Training Classifier Gradient Penalty GAN with Codebook Architecture

Jeongik Cho
jeongik.jo.01@gmail.com

Abstract

Classifier gradient penalty GAN is a GAN proposed to perform self-supervised class-conditional data generation and clustering on unlabeled datasets. The classifier gradient penalty GAN's generator takes a continuous latent vector and a categorical latent vector as input and generates a class-conditional data point corresponding to the categorical latent vector.

In this paper, we propose to leverage the codebook architecture to improve the performance of classifier gradient penalty GAN. In the proposed architecture, the generator takes the page vector of the codebook corresponding to the index of the categorical latent vector, instead of taking the one-hot categorical latent vector directly. Unlike the codebook used in generative models with vector quantization, the codebook of the proposed architecture is not embedded with the encoder. Instead, the codebook is simply trainable and updated via generator loss like trainable parameters in the generator.

The proposed architecture improved the quality of the generated data, class-conditional data generation performance, and clustering performance of the classifier gradient penalty GAN.

1 Introduction

Recently, some generative models, such as VQGAN [3] and VQVAE [4], have used vector quantization to improve the performance of generative models. These models consist of an encoder that converts input real data into embedded vectors, a codebook that stores embedded vectors, and a generator that generates data from the codebook. The encoder is trained to reconstruct the input data, and the generator generates the data by selecting one or several embedded vectors from the codebook. This is a kind of memory structure that allows the generative model to remember the embedded vector and utilize it to generate data.

Classifier Gradient Penalty GAN (CGPGAN) [2] is an architecture-agnostic GAN that allows the model to perform self-supervised class-conditional data generation and clustering without knowing labels, optimal prior categorical probability, or metric function. CGPGAN uses a discriminator, a classifier, and

a generator to train the model. The classifier is trained with cross-entropy loss to predict the categorical latent vector of the fake data. Also, the conditional vector of real data predicted by the classifier is used to train the class-conditional GAN. When training class-conditional GAN with this classifier, the decision boundary of the classifier falls to the local optima where the density of the data is minimized. CGPGAN uses a classifier gradient penalty loss to the classifier loss to prevent the classifier's decision boundary from falling into a narrow range of local optima. Classifier gradient penalty loss regulates the gradient of the classifier's output to prevent the gradient near the decision boundary from becoming too large.

In this paper, we introduce a codebook architecture to improve the performance of CGPGAN. Instead of directly inputting a categorical latent vector into CGPGAN's generator, the proposed architecture inputs the page vector of the codebook corresponding to the index of the categorical latent vector. Page vectors are simply trainable parameters that are updated with the CGPGAN generator loss during the CGPGAN training. Therefore, unlike other generative models that use vector quantization, the proposed architecture does not require autoencoder training and encoder inference to get embedded vectors of real data.

2 Classifier Gradient Penalty GAN

In this section, we briefly describe about CGPGAN [2]. CGPGAN was proposed for class-conditional data generation and clustering on unlabeled datasets. Specifically, CGPGAN can be used under the following conditions:

1. The labels of all data are unknown.
2. Optimal categorical latent distribution is unknown.
3. Metric to measure the distance between the data is unknown.

CGPGAN uses a generator, discriminator, and classifier for training. The following equations show the losses used to train CGPGAN.

$$L_q = \lambda_{cls} L_{cls} + \lambda_{cgp} L_{cgp} \tag{1}$$

$$L_d = L^d_{adv} \tag{2}$$

$$L_g = L^g_{adv} \tag{3}$$

$$L_{cgp} = \mathbb{E}_{z,c_f}\left[\|\nabla_{G(z,c_f)}\left((1 - Q(G(z,c_f)) \cdot c_f)^2\right)\|_2^2\right] \tag{4}$$

$$L_{cls} = \mathbb{E}_{z,c_f}\left[-c_f \cdot \log(Q(G(z,c_f)))\right] \tag{5}$$

$$L^d_{adv} = \mathbb{E}_{x,z,c_f}\left[A_d(D(x) \cdot argmax\ onehot(Q(x)), D(G(z,c_f)) \cdot c_f)\right] \tag{6}$$

$$L^g_{adv} = \mathbb{E}_{z,c_f}\left[A_g(D(G(z,c_f)) \cdot c_f)\right] \tag{7}$$

Eqs. 1, 2, and 3 show classifier loss, discriminator loss, and generator loss, respectively. Classifier Q is trained with classification loss L_{cls} and classifier gradient penalty loss L_{cgp}. Discriminator D and generator G are trained only with adversarial losses L_{adv}^d and L_{adv}^g, respectively. λ_{cls} and λ_{cgp} represent classification loss weight and classifier gradient penalty loss weight, respectively.

Eqs. 4, 5, 6, and 7 show classifier gradient penalty loss, classification loss, discriminator adversarial loss, and generator adversarial loss, respectively. Q, D, and G represent classifier, discriminator, and generator respectively. z and c_f are continuous latent vector and categorical latent vector to generate fake data point $G(z, c_f)$. x represents a real data point.

In Eq. 6, $argmax\,onehot$ is a function that makes the value of the largest index in the vector 1 and the rest of the values 0 (e.g., $argmaxonehot([0.3, 0.5, 0.2]) = [0.0, 1.0, 0.0]$). A_d and A_g represent adversarial functions for discriminator and generator, respectively.

With these losses, the classifier Q is trained to predict the categorical latent vector c_f of the generated data $G(z, c_f)$, and to converge to the point where the decision boundary minimizes the probability density of the real data distribution $P(X)$. CGPGAN also trains a class-conditional GAN based on the labels of the real data predicted by the classifier (i.e., $argmax\,onehot(Q(x))$).

Furthermore, CGPGAN updates the categorical latent prior $P(C)$ with the predicted categorical probability of the real data $Q(x)$ during the training.

$$P(C) \leftarrow update(P(C), \mathbb{E}_x\left[Q(x)\right]) \tag{8}$$

Eq. 8 shows updating categorical latent prior $P(C)$ with a predicted categorical probability of real data $\mathbb{E}_x\left[Q(x)\right]$. Through Eq. 8, categorical latent prior $P(C)$ approaches predicted the categorical probability of real data. Moving average functions, such as exponential moving average, can be used as *update* functions.

However, updating categorical latent prior $P(C)$ with Eq. 8 can cause the prior probability to converge to a meaningless vector (i.e., $P(C)$ approaches the one-hot vector) early in the CGPGAN training. To prevent it, CGPGAN applies probability normalization at the beginning of training.

$$\mathbf{c} = [Q(x_1), Q(x_2), \ldots, Q(x_b)] \tag{9}$$

$$prob\,normalize(\mathbf{c}) = \mathbf{c} - batch\,average(\mathbf{c}) + \frac{1}{d_c} \tag{10}$$

Eqs. 9 and 10 shows the probability normalization of CGPGAN. In Eq. 9, $\mathbf{c}$ is predicted probability batch of real data points $x_1, x_2, \ldots, x_b$. The $batchaverage$ function calculates the batch-wise average probability, and d_c represents the dimension of the categorical latent vector. Therefore, $batchaverage(probnormalize(\mathbf{c}))$ always become uniform distribution $\left[\frac{1}{d_c}, \frac{1}{d_c}, \ldots, \frac{1}{d_c}\right]$. Thus, $\mathbf{c} = [Q(x_1), Q(x_2), \ldots, Q(x_b)]$ is replaced by $prob\,normalize(\mathbf{c})$ at the beginning of the training.

3 Training CGPGAN with Codebook Architecture

In this paper, we propose an architecture that uses codebooks to improve the performance of CGPGAN. The proposed architecture simply replaces the categorical latent vector c_f with the page vector of the codebook corresponding to the index of the categorical latent vector, instead of directly inputting the categorical vector c_f to the generator. For example, if $c_f = [0.0, 1.0, 0.0]$, the page vector of the 2nd index of the codebook would be input to the generator instead of the categorical latent vector c_f. The codebook is updated with generator losses during training, just like other trainable parameters of the generator. Therefore, one can think that the codebook is the trainable parameter of the generator. We found that this simple method improves CGPGAN's generated data quality (precision) and class-conditional data generation & clustering performance.

4 Experiments

In this section, we compared the performance of Vanilla GAN with codebook architecture, and CGPGAN with and without codebook architecture. AFHQ dataset [9] resized to 256×256 resolution was used as the training dataset.

Models are trained with NSGAN adversarial loss [1] (i.e., A_d and A_g) with R2 regularization [5]. Equalized learning rate [6] were used for all trainable parameters. Also, an exponential moving average of generator parameters with $decay\,rate = 0.999$ was used for generative performance evaluation. The model architecture is simply composed of CNNs.

In without codebook architecture, the d_z-dimensional continuous latent vector and $d_l \times d_c$ shape multiple categorical latent vectors are the input to the generator, where d_l and d_c represent label dimension and category dimension. Then the input latent vector is projected to the $4 \times 4 \times 1024$ shape feature maps.

In with codebook architecture, d_z-dimensional continuous latent vector is projected to $4 \times 4 \times 512$ shape feature maps, then $4 \times 4 \times 512$ shaped page vectors are concatenated. The codebook is $d_l \times d_c \times d_p$ shape trainable matrix, where d_p represents the dimension of the page vector. It means that there are $d_l \times d_c$ page vectors, and $d_p = 4 \times 4 \times 512/d_l$.

The following hyperparameters were used for the experiments:

$$Z \sim N(0, I_{1024})$$
$$optimizer = AdamW \begin{pmatrix} learning\,rate = 0.003 \\ weight\,decay = 0.0001 \\ \beta_1 = 0.0 \\ \beta_2 = 0.99 \end{pmatrix}$$
$$batch\,size = 8$$
$$\lambda_{r2} = 10$$

$$epoch = 120$$
$$activation\ function = Leaky\ ReLU$$
$$d_l = 4$$
$$d_c = 4$$

For CGPGAN, $P(C)$ starts to be updated after epoch 50 (i.e., the $probnormalize$ function is disabled from epoch 50), and $\lambda_{cls} = 1, \lambda_{cgp} = 10$ were used for model training.

FID [7] and precision & recall [8] were used for generative performance evaluation.

Figs. 1, 2, and 3 shows generated samples of Vanilla GAN with codebook architecture, CGPGAN without codebook architecture, and CGPGAN with codebook architecture, respectively.

First, in Fig. 1, the Vanilla GAN with codebook architecture failed to train. The quality of the generated images is not bad, but there is almost no diversity in the generated images (recall was 0.0000). In particular, Vanilla GAN with codebook architecture generated the same image when the categorical latent vectors are the same but continuous latent vectors are different. This seems to be because the codebook architecture causes memorization of the generator, which makes it vulnerable to mode collapse. This resulted in very low generative performance for Vanilla GAN with codebook architecture.

In Fig. 2, one can see that both the continuous latent vector and categorical latent vectors have an impact on the generated images in CGPGAN without codebook architecture. However, the categorical latent vector did not have a very meaningful effect. In CGPGAN without codebook architecture, it looks like a continuous latent vector determines the species and pose of the animal, and a categorical latent vector determines some of the pattern. However, the categorical latent representation is so entangled that it is difficult for humans to interpret. For example, a white dog may belong to columns 2, 3, or 6, but it would be difficult for a human to determine exactly which category it belongs to.

In Fig. 3, one can see that each image is of good quality and that both the continuous latent vector and categorical latent vectors have a meaningful impact on the generated images in CGPGAN with codebook architecture. For example, images with the same categorical latent vectors are animals of the same species with similar patterns. And images with the same continuous latent vector have similar poses, backgrounds, and zooms. In CGPGAN with codebook architecture, categorical latent representation is disentangled, so it can be interpreted by humans. For example, a human can infer that lionesses belong in column 2.

The precision of CGPGAN with codebook architecture is 0.8124, which is higher than CGPGAN without codebook, which is 0.7305. This means that the quality of images generated by CGPGAN with codebook architecture is better than that of CGPGAN without codebook. On the other hand, the recall of CGPGAN with a codebook was 0.1772, which is lower than that of CGPGAN without a codebook. This means that the diversity of data generated by CGPGAN with a codebook is less than that of CGPGAN without a codebook.

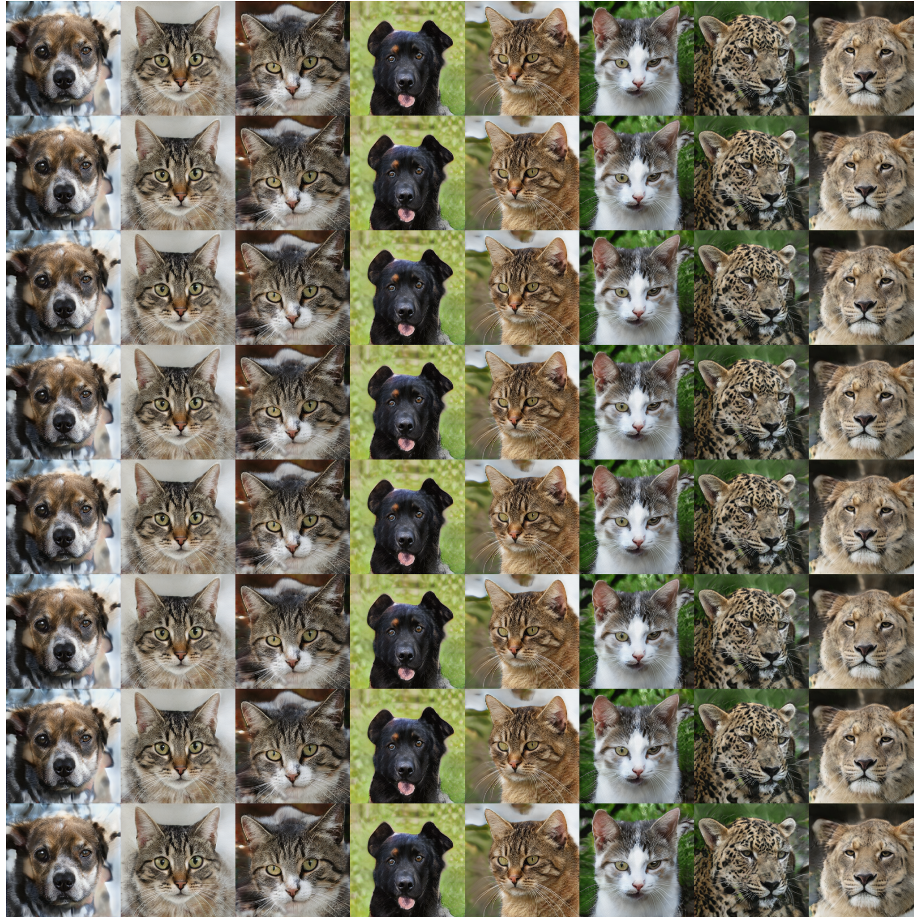

Figure 1: Generated samples of Vanilla GAN with codebook architecture. Each column has the same categorical latent vectors, and each row has the same continuous latent vector. FID:68.6567, Precision: 0.6300, Recall: 0.0000.

Figure 2: Generated samples of CGPGAN without codebook architecture. Each column has the same categorical latent vectors, and each row has the same continuous latent vector. FID: 10.7616, Precision: 0.7305, Recall:0.3060

Figure 3: Generated samples of CGPGAN with codebook architecture. Each column has the same categorical latent vectors, and each row has the same continuous latent vector. FID: 11.0781, Precision: 0.8124, Recall: 0.1772.

Still, there was little difference in the FID evaluation that represented overall generative performance.

Note that $d_l = 4$ and $d_c = 4$, so there are $4^4 = 256$ combinations of categorical latent vectors. Therefore, there are more combinations of categorical latent vectors than the samples in Figs. 1, 2, and 3.

5 Conclusion

In this paper, we introduced codebook architecture for CGPGAN. In the proposed architecture, the generator takes the page vector of the codebook corresponding to the index of the categorical latent vector, instead of taking the categorical latent vector directly. Unlike other vector quantization generative models that use encoders to make codebooks, the codebook in our proposed architecture is trained with generator loss like a trainable parameter of a generator.

In the experiments, the proposed architecture did not work with vanilla GAN. Also, the proposed architecture reduced the diversity of the generated data in CGPGAN. However, the codebook architecture improved the quality of the generated data and disentangled the categorical latent distribution, which allows humans to interpret it.

In conclusion, the proposed codebook architecture improved the class-conditional data generation and clustering performance of CGPGAN.

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] Jeongik Cho, "Training Self-supervised Class-conditional GANs with Classifier Gradient Penalty and Dynamic Prior," https://vixra.org/abs/2307.0121?ref=15805554

[3] Patrick Esser, Robin Rombach, Björn Ommer, "Taming Transformers for High-Resolution Image Synthesis," https://arxiv.org/abs/2012.09841

[4] Aaron van den Oord, Oriol Vinyals, Koray Kavukcuoglu, "Neural Discrete Representation Learning," https://arxiv.org/abs/1711.00937

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

[6] 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

[7] 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

[8] 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

[9] Y. Choi, Y. Uh, J. Yoo, J. Ha, "StarGAN v2: Diverse Image Synthesis for Multiple Domains," in CVPR 2020. https://openaccess.thecvf.com/content_CVPR_2020/papers/Choi_StarGAN_v2_Diverse_Image_Synthesis_for_Multiple_Domains_CVPR_2020_paper.pdf