Adversarially Learned Mixture Model

Andrew Jesson
Imagia Cybernetics Inc.
Montreal, QC, Canada
andrew.jesson@imagia.com

Cécile Low-Kam
Imagia Cybernetics Inc.
Montreal, QC, Canada
cecile.low-kam@imagia.com

Florian Soudan
Imagia Cybernetics Inc.
Montreal, QC, Canada
florian@imagia.com

Nicolas Chapados
Imagia Cybernetics Inc.
Montreal, QC, Canada
nic@imagia.com

Abstract

The Adversarially Learned Mixture Model (AMM) is a generative model for unsupervised or semi-supervised data clustering. The AMM is the first adversarially optimized method to model the conditional dependence between inferred continuous and categorical latent variables. Experiments on the MNIST and SVHN datasets show that the AMM allows for semantic separation of complex data when little or no labeled data is available. The AMM achieves a state-of-the-art unsupervised clustering error rate of 2.86% on the MNIST dataset. A semi-supervised extension of the AMM yields competitive results on the SVHN dataset.

1 Introduction

Semi-supervised or unsupervised representation learning enables the utilization of all available data when tackling problems where there are little or no labeled examples. This is a common scenario in many applications of machine learning, such as medical image analysis, where it is reinforced by the expense of obtaining expert labeled examples. Moreover, machine-learned representations are more likely to be used for subsequent tasks if they are interpretable and meaningful. Deep generative modelling is a suitable approach to this problem, as derived models have been shown capable of learning from both labeled and unlabeled examples, embedding data according to desired latent variable distributions, and producing realistic data examples generated from samples of those latent variables.

The Generative Adversarial Network (GAN) has recently emerged as a powerful framework for modeling complex data distributions without having to approximate intractable likelihoods. In the formulation by Goodfellow et al. (2014), a GAN consists of two networks: a *generator* G that is trained to yield unique samples from the data distribution, and a *discriminator* D that is trained to distinguish between generated and true data samples.

Dumoulin et al. (2016) and Donahue et al. (2017) have proposed the ALI and BiGAN models that add an inference process, i.e., the ability to map data samples to points in the latent space, to the GAN framework. A second generator for inference, or *encoder*, is added to the original GAN generator and the discriminator is adapted for the two-dimensional space of data inputs and latent representations. A variant of the resulting model is also introduced by Dumoulin et al. (2016) for conditional data generation, but still assumes that the class of the data is always observed, as inference of categorical variables is not included.

Adversarial approaches for the inference of both continuous and categorical variables are actively researched. Chen et al. (2016) introduce a hybrid adversarial method that is capable of modelling both continuous and categorical latent variables for unsupervised clustering and feature disentanglement.

Preprint. Work in progress.

Another hybrid adversarial method is introduced by Makhzani et al. (2016) where adversarial objectives on continuous and categorical latent variables are optimized for unlabeled examples and categorical cross entropy on categorical variables is optimized for labeled examples. Li et al. (2017) and Deng et al. (2017) point toward fully adversarial semi-supervised classification using inferred categorical variables by introducing a "three player" adversarial game, but stop short by adding auxiliary "collaborative" objectives. In each of these methods, it is assumed that categorical and continuous latent variables are independently distributed. This independence assumption results in discontinuities in the latent space between categories, which removes the notion of inter-categorical proximity.

Another notable family of generative models, Variational Autoencoders (VAEs), maximize the posterior distribution of latent representations given the data instead of using an adversarial approach. As VAEs integrate inference, semi-supervised classification can be performed by conditioning the continuous latent variable of the VAE on the class label (Kingma et al., 2014; Dilokthanakul et al., 2016; Maaløe et al., 2017). However, the quality of VAE results depend on the expressiveness of the inference distribution and every time the assumptions about the inference or data distributions are changed a new objective function needs to be derived. In this way, variational optimization is not as versatile as adversarial training.

We present the Adversarially Learned Mixture Model (AMM). The AMM is, to our knowledge, the first generative model inferring both continuous and categorical latent variables to perform either unsupervised or semi-supervised clustering of data using a single adversarial objective. This is enabled, in part, by explicitly modelling the dependence between continuous and categorical latent variables, which eliminates discontinuities between categories in the latent space. Semi-supervised clustering and classification is enabled by a simplified formulation of the "three player game", presented by Li et al. (2017). In this paper we show that the AMM achieves state of the art unsupervised clustering error rate on the MNIST dataset (LeCun & Cortes, 2010), and that it achieves competitive results for semi-supervised classification on the SVHN dataset (Netzer et al., 2011).

2 Method

2.1 Preliminaries

The ALI and BiGAN models are trained by matching two joint distributions of images $\boldsymbol{x} \in \mathbb{R}^D$ and their latent code $\boldsymbol{z} \in \mathbb{R}^L$. The two distributions to be matched are the inference distribution $q(\boldsymbol{x},\boldsymbol{z})$ and the synthesis distribution $p(\boldsymbol{x},\boldsymbol{z})$, where,

$$q(\boldsymbol{x},\boldsymbol{z}) \quad = \quad q(\boldsymbol{x})q(\boldsymbol{z}\,|\,\boldsymbol{x}), \tag{1}$$
$$p(\boldsymbol{x},\boldsymbol{z}) \quad = \quad p(\boldsymbol{z})p(\boldsymbol{x}\,|\,\boldsymbol{z}). \tag{2}$$

Samples of $q(\boldsymbol{x})$ are drawn from the training data and samples of $p(\boldsymbol{z})$ are drawn from a prior distribution, usually $\mathcal{N}(0,1)$. Samples from $q(\boldsymbol{z}\,|\,\boldsymbol{x})$ and $p(\boldsymbol{x}\,|\,\boldsymbol{z})$ are drawn from neural networks that are optimized during training. Dumoulin et al. (2016) show that sampling from $q(\boldsymbol{z}\,|\,\boldsymbol{x})=\mathcal{N}(\mu(\boldsymbol{x}),\sigma^2(\boldsymbol{x})I)$ is possible by employing the reparametrization trick (Kingma & Welling, 2013), i.e. computing

$$\boldsymbol{z}=\mu(\boldsymbol{x})+\sigma(\boldsymbol{x})\odot\epsilon, \quad \epsilon\sim\mathcal{N}(0,I), \tag{3}$$

where $\odot$ is element wise vector multiplication.

A conditional variant of ALI has also been explored by Dumoulin et al. (2016) where an observed class-conditional categorical variable $\boldsymbol{y}$ has been introduced. The joint factorization of each distribution to be matched are:

$$q(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \quad = \quad q(\boldsymbol{x},\boldsymbol{y})q(\boldsymbol{z}\,|\,\boldsymbol{y},\boldsymbol{x}), \tag{4}$$
$$p(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \quad = \quad p(\boldsymbol{y})p(\boldsymbol{z})p(\boldsymbol{x}\,|\,\boldsymbol{y},\boldsymbol{z}). \tag{5}$$

Samples of $q(\boldsymbol{x},\boldsymbol{y})$ are drawn from the data. Samples of $p(\boldsymbol{z})$ are drawn from a continuous prior on $\boldsymbol{z}$, and samples of $p(\boldsymbol{y})$ are drawn from a categorical prior on $\boldsymbol{y}$, both of which are marginally independent. Samples from $q(\boldsymbol{z}\,|\,\boldsymbol{y},\boldsymbol{x})$ and $p(\boldsymbol{x}\,|\,\boldsymbol{y},\boldsymbol{z})$ are drawn from neural networks that are optimized during training.

In the following sections we present graphical models for $q(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ and $p(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ that build off of conditional ALI. Where conditional ALI requires the full observation of categorical variables, the models we present will account for both unobserved and partially observed categorical variables. We finally show how they can be optimized using a single adversarial objective.
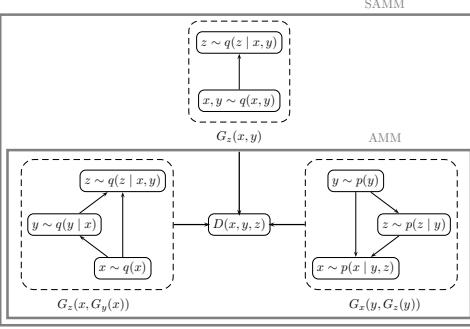
Figure 1: Overview of the unsupervised (AMM) and semi-supervised (SAMM) model with the first option (Equation (6)) for the inference distribution. AMM consists of two generators, encoder $G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))$ and decoder $G_{\boldsymbol{x}}(\boldsymbol{y},\boldsymbol{z})$, and a discriminator $D(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$. SAMM includes an additional generator for labeled data, $G_z(\boldsymbol{x},\boldsymbol{y})$.

2.2 Adversarially Learned Mixture Model

The AMM is an adversarial generative model for deep unsupervised clustering of data. Figure 1 presents an overview of the model.

Like conditional ALI, a categorical variable is introduced to model the labels. However, the unsupervised setting now requires a different factorization of the inference distribution in order to enable inference of the categorical variable $\boldsymbol{y}$, namely:

$$q_1(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})=q(\boldsymbol{x})q(\boldsymbol{y}\,|\,\boldsymbol{x})q(\boldsymbol{z}\,|\,\boldsymbol{x},\boldsymbol{y}), \tag{6}$$

or

$$q_2(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})=q(\boldsymbol{x})q(\boldsymbol{z}\,|\,\boldsymbol{x})q(\boldsymbol{y}\,|\,\boldsymbol{x},\boldsymbol{z}). \tag{7}$$

Samples of $q(\boldsymbol{x})$ are drawn from the training data, and samples from $q(\boldsymbol{y}\,|\,\boldsymbol{x})$, $q(\boldsymbol{z}\,|\,\boldsymbol{x},\boldsymbol{y})$ or $q(\boldsymbol{z}\,|\,\boldsymbol{x})$, $q(\boldsymbol{y}\,|\,\boldsymbol{x},\boldsymbol{z})$ are generated by neural networks. We follow Kendall & Gal (2017) and sample from $q(\boldsymbol{y}\,|\,\boldsymbol{x})$ by computing

$$h_{\boldsymbol{y}}(\boldsymbol{x}) \;=\; \mu_{\boldsymbol{y}}(\boldsymbol{x})+\sigma_{\boldsymbol{y}}(\boldsymbol{x})\odot\epsilon, \quad \epsilon\sim\mathcal{N}(0,I), \tag{8}$$
$$y(\boldsymbol{x}) \;=\; \text{softmax}(h_{\boldsymbol{y}}(\boldsymbol{x})). \tag{9}$$

Then, we can sample from $q(\boldsymbol{z}\,|\,\boldsymbol{x},\boldsymbol{y})$ by computing

$$z(\boldsymbol{x},h_{\boldsymbol{y}}(\boldsymbol{x}))=\mu_{\boldsymbol{z}}(\boldsymbol{x},h_{\boldsymbol{y}}(\boldsymbol{x}))+\sigma_{\boldsymbol{z}}(\boldsymbol{x},h_{\boldsymbol{y}}(\boldsymbol{x}))\odot\epsilon, \quad \epsilon\sim\mathcal{N}(0,I). \tag{10}$$

A similar sampling strategy can be used to sample from $q(\boldsymbol{y}\,|\,\boldsymbol{x},\boldsymbol{z})$ in (7).

The factorization of the synthesis distribution $p(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ also differs from conditional ALI:

$$p(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})=p(\boldsymbol{y})p(\boldsymbol{z}\,|\,\boldsymbol{y})p(\boldsymbol{x}\,|\,\boldsymbol{y},\boldsymbol{z}). \tag{11}$$

The product $p(\boldsymbol{y})p(\boldsymbol{z}\,|\,\boldsymbol{y})$ can be conveniently given by a mixture model. Samples from $p(\boldsymbol{y})$ are drawn from a multinomial prior, and samples from $p(\boldsymbol{z}\,|\,\boldsymbol{y})$ are drawn from a continuous prior, for example, $\mathcal{N}(\mu_{\boldsymbol{y}=k},1)$. Samples from $p(\boldsymbol{z}\,|\,\boldsymbol{y})$ can alternatively be generated by a neural network by again employing the reparameterization trick. Namely,

$$z(\boldsymbol{y})=\mu(\boldsymbol{y})+\sigma(\boldsymbol{y})\odot\epsilon, \quad \epsilon\sim\mathcal{N}(0,I). \tag{12}$$

This approach effectively learns the parameters of $\mathcal{N}(\mu_{\boldsymbol{y}=k},\sigma_{\boldsymbol{y}=k})$.

2.2.1 Adversarial Value Function

We follow Dumoulin et al. (2016) and define the value function that describes the unsupervised game between the discriminator D and the generator G as:

$$
\begin{aligned}
\min_G \max_D V(D,G) &= \mathbb{E}_{q(\boldsymbol{x})}[\log(D(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}),G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))))] \\
&\quad + \mathbb{E}_{p(\boldsymbol{y},\boldsymbol{z})}[\log(1-D(G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y})),\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y})))] \\
&= \iiint q(\boldsymbol{x})q(\boldsymbol{y}\,|\,\boldsymbol{x})q(\boldsymbol{z}\,|\,\boldsymbol{x},\boldsymbol{y})\log(D(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))d\boldsymbol{x}d\boldsymbol{y}d\boldsymbol{z} \\
&\quad + \iiint p(\boldsymbol{y})p(\boldsymbol{z}\,|\,\boldsymbol{y})p(\boldsymbol{x}\,|\,\boldsymbol{y},\boldsymbol{z})\log(1-D(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))d\boldsymbol{x}d\boldsymbol{y}d\boldsymbol{z}.
\end{aligned}
\tag{13}
$$

There are four generators in total: two for the encoder $G_{\boldsymbol{y}}(\boldsymbol{x})$ and $G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))$, which map the data samples to the latent space; and two for the decoder $G_{\boldsymbol{z}}(\boldsymbol{y})$ and $G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))$, which map samples from the prior to the input space. $G_{\boldsymbol{z}}(\boldsymbol{y})$ can either be a learned function, or be specified by a known prior. See Algorithm 1 for a detailed description of the optimization procedure.

Algorithm 1 AMM training procedure using distributions (6) and (11).

$\theta_{G_{\boldsymbol{y}}(\boldsymbol{x})},\theta_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))},\theta_{G_{\boldsymbol{z}}(\boldsymbol{y})},\theta_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))},\theta_D$ ▷ Initialize AMM parameters

while not done **do**

    $\boldsymbol{x}^{(1)},...,\boldsymbol{x}^{(M)} \sim q(\boldsymbol{x})$ ▷ Sample from data and priors

    $\boldsymbol{y}^{(1)},...,\boldsymbol{y}^{(M)} \sim p(\boldsymbol{y})$

    $\boldsymbol{z}^{(j)} \sim p(\boldsymbol{z}\,|\,\boldsymbol{y}=\boldsymbol{y}^{(j)}), \quad j=1,...,M$

    $\tilde{\boldsymbol{x}}^{(j)} \sim p(\boldsymbol{x}\,|\,\boldsymbol{y}=\boldsymbol{y}^{(j)},\boldsymbol{z}=\boldsymbol{z}^{(j)}), \quad j=1,...,M$ ▷ Sample from conditionals

    $\tilde{\boldsymbol{y}}^{(i)} \sim q(\boldsymbol{y}\,|\,\boldsymbol{x}=\boldsymbol{x}^{(i)}), \quad i=1,...,M$

    $\tilde{\boldsymbol{z}}^{(i)} \sim q(\boldsymbol{z}\,|\,\boldsymbol{x}=\boldsymbol{x}^{(i)},\boldsymbol{y}=\tilde{\boldsymbol{y}}^{(i)}), \quad i=1,...,M$

    $\rho_q^{(i)} \leftarrow D(\boldsymbol{x}^{(i)},\tilde{\boldsymbol{y}}^{(i)},\tilde{\boldsymbol{z}}^{(i)}), \quad i=1,...,M$ ▷ Compute discriminator predictions

    $\rho_p^{(j)} \leftarrow D(\tilde{\boldsymbol{x}}^{(j)},\boldsymbol{y}^{(j)},\boldsymbol{z}^{(j)}), \quad j=1,...,M$

    $\mathcal{L}_D \leftarrow -\frac{1}{M}\sum_{i=1}^{M}\log(\rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}log(1-\rho_p^{(j)})$ ▷ Compute discriminator losses

    $\mathcal{L}_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))} = \mathcal{L}_{G_{\boldsymbol{z}}(\boldsymbol{y})} \leftarrow -\frac{1}{M}\sum_{i=1}^{M}\log(\rho_p^{(i)})$ ▷ Compute x generator losses

    $\mathcal{L}_{G_{\boldsymbol{y}}(\boldsymbol{x})} = \mathcal{L}_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))} \leftarrow -\frac{1}{M}\sum_{i=1}^{M}\log(1-\rho_q^{(i)})$ ▷ Compute y and z generator loss

    $\theta_D \leftarrow \theta_D - \nabla_{\theta_D}\mathcal{L}_D$ ▷ Update discriminator parameters

    $\theta_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))} \leftarrow \theta_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))} - \nabla_{\theta_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))}}\mathcal{L}_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))}$ ▷ Update generator parameters

    $\theta_{G_{\boldsymbol{z}}(\boldsymbol{y})} \leftarrow \theta_{G_{\boldsymbol{z}}(\boldsymbol{y})} - \nabla_{\theta_{G_{\boldsymbol{z}}(\boldsymbol{y})}}\mathcal{L}_{G_{\boldsymbol{z}}(\boldsymbol{y})}$

    $\theta_{G_{\boldsymbol{y}}(\boldsymbol{x})} \leftarrow \theta_{G_{\boldsymbol{y}}(\boldsymbol{x})} - \nabla_{\theta_{G_{\boldsymbol{y}}(\boldsymbol{x})}}\mathcal{L}_{G_{\boldsymbol{y}}(\boldsymbol{x})}$

    $\theta_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))} \leftarrow \theta_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))} - \nabla_{\theta_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))}}\mathcal{L}_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))}$

2.3 Semi-Supervised Adversarially Learned Mixture Model

The Semi-Supervised Adversarially Learned Mixture Model (SAMM) is an adversarial generative model for supervised or semi-supervised clustering and classification of data. The objective for training SAMM involves two adversarial games to match pairs of joint distributions. The supervised game matches inference distribution (4) to synthesis distribution (11) and is described by the following value function:

$$
\begin{aligned}
\min_G \max_D V(D,G) &= \mathbb{E}_{q(\boldsymbol{x},\boldsymbol{y})}[\log(D(\boldsymbol{x},\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{x},\boldsymbol{y})))] + \mathbb{E}_{p(\boldsymbol{y},\boldsymbol{z})}[\log(1-D(G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y})),\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y})))] \\
&= \iiint q(\boldsymbol{x},\boldsymbol{y})q(\boldsymbol{z}\,|\,\boldsymbol{x},\boldsymbol{y})\log(D(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))d\boldsymbol{x}d\boldsymbol{y}d\boldsymbol{z} \\
&\quad + \iiint p(\boldsymbol{y})p(\boldsymbol{z}\,|\,\boldsymbol{y})p(\boldsymbol{x}\,|\,\boldsymbol{y},\boldsymbol{z})\log(1-D(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))d\boldsymbol{x}d\boldsymbol{y}d\boldsymbol{z}.
\end{aligned}
\tag{14}
$$

The unsupervised game matches either of the inference distributions, (6) or (7) to the synthesis distribution (11). In the case using distribution (6), the unsupervised game is described by (13).

The generator for semi-supervised learning has three components: encoders $G_{\boldsymbol{z}}(\boldsymbol{x}, G_{\boldsymbol{y}}(\boldsymbol{x}))$ and $G_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})$ map the labeled and unlabeled data samples, respectively, to the latent space, and a decoder $G_{\boldsymbol{x}}(\boldsymbol{y}, G_{\boldsymbol{z}}(\boldsymbol{y}))$ maps samples of $\boldsymbol{y}$ and $\boldsymbol{z}$ to the input space, where $G_{\boldsymbol{z}}(\boldsymbol{z})$ can either be a learned function or be specified by a prior. The encoder for labeled data again consists of two generators (Figure 1). A detailed description of the training algorithm is given in algorithm 2 of the appendix. In practice, optimization of each of the generators and the discriminator can be done simultaneously for both the unsupervised and semi-supervised updates.

3 Related Works

Unsupervised clustering using hybrid adversarial approaches are proposed by both Makhzani et al. (2016) (AAE) and Chen et al. (2016) (InfoGAN). For AAE, the synthesis generator is optimized by minimizing the per-example L2 loss between between input data $\{\boldsymbol{x}_i\}$ and their reconstructions $\{\dot{\boldsymbol{x}}_i = G_{\boldsymbol{x}_i}(G_{\boldsymbol{y}}(\boldsymbol{x}_i), G_{\boldsymbol{z}}(\boldsymbol{x}_i))\}$, while the inference generator is optimized using both the L2 objective and an adversarial objective. For InfoGAN, the inference generator is optimized by maximizing the per-example Mutual Information (MI) between samples of categorical latent variables $\{\boldsymbol{y}_i \sim p(\boldsymbol{y})\}$ and continuous latent variables $\{\boldsymbol{z}_i \sim p(\boldsymbol{z})\}$ and their "reconstructions" $\{\{\dot{\boldsymbol{y}}_i, \dot{\boldsymbol{z}}_i\} = G_{\boldsymbol{y}, \boldsymbol{z}}(G_{\boldsymbol{x}}(\boldsymbol{y}_i, \boldsymbol{z}_i))\}$, while the synthesis generator is optimized using both the MI objective and an adversarial objective.

On the other end of the generative spectrum, Dilokthanakul et al. (2016) and Jiang et al. (2016) offer non-adversarial, VAE-based approaches for unsupervised clustering. Like in the AMM, the combination of priors for the latent variables $\boldsymbol{y}$ and $\boldsymbol{z}$ is modeled as a Gaussian mixture model, where $\boldsymbol{y}$ corresponds to the mixture components.

Multiple adversarial methodologies have been proposed for supervised or semi-supervised learning (Springenberg, 2015; Salimans et al., 2016; Miyato et al., 2017), but they suffer from the same limitation as the original GAN: they do not provide inference. Gan et al. (2017), Li et al. (2017) and Deng et al. (2017) introduce a third player to the adversarial game. Although this extra player allows to infer categorical variables, these approaches are not fully adversarial as auxiliary "collaborative" terms are added to the objective function. Moreover, categorical and continuous latent variables are modeled independently.

The adversarial and hybrid-adversarial approaches thus far discussed all model $\boldsymbol{y}$ and $\boldsymbol{z}$ as being conditionally independent from each other. This may be an ideal prior structure for inference, for example, in learning disentangled representations of $\boldsymbol{x}$ sampled from a limited domain (Chen et al., 2016). However, the independence assumption cannot account for the notion of proximity between categories because $\boldsymbol{z}$ is identically distributed for each category in $\boldsymbol{y}$. Therefore, the distance between categories is equal and indeterminate. AMM and SAMM are presented as adversarial approaches to model conditional dependencies between $\boldsymbol{y}$ and $\boldsymbol{z}$, but they do not preclude the independence assumption. The proposed methods can model $\boldsymbol{y}$ and $\boldsymbol{z}$ as conditionally independent with inference distribution

$$q(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = q(\boldsymbol{x})q(\boldsymbol{y} \mid \boldsymbol{x})q(\boldsymbol{z} \mid \boldsymbol{x}), \qquad (15)$$

and synthesis distribution

$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{y})p(\boldsymbol{z})p(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{z}); \qquad (16)$$

however, analysis of this graphical model is left for future work.

4 Evaluation

AMM and SAMM are evaluated using two image datasets: MNIST (LeCun & Cortes, 2010) and SVHN (Netzer et al., 2011). The provided training and testing splits are used for MNIST experiments with 5000 randomly selected examples left out of the training set for validation. The same training, testing, and validation splits as Dumoulin et al. (2016) are used for SVHN. Preprocessing is limited to scaling image intensities on the range $[0,1]$. Detailed architectures for each experiment are shown in figure 6 of the appendix. We optimize all networks using Adam (Kingma & Ba, 2014) with $\alpha = 0.0002$ and $\beta_1 = 0.5$. All kernel weights are initialized using a Gaussian distribution with standard deviation 0.02, all biases are initialized to 0.0.

4.1 Gradient Penalty

The gradient penalty introduced by Gulrajani et al. (2017) is added to the discriminator loss to help stabilize training of AMM and SAMM models. This penalty keeps the gradients of the discriminator with respect to the inputs x, y, and z on the same order of magnitude. The penalty applied to the discriminator loss is

$$\mathcal{L}_{\nabla_{\hat{x},\hat{y},\hat{z}}} = \lambda \mathbb{E}_{(\hat{x},\hat{y},\hat{z}) \sim \mathbb{P}_{\hat{x},\hat{y},\hat{z}}} \left[(||\nabla_{\hat{x},\hat{y},\hat{z}} D(\hat{x},\hat{y},\hat{z})||_2 - 1)^2 \right], \tag{17}$$

where points $(\hat{x},\hat{y},\hat{z})$ are drawn at random on straight lines between real or prior samples (x,y,z) and synthesized or inferred samples $(\tilde{x},\tilde{y},\tilde{z})$. The gradient penalty for Jensen-Shannon GAN introduced by Roth et al. (2017) has also been explored, but did not produce better results. The regularization term is set to $\lambda = 10.0$, and $\lambda = 0.01$ for MNIST and SVHN experiments, respectively.

4.2 MNIST

In this section, the AMM is evaluated on the task of unsupervised clustering of hand-drawn digits using the MNIST dataset. To model $p(y)p(z \mid y)$, a 10-component, 64 dimensional mixture of Gaussians is used. A multinomial prior is used for $p(y)$ with uniform probability for each class. The means of the component distributions are learned using the reparameterization trick via (12), and the variance for each distribution is fixed to unit value. Table 1 reports the test-set clustering error-rate mean and variance over 5 trials. The AMM achieves 2.86 ± 0.46 percent error rate, which is an improvement over the state-of-the-art. Figure 2 shows visualizations of results from 1 of the 5 trials.

Figure 2: Unsupervised clustering of MNIST data with 10 mixture components. *(a)* Comparing test image membership and randomly generated digits for each mixture component. *(b)* Cluster matrix: rows correspond to true test labels, and columns correspond to component membership. *(c)* Reconstructions of input images: original data on the left of each pair. *(d)* Interpolation between examples: original data samples are shown in the first and last columns with linearly interpolated generations between. *(e)* t-SNE projection of testing samples, color-coded for the MNIST class labels (0 to 9).

Table 1: Test set clustering error rate and standard deviation for MNIST data.

MODEL	MNIST
CATGAN (SPRINGENBERG, 2015)	9.70±NR
VADE (JIANG ET AL., 2016)	5.54±NR
INFOGAN (CHEN ET AL., 2016)	5.00±NR
AAE (MAKHZANI ET AL., 2016)	4.10±1.13
AMM	2.86±0.46

4.3 SVHN

4.3.1 Unsupervised Clustering

In this section, unsupervised clustering is revisited. The SVHN dataset is used to investigate how the introduction of confounding attributes, such as color and contrast, affects the semantic separation of digits. To model $p(\boldsymbol{y})p(\boldsymbol{z}\,|\,\boldsymbol{y})$ a 32 dimensional mixture of 18 spherical, unit variance, Gaussians is used. A multinomial prior is used for $p(\boldsymbol{y})$ with uniform probability for each class The means of each distribution are regularly spaced at intervals of 6 units from -6 to 6 along the first two dimensions and from -3 to 3 along the third dimension. The trailing 29 dimensions are set to 0 for each mean.

Figure 3a shows random samples drawn from each component distribution generated by $G_{\boldsymbol{x}}$. We can see four distinct groupings based on the global features of SVHN examples. The top row and last three columns of the bottom row show images with dark backgrounds with light numbers. The middle row and first three columns of the last row show images with light backgrounds and dark numbers. Looking closer at the top two rows we see a nearly symmetric clustering based on number. For example, in the first column we see clusters corresponding to *zero*, in the second column we see clusters corresponding to *one*, and in all of the main groupings we see clusters with numbers *two* and *seven* together. The clusters that combine *twos* and *sevens* are reflected by the orange and green groupings in figure 3b, which is a t-SNE projection of testing samples drawn from $G_{\boldsymbol{z}}$ onto a 2D manifold. We show in 3c that AMM learns a smooth latent manifold as we interpolate between examples from SVHN.

(a) Randomly generated images   (b) t-SNE projection   (c) Interpolation

Figure 3: Unsupervised clustering of SVHN data with 18 mixture components. *(a)* Randomly generated images for each mixture component. The color-boxes delineate four groups of clusters *(Rows 1, 2, 3 (Left) and 3 (Right))* with shared global characteristics. *(b)* t-SNE projection of testing samples, color-coded for the SVHN class label (0 to 9). *(c)* Interpolation between examples: original data samples *(Columns i) and v))*, associated reconstructions *(Columns ii) and iv))*, linearly interpolated reconstructions *(Columns iii))*.

4.3.2 Semi-supervised clustering and classification

It is evident from the last experiment that the confounders introduced by the SVHN dataset made unsupervised semantic clustering more difficult. In this section we show how SAMM can be used to guide clustering along predefined categories using only a small amount of labeled data. To this end we limit the samples drawn from $q(\boldsymbol{x},\boldsymbol{y})$ to a random selection of 1000 examples from the training set. To model $p(\boldsymbol{y})p(\boldsymbol{z}\,|\,\boldsymbol{y})$ we use a 64 dimensional mixture of 10 spherical Gaussians, each with unit variance. In placing the means of each distribution, we take advantage of our prior knowledge of the task. For example, from figure 2e, we can see that *nines* are closer to *fours* than they are to *zeros*, and reflect these assumptions in designing $p(\boldsymbol{y})p(\boldsymbol{z}\,|\,\boldsymbol{y})$. There is considerable class imbalance in the SVHN dataset

7

so a multinomial prior is used for $p(\boldsymbol{y})$ with each class probability set to the frequency observed in the training data. The placement of each mean $\boldsymbol{\mu}_k$ within the continuous latent manifold $\boldsymbol{z}$ is shown in table 3 of the appendix. We also run this experiment allowing the $\boldsymbol{\mu}_k$'s to be learned using equation (12).

Table 2 reports the test-set error-rate mean and variance over 10 trials. SAMM achieves 7.02 ± 0.17 percent error rate with the fixed means, and 6.43 ± 0.12 when the means are learned, which is an improvement over the ALI baseline. Figure 4 shows visualizations of results from 1 of the 10 trials. Finally, given that we have defined $p(\boldsymbol{y})p(\boldsymbol{z} \mid \boldsymbol{y})$ we can use Bayes' theorem to derive $p(\boldsymbol{y} \mid \boldsymbol{z})$ and get a classifier given an image embedding $\tilde{\boldsymbol{z}}$:

$$\tilde{\boldsymbol{y}}_{\tilde{\boldsymbol{z}}} = \underset{k}{\operatorname{argmax}}[p(\boldsymbol{z} = \tilde{\boldsymbol{z}} \mid \boldsymbol{y} = k)p(\boldsymbol{y} = k)] \tag{18}$$

Figures 4e and 4f compare the confusion matrices for predictions given by $\tilde{\boldsymbol{y}}_{\tilde{\boldsymbol{z}}}$ and those given by $\tilde{\boldsymbol{y}}$ from $G_{\boldsymbol{y}}$. The similarity between each is further evidence that the inference network has learned to embed data according to the desired distribution.

(a) Test images   (b) Random   (c) Interpolation

(d) t-SNE projection   (e) $\tilde{\boldsymbol{y}}_{\tilde{\boldsymbol{z}}}$ (f) $\tilde{\boldsymbol{y}}$

Figure 4: Semi-supervised clustering and classification of SVHN data with 10 mixture components. *(a)* Test image predictions: each row corresponds to the predicted class. *(b)* Randomly generated images for each mixture component. *(c)* Interpolation between examples: original data samples in first and last columns. *(d)* t-SNE projection of testing samples, color-coded for the SVHN class label (0 to 9). Confusion matrix for predictions given an image embedding *(e)* and given the generator $G_{\boldsymbol{y}}$ *(e)*.

Table 2: Semi-supervised test set missclassification rate and standard deviation for SVHN data.

MODEL	SVHN ($N=1000$)
AAE (MAKHZANI ET AL., 2016)	17.70 ± 0.24
IMPROVEDGAN (SALIMANS ET AL., 2016)	8.11 ± 1.30
ALI (DUMOULIN ET AL., 2016)	7.42 ± 0.65
TRIPLEGAN (LI ET AL., 2017)	5.77 ± 0.17
SGAN (DENG ET AL., 2017)	5.73 ± 0.12
SAMM	7.02 ± 0.17
SAMM LEARNED $\boldsymbol{\mu}_k$	6.43 ± 0.12

8

5 Conclusion

The AMM is presented as a generative model for unsupervised or semi-supervised data clustering. It is the first adversarially optimized method to model the conditional dependence between categorical and continuous latent variables. The AMM achieves state of the art unsupervised clustering results and competitive semi-supervised classification results on benchmark datasets.

References

Xi Chen, Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems 29*, pp. 2172–2180. 2016.

Zhijie Deng, Hao Zhang, Xiaodan Liang, Luona Yang, Shizhen Xu, Jun Zhu, and Eric P. Xing. Structured generative adversarial networks. In *Advances in Neural Information Processing Systems 30*, pp. 3902–3912. 2017.

Nat Dilokthanakul, Pedro A. M. Mediano, Marta Garnelo, Matthew C. H. Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *International Conference on Learning Representations*, 2017.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. In *International Conference on Learning Representations*, 2016.

Zhe Gan, Liqun Chen, Weiyao Wang, Yuchen Pu, Yizhe Zhang, and Lawrence Carin. Triangle generative adversarial networks. In *Advances in Neural Information Processing Systems 30*, pp. 5251–5260, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. 2014.

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

Zhuxi Jiang, Yin Zheng, et al. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems 30*, pp. 5580–5590. 2017.

Diederik Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2013.

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

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27*, pp. 3581–3589. 2014.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Chongxuan Li, Kun Xu, Jun Zhu, and Bo Zhang. Triple generative adversarial nets. *arXiv preprint arXiv:1703.02291*, 2017.

Lars Maaløe, Marco Fraccaro, and Ole Winther. Semi-supervised generation with cluster-aware generative models. *arXiv preprint arXiv:1704.00637*, 2017.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial autoencoders. In *International Conference on Learning Representations*, 2016.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems 30*, pp. 2015–2025. 2017.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29*, pp. 2234–2242. 2016.

Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

A SAMM Algorithm

Algorithm 2 outlines the SAMM training procedure.

Algorithm 2 SAMM training procedure using distributions (4), (6), and (11).

$\theta_{G_{\boldsymbol{y}}(\boldsymbol{x})}, \theta_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))}, \theta_{G_{\boldsymbol{z}}(\boldsymbol{y})}, \theta_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))}, \theta_D$ $\quad\quad\quad\quad$ ▷ Initialize SAMM parameters

while not done **do**

$\quad \boldsymbol{x}_u^{(1)},...,\boldsymbol{x}_u^{(M)} \sim q(\boldsymbol{x})$ $\quad\quad\quad\quad$ ▷ Sample from unlabeled data and priors

$\quad \boldsymbol{y}_u^{(1)},...,\boldsymbol{y}_u^{(M)} \sim p(\boldsymbol{y})$

$\quad \boldsymbol{z}_u^{(j)} \sim p(\boldsymbol{z}\,|\,\boldsymbol{y}=\boldsymbol{y}_u^{(j)}), \quad j=1,...,M$

$\quad \tilde{\boldsymbol{x}}_u^{(j)} \sim p(\boldsymbol{x}\,|\,\boldsymbol{y}=\boldsymbol{y}_u^{(j)},\boldsymbol{z}=\boldsymbol{z}_u^{(j)}), \quad j=1,...,M$ $\quad\quad\quad\quad$ ▷ Sample from conditionals

$\quad \tilde{\boldsymbol{y}}_u^{(i)} \sim q(\boldsymbol{y}\,|\,\boldsymbol{x}=\boldsymbol{x}_u^{(i)}), \quad i=1,...,M$

$\quad \tilde{\boldsymbol{z}}_u^{(i)} \sim q(\boldsymbol{z}\,|\,\boldsymbol{x}=\boldsymbol{x}_u^{(i)},\boldsymbol{y}=\tilde{\boldsymbol{y}}_u^{(i)}), \quad i=1,...,M$

$\quad \left(\boldsymbol{x}_\ell^{(1)},...,\boldsymbol{x}_\ell^{(M)}\right),\left(\tilde{\boldsymbol{y}}_\ell^{(1)},...,\tilde{\boldsymbol{y}}_\ell^{(M)}\right) \sim q(\boldsymbol{x},\boldsymbol{y})$ $\quad\quad\quad\quad$ ▷ Sample from labeled data and priors

$\quad \boldsymbol{y}_\ell^{(1)},...,\boldsymbol{y}_\ell^{(M)} \sim p(\boldsymbol{y})$

$\quad \boldsymbol{z}_\ell^{(j)} \sim p(\boldsymbol{z}\,|\,\boldsymbol{y}=\boldsymbol{y}_\ell^{(j)}), \quad j=1,...,M$

$\quad \tilde{\boldsymbol{x}}_\ell^{(j)} \sim p(\boldsymbol{x}\,|\,\boldsymbol{y}=\boldsymbol{y}_\ell^{(j)},\boldsymbol{z}=\boldsymbol{z}_\ell^{(j)}), \quad j=1,...,M$ $\quad\quad\quad\quad$ ▷ Sample from conditionals

$\quad \tilde{\boldsymbol{z}}_\ell^{(i)} \sim q(\boldsymbol{z}\,|\,\boldsymbol{x}=\boldsymbol{x}_\ell^{(i)},\boldsymbol{y}=\tilde{\boldsymbol{y}}_\ell^{(i)}), \quad i=1,...,M$

$\quad \boldsymbol{\rho}_{q_u}^{(i)} \leftarrow D(\boldsymbol{x}_u^{(i)},\tilde{\boldsymbol{y}}_u^{(i)},\tilde{\boldsymbol{z}}_u^{(i)}), \quad i=1,...,M$ $\quad\quad\quad\quad$ ▷ Compute predictions for unlabeled data

$\quad \boldsymbol{\rho}_{p_u}^{(j)} \leftarrow D(\tilde{\boldsymbol{x}}_u^{(j)},\boldsymbol{y}_u^{(j)},\boldsymbol{z}_u^{(j)}), \quad j=1,...,M$

$\quad \boldsymbol{\rho}_{q_\ell}^{(i)} \leftarrow D(\boldsymbol{x}_\ell^{(i)},\tilde{\boldsymbol{y}}_\ell^{(i)},\tilde{\boldsymbol{z}}_\ell^{(i)}), \quad i=1,...,M$ $\quad\quad\quad\quad$ ▷ Compute predictions for labeled data

$\quad \boldsymbol{\rho}_{p_\ell}^{(j)} \leftarrow D(\tilde{\boldsymbol{x}}_\ell^{(j)},\boldsymbol{y}_\ell^{(j)},\boldsymbol{z}_\ell^{(j)}), \quad j=1,...,M$

$\quad \mathcal{L}_{D_u} \leftarrow -\frac{1}{2M}\sum_{i=1}^{M}\log(\boldsymbol{\rho}_{q_u}^{(i)}) - \frac{1}{2M}\sum_{j=1}^{M} log(1-\boldsymbol{\rho}_{p_u}^{(j)})$ $\quad\quad\quad\quad$ ▷ Compute discriminator losses

$\quad \mathcal{L}_{D_\ell} \leftarrow -\frac{1}{2M}\sum_{i=1}^{M}\log(\boldsymbol{\rho}_{q_\ell}^{(i)}) - \frac{1}{2M}\sum_{j=1}^{M} log(1-\boldsymbol{\rho}_{p_\ell}^{(j)})$

$\quad \mathcal{L}_{G_{\boldsymbol{y}_u}(\boldsymbol{x})} = \mathcal{L}_{G_{\boldsymbol{z}_u}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))} \leftarrow -\frac{1}{2M}\sum_{i=1}^{M}\log(1-\boldsymbol{\rho}_{q_u}^{(i)})$ $\quad\quad\quad\quad$ ▷ Compute inference losses

$\quad \mathcal{L}_{G_{\boldsymbol{z}_\ell}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))} \leftarrow -\frac{1}{2M}\sum_{i=1}^{M}\log(1-\boldsymbol{\rho}_{q_\ell}^{(i)})$

$\quad \mathcal{L}_{G_{\boldsymbol{x}_u}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))} = \mathcal{L}_{G_{\boldsymbol{z}_u}(\boldsymbol{y})} \leftarrow -\frac{1}{2M}\sum_{i=1}^{M}\log(\boldsymbol{\rho}_{p_u}^{(i)})$ $\quad\quad\quad\quad$ ▷ Compute $\boldsymbol{x}$ generator losses

$\quad \mathcal{L}_{G_{\boldsymbol{x}_\ell}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))} = \mathcal{L}_{G_{\boldsymbol{z}_\ell}(\boldsymbol{y})} \leftarrow -\frac{1}{2M}\sum_{i=1}^{M}\log(\boldsymbol{\rho}_{p_\ell}^{(i)})$

$\quad \theta_D \leftarrow \theta_D - \nabla_{\theta_D}(\mathcal{L}_{D_u}+\mathcal{L}_{D_\ell})$ $\quad\quad\quad\quad$ ▷ Update discriminator parameters

$\quad \theta_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))} \leftarrow \theta_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))} - \nabla_{\theta_{G_{\boldsymbol{z}}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))}}\left(\mathcal{L}_{G_{\boldsymbol{z}_u}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))}+\mathcal{L}_{G_{\boldsymbol{z}_\ell}(\boldsymbol{x},G_{\boldsymbol{y}}(\boldsymbol{x}))}\right)$ $\quad$ ▷
Update $\boldsymbol{z}$ inference parameters

$\quad \theta_{G_{\boldsymbol{y}}(\boldsymbol{x})} \leftarrow \theta_{G_{\boldsymbol{y}}(\boldsymbol{x})} - \nabla_{\theta_{G_{\boldsymbol{y}}(\boldsymbol{x})}}\mathcal{L}_{G_{\boldsymbol{y}_u}(\boldsymbol{x})}$ $\quad\quad\quad\quad$ ▷ Update $\boldsymbol{y}$ inference parameters

$\quad \theta_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))} \leftarrow \theta_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))} - \nabla_{\theta_{G_{\boldsymbol{x}}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))}}\left(\mathcal{L}_{G_{\boldsymbol{x}_u}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))}+\mathcal{L}_{G_{\boldsymbol{x}_\ell}(\boldsymbol{y},G_{\boldsymbol{z}}(\boldsymbol{y}))}\right)$ $\quad$ ▷
Update $\boldsymbol{x}$ synthesis parameters

$\quad \theta_{G_{\boldsymbol{z}}(\boldsymbol{y})} \leftarrow \theta_{G_{\boldsymbol{z}}(\boldsymbol{y})} - \nabla_{\theta_{G_{\boldsymbol{z}}(\boldsymbol{y})}}\left(\mathcal{L}_{G_{\boldsymbol{z}_u}(\boldsymbol{y})}+\mathcal{L}_{G_{\boldsymbol{z}_\ell}(\boldsymbol{y})}\right)$ $\quad\quad\quad\quad$ ▷ Update $\boldsymbol{z}$ synthesis parameters

---

B Experiment Information

B.1 Model Architectures

Figures 5 and 6 detail the model architectures for the SVHN and MNIST experiments, respectively.

B.2 Mean Placement

The placement of each mean for the fixed mean semi-supervised SVHN experiment is shown in table 3

Table 3: SVHN Semi-Supervised: Placement of means for $p(\boldsymbol{y})p(\boldsymbol{z}\,|\,\boldsymbol{y})$

MEAN	z_0	z_1	z_2	z_3	z_{4-31}
μ_0	-3	3	-3	-3	0
μ_1	-3	-3	3	3	0
μ_2	-3	3	3	-3	0
μ_3	3	-3	-3	-3	0
μ_4	-3	-3	3	-3	0
μ_5	3	-3	3	-3	0
μ_6	3	3	3	-3	0
μ_7	-3	3	3	3	0
μ_8	3	3	-3	-3	0
μ_9	-3	-3	-3	-3	0

Layer Name	Inputs	Channels	Width	Stride	Dropout	Batch Norm	Activation
x0	-	1	-	-	-	-	-
y1	x0	16	2	1	0.0	-	-
z1	x0	16	2	1	0.0	-	-
y1a	z1 + y1	16	-	-	-	•	Leak 0.2
z1a	z1	16	-	-	-	•	Leak 0.2
y2	y1a	128	3	2	0.0	-	-
z2	z1a	128	3	2	0.0	-	-
y2a	z2 + y2	128	-	-	-	•	Leak 0.2
z2a	z2	128	-	-	-	•	Leak 0.2
y3	y2a	128	3	1	0.0	-	-
z3	z2a	128	3	1	0.0	-	-
y3a	z3 + y3	128	-	-	-	•	Leak 0.2
z3a	z3	128	-	-	-	•	Leak 0.2
y4	y3a	256	3	2	0.0	-	-
z4	z3a	256	3	2	0.0	-	-
y4a	z4 + y4	256	-	-	-	•	Leak 0.2
z4a	z4	256	-	-	-	•	Leak 0.2
y5	y4a	256	3	1	0.0	-	-
z5	z4a	256	3	1	0.0	-	-
y5a	z5 + y5	256	-	-	-	•	Leak 0.2
z5a	z5	256	-	-	-	•	Leak 0.2
y6	y5a	512	4	1	0.0	-	-
z6	z5a	512	4	1	0.0	-	-
y6a	z6 + y6	512	-	-	-	•	Leak 0.2
z6a	z6	512	-	-	-	•	Leak 0.2
y_mu	y6a	10	1	1	-	-	-
y_log_var	y6a	10	1	1	-	-	-
z_mu	z6a	64	1	1	-	-	-
z_log_var	z6a	64	1	1	-	-	-

(a) SVHN: $G_z(x)G_y(x,G_z(x))$

Layer Name	Inputs	Channels	Width	Stride	Dropout	Batch Norm	Activation
y0	-	10	-	-	-	-	-
z_mu	y0	64	1	1	-	-	-

(b) SVHN: $G_z(y)$

Layer Name	Inputs	Channels	Width	Stride	Dropout	Batch Norm	Activation
y	-	10	-	-	-	-	-
z	-	64	-	-	-	-	-
y6	y	512	1	1	0.0	-	-
z6	z	512	1	1	0.0	-	-
z6a	z6 + y6	512	-	-	-	•	Leak 0.2
y5	y6a	256	4	1	0.0	-	-
z5	z6a	256	4	1	0.0	-	-
z5a	z5 + y5	256	-	-	-	•	Leak 0.2
y4	y5	256	3	1	0.0	-	-
z4	z5a	256	3	1	0.0	-	-
z4a	z4 + y4	256	-	-	-	•	Leak 0.2
y3	y4	128	3	2	0.0	-	-
z3	z4a	128	3	2	0.0	-	-
z3a	z3 + y3	128	-	-	-	•	Leak 0.2
y2	y3	128	3	1	0.0	-	-
z2	z3a	128	3	1	0.0	-	-
z2a	z2 + y2	128	-	-	-	•	Leak 0.2
y1	y2	16	3	2	0.0	-	-
z1	z2a	16	3	2	0.0	-	-
z1a	z1 + y1	16	-	-	-	•	Leak 0.2
y0	y1	1	2	1	0.0	-	-
z0	z1a	1	2	1	0.0	-	-
x0	z0 + y0	1	-	-	-	•	Sigmoid

(c) SVHN: $G_x(y,G_z(y))$

Layer Name	Inputs	Channels	Width	Stride	Dropout	Batch Norm	Activation
x0	-	1	-	-	-	-	-
y0	-	10	-	-	-	-	-
z0	-	64	-	-	-	-	-
x1	x0	16	2	1	0.1	-	Leak 0.2
x2	x1	128	3	2	0.1	-	Leak 0.2
x3	x2	128	3	1	0.1	-	Leak 0.2
x4	x3	256	3	2	0.1	-	Leak 0.2
x5	x4	256	3	1	0.1	-	Leak 0.2
x6	x5	512	4	1	0.1	-	Leak 0.2
y1	y0	512	1	1	0.0	-	Leak 0.2
y2	y1	412	1	1	0.1	-	Leak 0.2
z1	z0	512	1	1	0.0	-	Leak 0.2
z2	z1	512	1	1	0.1	-	Leak 0.2
xyz	[x6, y2, z2]	1536	1	1	0.1	-	Leak 0.2
xyz	xyz	1536	1	1	0.1	-	Leak 0.2
rho	xyz	1	1	1	0.1	-	-

(d) SVHN: $D(x,y,z)$

Figure 5: Model architecture for SVHN

Layer Name	Inputs	Channels	Width	Stride	Dropout	Batch Norm	Activation
x0	-	1	-	-	-	-	-
y1	x0	16	2	1	0.2	-	-
z1	x0	16	2	1	0.2	-	-
y1a	z1 + y1	16	-	-	-	•	Leak 0.2
z1a	z1	16	-	-	-	•	Leak 0.2
y2	y1a	32	3	2	0.2	-	-
z2	z1a	32	3	2	0.2	-	-
y2a	z2 + y2	32	-	-	-	•	Leak 0.2
z2a	z2	32	-	-	-	•	Leak 0.2
y3	y2a	64	3	2	0.2	-	-
z3	z2a	64	3	2	0.2	-	-
y3a	z3 + y3	64	-	-	-	•	Leak 0.2
z3a	z3	64	-	-	-	•	Leak 0.2
y4	y3a	64	3	1	0.2	-	-
z4	z3a	64	3	1	0.2	-	-
y4a	z4 + y4	64	-	-	-	•	Leak 0.2
z4a	z4	64	-	-	-	•	Leak 0.2
y5	y4a	128	4	1	0.2	-	-
z5	z4a	128	4	1	0.2	-	-
y5a	z5 + y5	128	-	-	-	•	Leak 0.2
z5a	z5	128	-	-	-	•	Leak 0.2
y_mu	y5a	10	1	1	0.0	-	-
y_log_var	y5a	10	1	1	0.0	-	-
z_mu	z5a	64	1	1	0.0	-	-
z_log_var	z5a	64	1	1	0.0	-	-

(a) MNIST: $G_z(x)G_y(x, G_z(x))$

Layer Name	Inputs	Channels	Width	Stride	Dropout	Batch Norm	Activation
y0	-	10	-	-	-	-	-
z1	y0	64	1	1	0.0	•	Leak 0.2
z2	z1	64	1	1	0.2	•	Leak 0.2
z_mu	z2	64	1	1	0.0	-	-

(b) MNIST: $G_z(y)$

Layer Name	Inputs	Channels	Width	Stride	Dropout	Batch Norm	Activation
y	-	10	-	-	-	-	-
z	-	64	-	-	-	-	-
y5	y	128	1	1	0.0	-	-
z5	z	128	1	1	0.0	-	-
z5a	z5 + y5	128	-	-	-	•	Leak 0.2
y4	y5	64	4	1	0.2	-	-
z4	z5a	64	4	1	0.2	-	-
z4a	z4 + y4	64	-	-	-	•	Leak 0.2
y3	y4	64	3	1	0.2	-	-
z3	z4a	64	3	1	0.2	-	-
z3a	z3 + y3	64	-	-	-	•	Leak 0.2
y2	y3	32	3	2	0.2	-	-
z2	z3a	32	3	2	0.2	-	-
z2a	z2 + y2	32	-	-	-	•	Leak 0.2
y1	y2	16	3	2	0.2	-	-
z1	z2a	16	3	2	0.2	-	-
z1a	z1 + y1	16	-	-	-	•	Leak 0.2
y0	y1	1	2	1	0.2	-	-
z0	z1a	1	2	1	0.2	-	-
x0	z0 + y0	1	-	-	-	•	Sigmoid

(c) MNIST: $G_x(y, G_z(y))$

Layer Name	Inputs	Channels	Width	Stride	Dropout	Batch Norm	Activation
x0	-	1	-	-	-	-	-
y0	-	10	-	-	-	-	-
z0	-	64	-	-	-	-	-
x1	x0	16	2	1	0.2	-	Leak 0.2
x2	x1	32	3	2	0.2	-	Leak 0.2
x3	x2	64	3	2	0.2	-	Leak 0.2
x4	x3	64	3	1	0.2	-	Leak 0.2
x5	x4	128	4	1	0.2	-	Leak 0.2
y1	y0	64	1	1	0.0	-	Leak 0.2
y2	y1	64	1	1	0.2	-	Leak 0.2
z1	z0	64	1	1	0.0	-	Leak 0.2
z2	z1	64	1	1	0.2	-	Leak 0.2
xyz	[x5, y2, z2]	256	1	1	0.2	-	Leak 0.2
xyz	xyz	256	1	1	0.2	-	Leak 0.2
rho	xyz	1	1	1	0.2	-	-

(d) MNIST: $D(x, y, z)$

Figure 6: Model architecture for MNIST