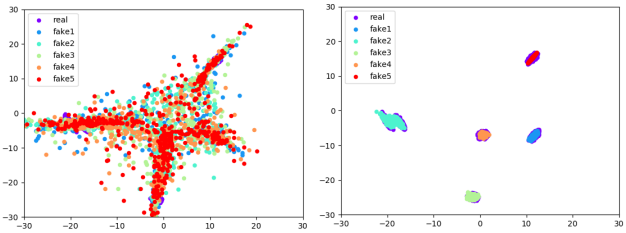

Figure 4: Synthetic data experiment, 5 modes, uniform training and after 100 iterations of the algorithm.

8.2. MNIST

For the experiment on MNIST we do not know the number of modes. Therefore we arbitrarily fix 15 models. We decided to use more than 10 models as, following the insights from (Tolstikhin et al., 2017), we were targeting stylistic difference between digits. Indeed, different modes might not be coincides with different classes. We again use a small and simple architecture, with relu activation functions. The encoder has 4 convolutional layers with 8-16-32-64 filters with 4 dimensions each. The decoder has 4 deconvolutional layers, with the same number of filters as the decoder. We use batch normalization with epsilon $1e-5$ and decay 0.9. Each VAE has latent space dimension of 8 and we fix the learning rate of Adam for all networks to 0.005. The discriminator is has 3 convolutional layers and a linear layer with number of filters 64-128-256. As opposed to the synthetic data example, we do not reinitialize the classifier each iteration. Instead, we train it for a single batch every iteration. The reason is that we found the classifier output to be too sensitive to the initialization if it is not trained long enough. On the other hand, training a full discriminator every iteration was too expensive and if trained too much it would learn to distinguish fake example by just looking at specific blurriness patterns. On the other hand, the data produced by the VAE in the synthetic experiments was indistinguishable from the real data, so training a classifier from scratch was feasible and gave best results. In Figure 6 we show the number of training samples assigned to each model divided by digit and in Figure 7 samples from each decoder. We highlight that the different VAEs did separate the data and specialized on different parts of the data distribution. We notice that similar digits tends to be grouped together (for example 3 and 8, 6 and 4) as well as similar styles (tilted, thin, large, round, bold and combinations thereof). Since it is not clear what the modes are supposed to be, we also show samples from the whole mixture and notice that there is a large variety of different styles. **FL:** Comment the log likelihood

experiment	kVAEs train/test	bag train/test	VAE-50 train/test	VAE-150 train/test
gmm 3 modes	**-3.36 / -3.47**	TODO/TODO	-6.21 / 6.08	-5.79 / -5.92
gmm 5 modes	**-1.25 / -1.29**	TODO/TODO	-5.38 / -5.54	-5.39 / -5.66
gmm 9 modes	**-0.97 / -0.99**	TODO/ TODO	-4.08 / -4.04	-3.95 / -4.23

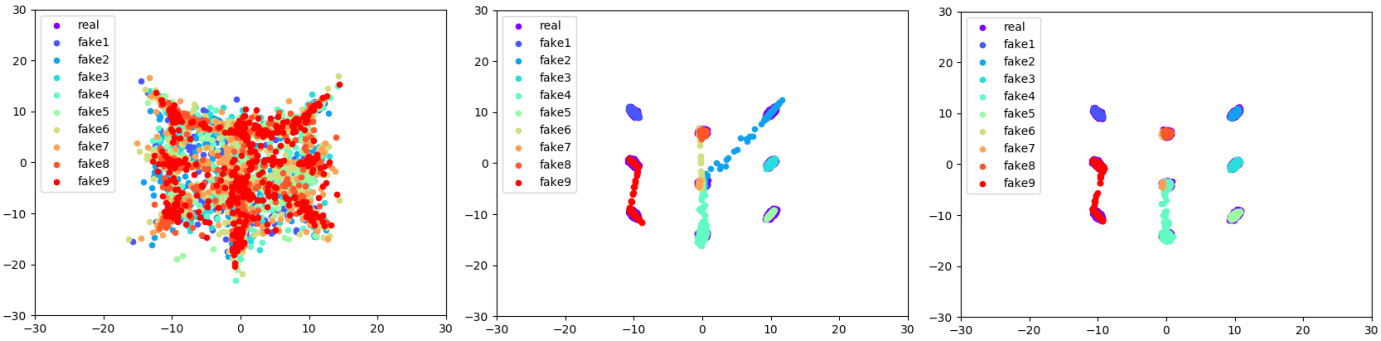

Figure 5: Synthetic data experiment, 9 modes, uniform training and after 10 and 100 iterations of the algorithm.

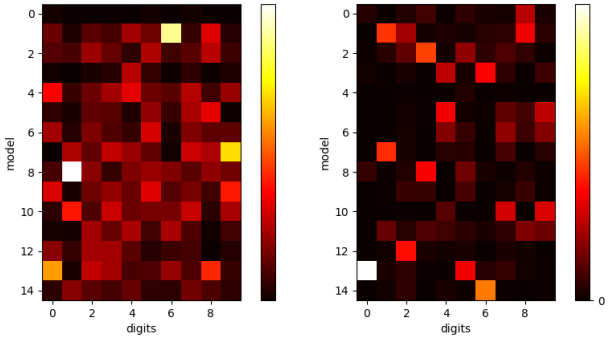

Figure 6: MNIST assignments: how many points with the same label are assigned to each model

Selecting K Selecting the number of models is a question which does not have a trivial answer. It is not reasonable to assume that one can fix a large K and the extra models will just be defeated in the competitive procedure. Indeed, as in k-means clustering, if K is too large we would simply fit more models which in turns will yield to an unstable clustering. On the other hand, the extra models will have very few points assigned, thus the mixture will contain only a few of their samples. For these experiments, we fixed a priori the number of VAEs without tuning it as we wanted to demonstrate that the algorithm behaves as expected. Solutions for finding K involving stability of the clustering are unfortunately unfeasible (as running the algorithm several times is time consuming). Finding an efficient technique to estimate K while the algorithm run is an interesting future direction.

Figure 7: MNIST: samples separated per model

Figure 8: MNIST: samples from the mixture

kVAEs train/test	bag train/test	VAE-64 train/test	VAE-512 train/test	VAE-1024 train/test
TODO / TODO	TODO / TODO	- 3.12e+03/ - 3.28e+03	-1.63e+04 / -5.32e+04	-1.64e+04/ -5.34e+04

References

Aggarwal, Charu C and Reddy, Chandan K. *Data clustering: algorithms and applications*. CRC press, 2013.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Hoang, Quan, Nguyen, Tu Dinh, Le, Trung, and Phung, Dinh. Multi-generator gernerative adversarial nets. *arXiv preprint arXiv:1708.02556*, 2017.

Janzing, Dominik and Scholkopf, Bernhard. Causal inference using the algorithmic markov condition. *IEEE Transactions on Information Theory*, 56(10):5168–5194, 2010.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Nowozin, Sebastian, Cseke, Botond, and Tomioka, Ryota. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pp. 271–279, 2016.

Peters, J., Janzing, D., and Schölkopf, B. *Elements of Causal Inference - Foundations and Learning Algorithms*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, MA, USA, 2017.

Schölkopf, Bernhard, Janzing, Dominik, Peters, Jonas, Sgouritsa, Eleni, Zhang, Kun, and Mooij, Joris. On causal and anticausal learning. *arXiv preprint arXiv:1206.6471*, 2012.

Tolstikhin, Ilya O, Gelly, Sylvain, Bousquet, Olivier, Simon-Gabriel, Carl-Johann, and Schölkopf, Bernhard. AdaGAN - Boosting Generative Models. *CoRR*, 2017.