Training self-supervised class-conditional GAN with virtual labels

Jeongik Cho
jeongik.jo.01@gmail.com

Abstract

Class-conditional GAN is a conditional GAN that can generate class-conditional distribution. Among class-conditional GANs, InfoGAN with categorical latent distribution can generate class-conditional data through a self-supervised (unsupervised) method without a labeled dataset. Instead, InfoGAN requires optimal categorical latent distribution to train the model.

In this paper, we propose a novel GAN that allows the model to perform self-supervised class-conditional data generation and clustering without knowing the optimal categorical latent distribution. The proposed method uses three losses. The first loss is the cross-entropy classification loss to predict the label of the fake data. The classifier is trained with the classification loss. The second loss is the CAGAN loss for class-conditional data generation. The virtual label of the real data predicted by the classifier is used for CAGAN loss. The generator and discriminator are trained with CAGAN loss. The third loss is the classifier gradient penalty loss. The classifier gradient penalty loss regularizes the slope of the classifier's decision boundary so that the decision boundary converges to a local optimum over a wide region.

Additionally, the proposed method updates the categorical latent distribution with the output distribution of the classifier on the real data. As training progresses, the entropy of the categorical latent distribution gradually decreases by the classifier gradient penalty loss and converges to the appropriate value. The converged categorical latent distribution becomes appropriate to represent the discrete part of the data distribution.

The proposed method does not require labeled data, optimal categorical latent distribution, and a good metric to calculate the distance between data.

1 Introduction

Class-conditional GAN is a conditional GAN [2] that can generate class-conditional distribution with a labeled dataset. In general, the generator of class-conditional GAN takes a continuous latent distribution and a discrete categorical distribution as inputs and generates class-conditional distribution. ACGAN [3] and CAGAN [4] are examples of class-conditional GANs. However, these class-conditional GANs can only be trained given labels, which is the conditional categorical distribution of the dataset. Therefore, these methods cannot be utilized with unlabeled datasets.

Unlike ACGAN or CAGAN, InfoGAN [5] with categorical latent distribution can generate class-conditional data distribution even if the data is not labeled. However, InfoGAN requires optimal categorical latent distribution. It includes the number of categories and the probability for each category. For example, to perform class-conditional data generation with InfoGAN on the MNIST handwritten digits dataset [9] without labels, InfoGAN needs to know the number of categories (10 categories) and the probability of each category (0.1 for each category) for categorical latent distribution.

In this paper, we introduce a Virtual Conditional Activation GAN (VCAGAN) that is capable of generating class-conditional data without being given labels and optimal categorical latent distribution.

A VCAGAN consists of a discriminator, classifier, and class-conditional generator. The discriminator has d_c-dimensional output like CAGAN (d_c represents the dimensionality of the categorical latent distribution). The class-conditional generator takes d_z-dimensional continuous latent distribution and d_c-dimensional categorical latent distribution as inputs to generate class-conditional data.

VCAGAN uses three different losses. The first loss is the cross-entropy classification loss to predict the label of the fake data. The classifier is trained to minimize the classification loss. The second loss is the CAGAN loss for class-conditional data generation. The label of the real data for CAGAN

loss is predicted from the classifier. The generator and discriminator are trained with CAGAN loss. The third loss is the classifier gradient penalty loss. The classifier gradient penalty loss regularizes the slope of the classifier's decision boundary so that the decision boundary converges to a better local optimum.

In addition, VCAGAN updates the categorical latent distribution with the classifier output distribution of real data. This makes the categorical latent distribution of the fake data similar to that of the real data.

There are several differences between InfoGAN and VCAGAN. The first difference is that the VCAGAN uses CAGAN loss as the adversarial loss and does not use classification loss for the generator. The generator of InfoGAN is trained to minimize both adversarial loss and classification loss like ACGAN. This lowers the generator performance of InfoGAN because adversarial loss and classification loss in the generator conflict with each other. On the other hand, since VCAGAN's generator is only trained with CAGAN loss (only adversarial losses), there is no conflict between adversarial loss and classification loss in the generator.

Second, VCAGAN gradually changes the categorical latent distribution during training. VCAGAN updates the categorical latent distribution to follow the classifier output distribution for real data. This allows VCAGAN to approximate the optimal categorical latent distribution without knowing it, unlike InfoGAN.

Third, VCAGAN can adjust the sensitivity of the categories through classifier gradient penalty loss. Without the classifier gradient penalty loss, the gradient of the VCAGAN classifier's decision boundary can become very large, and the decision boundary will fall into a local optimum in a narrow region. The larger the classifier gradient penalty loss weight of VCAGAN, the decision boundary of the classifier falls into the local optimum of a wider region, and the entropy of the categorical latent distribution decreases. In other words, VCAGAN can adjust the sensitivity of the categories through the weight (multiplier) of the classifier gradient penalty loss.

2 Class-conditional data generation

Typically, when training a GAN, everything is assumed to be continuous. This means that the data distribution and latent distribution are assumed to be continuous, and the generator and discriminator of GAN are assumed to be continuous functions (deep learning models must be differentiable continuous functions to be trained).

However, the data distribution is not necessarily continuous. When data distribution includes a discrete part and latent distribution is continuous, a sufficiently complex deep generative model can approximate the discrete part of the data distribution. However, approximating the discrete part of the data distribution is still not easy for most deep generative models, which is a continuous function.

The left part of Fig. 1 shows a data distribution example consisting of four Gaussian clusters. There is no perfect discrete part in this data distribution (i.e., the probability density function is still continuous), but one can see that it is easier to represent this data distribution with a discrete (categorical) latent distribution.

The right part of Fig. 1 shows generated data with GAN trained only with a continuous latent distribution. One can see that the model generates lines connecting each cluster. This is because the latent distribution is continuous, and the generator is a continuous function, making it difficult to represent the discrete part of the data distribution. As training progresses, the probability density of the line connecting the clusters decreases, but it requires a long training period, and it is hard to say that the continuous latent distribution correctly represents the real data distribution.

For datasets with discrete parts, using a discrete latent distribution is more appropriate for model training and data representation. Class-conditional generative models, such as ACGAN [3] or CAGAN [4], take both continuous latent distribution and discrete categorical latent distribution as inputs and generate class-conditional data distribution. It makes them appropriate for representing datasets with discrete parts. However, ACGAN or CAGAN cannot be trained if there is no label for data.

InfoGAN [5] can perform class-conditional data generation and inversion (clustering) by maximizing mutual information of generator input categorical latent distribution and classifier output distribution, even if the data is not labeled. Following equations show losses for InfoGAN with the categorical latent distribution.

$$L_{cls} = \mathbb{E}_{z,c}\left[cross\ entropy(c, Q(G(z,c)))\right] \tag{1}$$

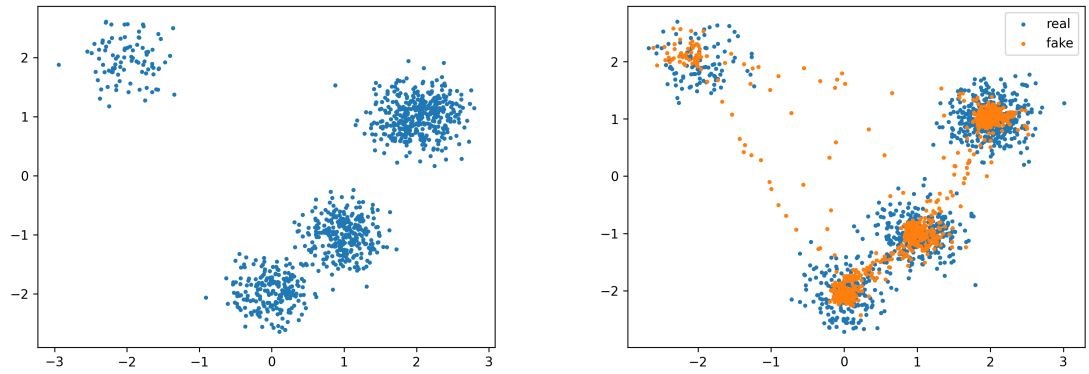

Figure 1: Left part: Two-dimensional dataset consisting of four Gaussian clusters. The centers and probabilities of each cluster are $(-2.0, 2.0), (0.0, -2.0), (1.0, -1.0), (2.0, 1.0)$ and $[0.1, 0.2, 0.3, 0.4]$, respectively. The standard deviation for all clusters is 0.3. Right part: Samples generated by GAN trained only with a continuous latent distribution.

$$L^d_{adv} = \mathbb{E}_{x,z,c}\left[f_d(D(x), D(G(z,c)))\right] \tag{2}$$

$$L^g_{adv} = \mathbb{E}_{z,c}\left[f_g(D(G(z,c)))\right] \tag{3}$$

$$L_d = L^d_{adv} \tag{4}$$

$$L_q = \lambda_{cls} L_{cls} \tag{5}$$

$$L_g = L^g_{adv} + \lambda_{cls} L_{cls} \tag{6}$$

In Eqs. 4, 5, and 6, L_d, L_q, and L_g represent discriminator loss, classifier loss, and generator loss of InfoGAN, respectively. L_{cls} and λ_{cls} represent classification loss and classification loss weight, respectively. In Eqs. 2 and 3, L^d_{adv} and L^g_{adv} represent adversarial losses [6] for GAN training. In Eq. 1, L_{cls} is cross entropy between categorical latent code c and label prediction of generated data $Q(G(z,c))$. D, Q, and G represent the discriminator, classifier, and generator, respectively. z represents continuous latent code sampled from the continuous latent distribution Z. In InfoGAN, a classifier Q can share hidden layers with a discriminator D for efficiency.

From the above equations, one can see that a classifier and a generator are trained to minimize classification loss. InfoGAN has shown that, given an appropriate categorical latent distribution C, it can perform class-conditional data generation and clustering (inversion) even when the data is unlabeled.

However, InfoGAN still needs insight into the categorical latent distribution C. Without knowing the appropriate categorical latent distribution C, InfoGAN cannot perform class-conditional data generation and clustering.

Additionally, InfoGAN's generator is trained with both adversarial loss and classification loss. It means that adversarial loss and classification loss conflict with each other in the generator. This conflict reduces the generative performance of InfoGAN.

In this paper, we introduce VCAGAN, which performs class-conditional data generation and clustering under more general conditions than InfoGAN. We assume the following general conditions:

1. All data is unknown to which cluster it belongs (i.e., there are no labels for all data).
2. Optimal categorical latent distribution is unknown.
3. Metric to measure the distance between the data is unknown.

Under these conditions, ACGAN and CAGAN cannot be used due to condition 1. InfoGAN cannot be used due to condition 2, and recent methods utilizing the K-means algorithm cannot be used due to condition 3. On the other hand, VCAGAN can still perform class-conditional data generation and clustering (inversion) even under these conditions.

3 Virtual Conditional Activation GAN

VCAGAN uses three different losses. Following equations show losses for training VCAGAN.

$$L^d_{adv} = \mathbb{E}_{x,z,c}\left[f_d(D(x) \cdot argmax\ onehot(Q(x)), D(G(z,c)) \cdot c)\right] \tag{7}$$

$$L^g_{adv} = \mathbb{E}_{z,c}\left[f_g(D(G(z,c)) \cdot c)\right] \tag{8}$$

$$L_{creg} = \mathbb{E}_x\left[\|\nabla \log(Q(x) \cdot argmax\ onehot(Q(x)))\|_2^2\right] \tag{9}$$

$$L_d = L^d_{adv} \tag{10}$$

$$L_q = \lambda_{cls} L_{cls} + \lambda_{creg} L_{creg} \tag{11}$$

$$L_g = L^g_{adv} \tag{12}$$

Eqs. 7 and 8 show CAGAN adversarial losses for VCAGAN. f_d and f_g represent adversarial loss functions for discriminator and generator, respectively. Operation "$\cdot$" represents the inner product. Since the true label c of the fake data $G(z, c)$ is known, the adversarial loss for fake data in VCAGAN is the same as CAGAN loss. However, the label of the real data x is unknown. Thus, in VCAGAN, $argmax\,onehot(Q(x))$ is used as the label of the real data x. The $argmax\,onehot$ function replaces the maximum value of the vector with 1 and all other values with 0 (e.g., $argmax\,onehot([0.2, 0.5, 0.3]) = [0.0, 1.0, 0.0]$).

Eq. 9 shows classifier gradient penalty loss for VCAGAN. As with adversarial loss, $argmaxonehot(Q(x))$ is used as the label of real data x. With CAGAN adversarial loss and classification loss, the classifier is trained so that the decision boundary falls on the local optimum that minimizes $P(X)$. However, the classifier may only increase the slope of the decision boundary to minimize classification loss. In such a case, the decision boundary of the classifier will converge to a local optimum in a very narrow region. To avoid this and ensure that the classifier's decision boundary falls on the local optimum of a larger region that minimizes $P(X)$, VCAGAN uses a classifier gradient penalty loss L_{creg}.

In Eq. 11, λ_{cls} and λ_{creg} represent weighting values for each loss. Classification loss L_{cls} is the same as InfoGAN's classification loss (Eq. 1). In Eq. 12, one can see that there is no classification loss L_{cls} in generator loss L_g. Since VCAGAN's generator is trained with adversarial losses only, there is no conflict between L_{cls} and L^g_{adv} as in InfoGAN.

Additionally, VCAGAN updates the probability of the categorical latent distribution $P(C)$ during the training with $\mathbb{E}_x\left[Q(x)\right]$ (i.e., $P(C) \approx \mathbb{E}_x\left[Q(x)\right]$). Through this, VCAGAN can approximate $P(C)$ without knowing it.

Algo. 1 shows the training steps of VCAGAN.

The training step of VCAGAN requires X (data random variable), Z (continuous latent random variable), C (categorical latent random variable), D (discriminator), Q (classifier), and G (generator).

In lines 1-3, the *sample* function represents the sampling function from a random variable. x (real data point), z (continuous latent code), and c_f (fake categorical latent code) are sampled from X, Z, and C, respectively.

In line 4, G generates fake data x' with z and c_f. In lines 5 and 6, D and Q takes a fake data point x' as input and outputs a_f (fake adversarial vector) and c'_f (fake categorical latent code prediction), respectively.

Similarly, in lines 7 and 8, D and Q take a real data point x as input and outputs a_r (real adversarial vector) and c'_r (real categorical latent code prediction). In line 9, real categorical latent code c_r is calculated from c'_r.

In line 10, L_{cls} represents classification loss. *crossentropy* is a function that calculates cross-entropy loss. In line 11, $gradient(y, x)$ function calculates slope dy/dx.

In lines 12, 13, and 14, L_d, L_q, and L_g represent discriminator loss, classifier loss, and generator loss, respectively. f_d and f_g represent adversarial functions for GAN.

In line 15, $P(C)$ is updated with predicted real categorical latent code c'_r. The *update* function can be a simple moving average, an exponential moving average, or others. Updating C early in training makes it easier for C to converge to a trivial solution since the generator is not yet generating class-conditional data properly. Therefore, we recommend initializing $P(C)$ with $[1/d_c, 1/d_c, ..., 1/d_c]$ and updating $P(C)$ after the model has been trained for some time.

After line 16, D, Q and G are updated with losses L_d, L_q, and L_g, respectively.

Algorithm 1 Training step of VCAGAN

Require: X, Z, C, D, Q, G

$x \leftarrow sample(X)$
$z \leftarrow sample(Z)$
$c_f \leftarrow sample(C)$

$x' \leftarrow G(z, c_f)$
$a_f \leftarrow D(x')$
$c'_f \leftarrow Q(x')$
$a_r \leftarrow D(x)$
$c'_r \leftarrow Q(x)$
$c_r \leftarrow argmax\ onehot(c'_r)$

$L_{cls} \leftarrow cross\ entropy(c_f, c'_f)$
$L_{creg} \leftarrow \|gradient(\log(c_r \cdot c'_r), x)\|_2^2$

$L_d \leftarrow f_d(a_r \cdot c_r, a_f \cdot c_f)$
$L_q \leftarrow \lambda_{cls} L_{cls} + \lambda_{creg} L_{creg}$
$L_g \leftarrow f_g(a_f \cdot c_f)$

$P(C) \leftarrow update(P(C), c'_r)$

$return\ L_d, L_q, L_g, C$

4 Experiments

We trained the models to generate two-dimensional Gaussian clusters distribution and the MNIST dataset [9]. In Gaussian clusters experiments, we compare the performance of Vanilla GAN [1], InfoGAN [5], and our proposed VCAGAN. In MNIST experiments, we compared the clustering of VCAGANs according to classifier gradient penalty loss weight λ_{creg}.

The following hyperparameters were used for experiments.

$$
\begin{aligned}
Z &\sim N(0, I_{d_z}) \\
optimizer &= Adam \begin{pmatrix} learning\ rate = 0.001 \\ \beta_1 = 0.0 \\ \beta_2 = 0.99 \end{pmatrix} \\
batch\ size &= 32 \\
train\ step\ per\ epoch &= 1000
\end{aligned}
$$

Classification loss weight $\lambda_{cls} = 1.0$ was used for InfoGAN and VCAGAN. We used exponential moving average with $decayrate = 0.999$ as $update$ function for VCAGAN. In InfoGAN and VCAGAN, classifier Q and discriminator D do not share hidden layers. Equalized learning rate [8] was used for all weights.

4.1 Gaussian clusters experiments

In this experiment, we used the dataset consisting of four 2-dimensioanl Gaussian clusters as a training dataset. Left part of Fig. 1 shows data distribution for the experiments. One can see that there are four Gaussian clusters with different probabilities in data distribution. The generator, discriminator, and classifier consisting of four fully connected hidden layers with 512 units were used for training. We used $d_z = 32$, $\lambda_{r1} = 1$ and $epoch = 100$ for model training. λ_{r1} represents R1 regularization [7] loss weight. In VCAGAN, $d_c = 8$ was used, and $P(C)$ was updated after epoch 20.

Fig. 2 shows samples generated with vanilla GAN (trained only with adversarial loss). The left part of Fig. 2 shows data generated by a vanilla GAN trained with a one-dimensional categorical latent distribution. Since there is no discrete part in the latent distribution, one can see that the vanilla GAN generates lines between clusters.

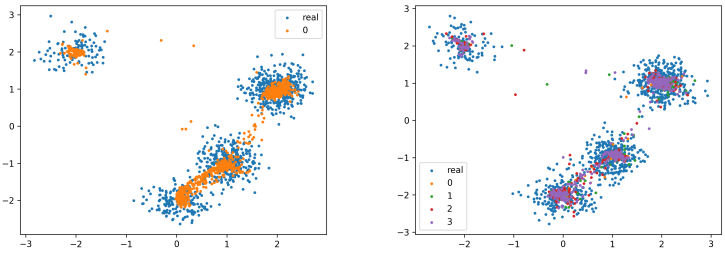

Figure 2: Vanilla GAN (trained only with adversarial loss) is trained with both continuous latent distribution and categorical latent distribution. Left: 1-dimensional categorical latent distribution ($P(C) = [1.0]$). Right: 4-dimensional optimal categorical latent distribution ($P(C) = [0.1, 0.2, 0.3, 0.4]$).

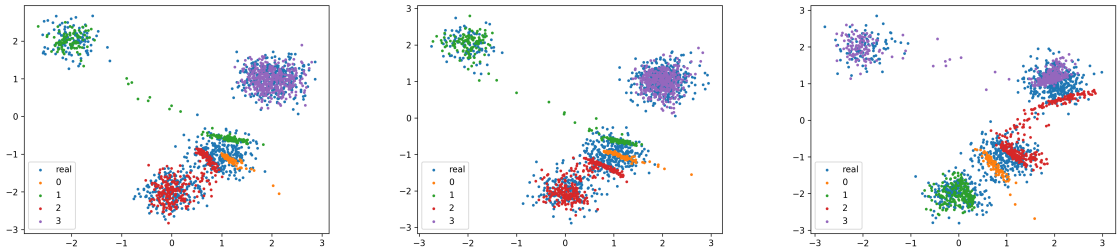

Figure 3: Three InfoGAN trained with 4-dimensional optimal categorical latent distribution ($P(C) = [0.1, 0.2, 0.3, 0.4]$).

The right part of Fig. 2 shows data generated by a vanilla GAN trained with optimal categorical latent distribution ($P(C) = [0.1, 0.2, 0.3, 0.4]$). One can see that the generator of vanilla GAN did not use categorical latent distribution, and training was exclusively performed only with a continuous latent distribution. Therefore, the generator output was also continuous, which caused a line generation between each cluster.

Fig. 3 shows data generated by three InfoGAN trained with the optimal categorical latent distribution. Unlike the Vanilla GAN, one can see that the model generates class-conditional distribution with the categorical latent distribution. However, one can still see some of the problems with InfoGAN in this figure.

The first problem was that even though the categorical latent distribution was optimal, each cluster was not assigned to the category correctly. In the left and middle parts, the top right cluster was assigned to category 3 correctly, but the rest of the clusters were not assigned to the correct cluster. In the right part, the bottom middle cluster was assigned to category 1 correctly, but the rest of the clusters were not. This shows that even when InfoGAN is trained with the optimal categorical latent distribution, it may not be able to assign each cluster to the correct category.

The second problem is that the generator does not generate data near the decision boundary of the classifier. This is because classification loss and adversarial loss are in conflict in the generator of InfoGAN. In particular, one can see a very sharp split between category 0 and 1 in the right part of Fig. 3.

Fig. 4 shows samples generated by VCAGAN trained with different λ_{creg}. In the left part of Fig. 4, one can see that each cluster is divided into more categories than necessary. This is because there is no L_{creg}, and the generated data distribution was not perfectly smooth, so the classifier converged to a local optimum in a very narrow region.

However, in the middle part, one can see that each category is assigned to each cluster correctly. The probability of each category is also very accurate. Also, unlike InfoGAN, one can see that there is a natural division between categories 1 and 3. This is because VCAGAN's generator is only trained with adversarial loss, not classification loss.

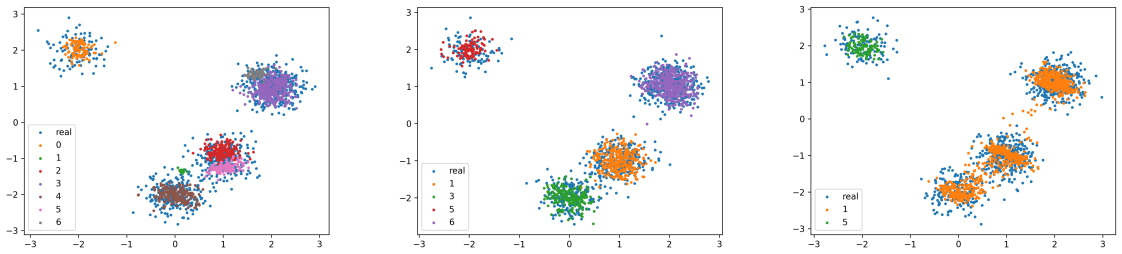

Figure 4: VCAGAN trained with different λ_{creg}. Categories with a probability of less than 1% were ignored. Left: $\lambda_{creg} = 0$, $P(C) = [0.1016, 0.0114, 0.1764, 0.3402, 0.1907, 0.1224, 0.0573]$. Middle: $\lambda_{creg} = 0.01$, $P(C) = [0.3036, 0.2021, 0.1002, 0.3940]$. Right: $\lambda_{creg} = 1$, $P(C) = [0.9002, 0.0998]$.

In the right part, multiple clusters were assigned to the same category. This is because λ_{creg} was too high, causing the classifier to converge to a local optimum (minimize $P(X)$) in a region that is too wide.

From this, one can see that VCAGAN can adjust the sensitivity of each category via λ_{creg}.

4.2 MNIST experiments

In this section, we trained VCAGAN to generate the MNIST handwritten digits dataset [9]. The generator, discriminator, and classifier are simply composed of CNNs. $d_z = 128$, $d_c = 32$, $\lambda_{r1} = 0.1$, $epoch = 500$ were used for the experiments. In VCAGAN, $P(C)$ was updated after epoch 100. We used FID [10], precision & recall [11] for generative performance evaluation. 32k training samples were used for evaluation.

Figs. 5, 6, and 7 show the difference in clustering of VCAGAN according to λ_{creg}.

First, in Fig. 5, because $\lambda_{creg} = 10$ was too low, the classifier decision boundary converged on a local optimum in a narrow region. Thus, similar clusters were split into two or more categories. For example, number 2 was divided into categories 13 and 14. Notice the difference in the lower left part of the number 2 in categories 13 and 14. One can see that probability of category 13 is 2.84%, and category 14 is 6.93%, which together adds up to about 10%. Number 7 was divided into categories 4 and 12. Notice the difference in the representation of the number 7. One can see that probability of category 4 is 1.36%, and category 12 is 8.77%, which together adds up to about 10%. The numbers 4 (categories 6 and 8), 6 (categories 0 and 5), and 9 (categories 2 and 7) were divided into two clusters, each based on their slope.

However, as λ_{creg} increases, the classifier decision boundary converges to the local optimum over a wider region. In Fig. 6, $\lambda_{creg} = 20$ was used for training. One can see that numbers 1 (categories 1 and 10), 2 (categories 0 and 8) and 5 (categories 3 and 11) were divided into two clusters. Also, the numbers 4 and 9 fell into the same category 2. One can see that the other numbers are clustered correctly.

In Fig. 7, $\lambda_{creg} = 30$ was used for training. One can see that the other numbers are clustered correctly, except that the number 1 is split into categories 6 and 8 based on the slope.

Separately, one can see that all three VCAGANs have good unconditional generative performance from FID and precision & recall.

5 Conclusion

In this paper, we introduced VCAGAN, a self-supervised class-conditional GAN. VCAGAN uses CA-GAN loss, classification loss, and classifier gradient penalty loss. Also, the categorical latent distribution is updated to approximate the classifier output distribution of the real data.

The classifier gradient penalty loss weight of VCAGAN controls the sensitivity of each category. The entropy of the categorical latent distribution gradually decreases and converges to the appropriate value.

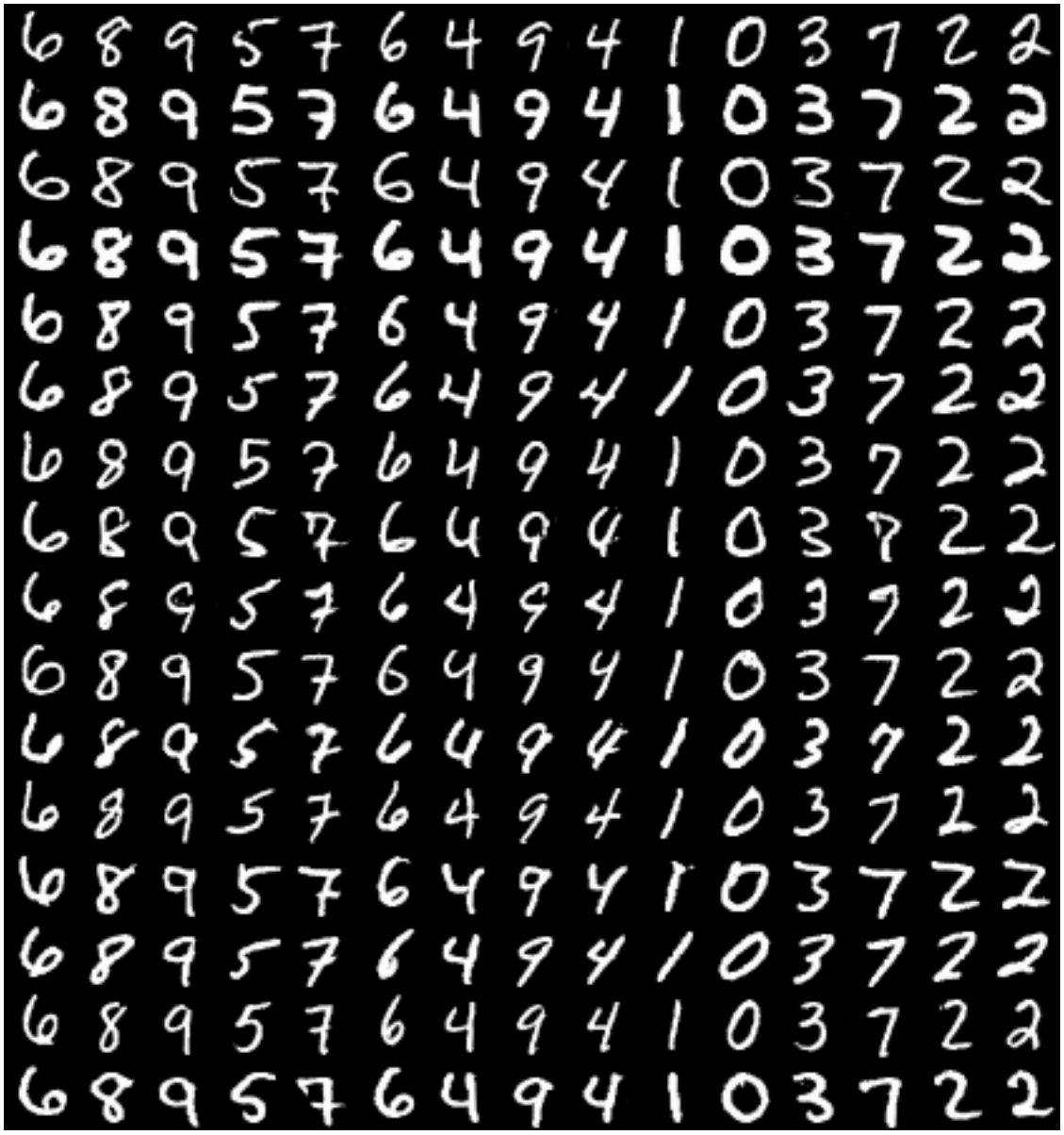

Figure 5: MNIST generated data with $\lambda_{creg} = 10$. Each row has the same continuous latent code, and each column has the same categorical latent code. Out of d_c categories, those with a probability less than 1% were excluded. The probabilities for each category are $[0.0131, 0.0970, 0.0366, 0.0898, 0.0136, 0.0861, 0.0375, 0.0584, 0.0587, 0.1123, 0.0996, 0.1051, 0.0877, 0.0284, 0.0693]$. The entropy of categorical latent distribution is 2.5958. FID: 2.3351, precision: 0.8158, recall: 0.6947.

Figure 6: MNIST generated data with $\lambda_{creg} = 20$. Each row has the same continuous latent code, and each column has the same categorical latent code. Out of d_c categories, those with a probability less than 1% were excluded. The probabilities for each category are $[0.0268, 0.0456, 0.1936, 0.0404, 0.0992, 0.0970, 0.1061, 0.0993, 0.0694, 0.1037, 0.0673, 0.0481]$. The entropy of categorical latent distribution is 2.3776. FID: 2.1096, precision: 0.8104, recall: 0.7033.

Figure 7: MNIST generated data with $\lambda_{creg} = 30$. Each row has the same continuous latent code, and each column has the same categorical latent code. Out of d_c categories, those with a probability less than 1% were excluded. The probabilities for each category are $[0.0943, 0.0967, 0.1026, 0.0974, 0.1072, 0.0453, 0.1000, 0.0645, 0.0998, 0.0902, 0.1019]$. The entropy of categorical latent distribution is 2.3756. FID: 2.044466, precision: 0.8167, recall: 0.6954.

Unlike InfoGAN, VCAGAN's generator is trained with adversarial loss only, so there is no conflict between classification loss and adversarial loss. Therefore, VCAGAN can correctly generate data near the decision boundary of a classifier.

VCAGAN does not require a label of data, optimal categorical latent distribution, and a good metric to calculate the distance between data. This means that VCAGAN can be used in most situations regardless of the data domain.

VCAGAN performed better than Vanilla GAN or InfoGAN with categorical latent distribution in Gaussian clusters generation experiments. We also showed that VCAGAN could also perform self-supervised class-conditional data generation on the MNIST experiment.

References

[1] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Nets. In Commun. ACM, vol. 63, no. 11, pp. 139-144, Nov. 2020. https://doi.org/10.1145/3422622

[2] Mirza, M., Osindero, S.: Conditional Generative Adversarial Nets. In arXiv preprint, 2014, arXiv:1411.1784. https://arxiv.org/abs/1411.1784

[3] A. Odena, C. Olah, J. Shlens, "Conditional Image Synthesis with Auxiliary Classifier GANs," in proceedings of the 34th International Conference on Machine Learning, PMLR 70:2642-2651, 2017. https://proceedings.mlr.press/v70/odena17a.html

[4] Cho, J., Yoon, K.: Conditional Activation GAN: Improved Auxiliary Classifier GAN. In IEEE Access, vol. 8, pp. 216729-216740, 2020. https://doi.org/10.1109/ACCESS.2020.3041480

[5] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P.: InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In NIPS proceedings, 2016. https://papers.nips.cc/paper/2016/hash/7c9d0b1f96aebd7b5eca8c3edaa19ebb-Abstract.html

[6] Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O.: Are GANs Created Equal? A Large-Scale Study. In NIPS, 2018. https://papers.nips.cc/paper/2018/hash/e46de7e1bcaaced9a54f1e9d0d2f800d-Abstract.html

[7] Mescheder, L., Geiger, A., Nowozin S.: Which Training Methods for GANs do actually Converge? In PMLR, 2018. https://proceedings.mlr.press/v80/mescheder18a.html

[8] Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive Growing of GANs for Improved Quality, Stability, and Variation. In ICLR conference, Vancouver, Canada, Apr. 30-May 3, 2018. https://openreview.net/forum?id=Hk99zCeAb

[9] Lecun, Y., Cortes, C., and Burges, C.: MNIST handwritten digit database, 2010. In ATT Labs. http://yann.lecun.com/exdb/mnist

[10] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In NIPS, 2017. https://papers.nips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html

[11] Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., Aila, T.: Improved precision and recall metric for assessing generative models. In NIPS proceedings, 2019. https://proceedings.neurips.cc/paper/2019/hash/0234c510bc6d908b28c70ff313743079-Abstract.html