# Conditional Activation GAN: Improved Auxiliary Classifier GAN

**Jeongik Cho**[1], **Kyoungro Yoon**[2]

**ABSTRACT** Conditional Generative Adversarial Network (GAN) is a GAN that generates data with the desired condition from the latent vector. The auxiliary classifier GAN is the most used among the variations of conditional GANs. In this study, we explain the problem of auxiliary classifier GAN and propose conditional activation GAN that can replace auxiliary classifier GAN to reduce the number of hyperparameters and improve training speed. The loss function of conditional activation GAN is defined as the sum of the loss of each GAN created for each condition. Since each GAN shares all hidden layers, the GANs can be considered as a single GAN and it does not increase the amount of computation much. Also, in order to prevent ignorance of conditions in the discriminator of conditional GANs with batch normalization, we propose a mixed batch training, in which each batch for discriminator is always configured to have the same ratio of real data and generated data so that each batch always has the similar condition distribution.

**INDEX TERMS** Artificial neural networks, auxiliary classifier GAN, batch normalization, conditional GAN, deep learning, generative adversarial networks, loss function

## I. INTRODUCTION

Conditional GAN [1] is a GAN [2] that can generate data with the desired condition from the latent vector. Among the variations of conditional GANs [3, 4], the most used conditional GAN is the Auxiliary Classifier GAN (AC-GAN) [5] used in [6, 7, 8, 9, 10, 11]. Some papers used a variation of AC-GAN [10, 11] without giving any details on the rationalization of the variations made. In this study, we explain the reasons for the modification of AC-GAN and the disadvantages of AC-GAN.

In AC-GAN, when real data distribution and generated data distribution is the same, auxiliary classifier of the discriminator and the generator can be considered as a group of GANs, each of which trains each condition and cross-entropy adversarial loss by sharing all hidden layers. Considering the AC-GAN as a set of GANs, the generated data classification loss of the AC-GAN discriminator loss interferes with the training of each GAN and hence is removed in the modified AC-GAN.

Since each GAN can be trained as a GAN only when the real data distribution and the generated data distribution are the same, there is a problem that individual GAN may not be trained at the beginning of the AC-GAN training.

Also, to use the advanced adversarial loss as used in papers such as Least Squares Generative Adversarial Networks(LSGAN) [12] or Wasserstein Generative Adversarial Networks-Gradient Penalty(WGAN-GP) [13] in AC-GAN, a hyperparameter that is adjusting the ratio of adversarial loss and classification loss should be decided.

We propose a Conditional Activation Generative Adversarial Networks (CA-GAN) that can replace AC-GAN to reduce the number of hyperparameters and improve training speed to overcome the upper mentioned problems of AC-GAN. Loss of CA-GAN is the sum of the losses of each GAN when each GAN is created for each condition. Since each GAN shares all hidden layers, the CA-GAN composed on a conceptual aggregation of individual GAN can be considered as a single GAN.

Unlike AC-GAN's use of two losses (adversarial loss, classification loss), CA-GAN uses only one loss (conditional activation loss), so there is no need to find the proper ratio of adversarial loss and classification loss.

Also, while AC-GAN starts to train each condition when the real data distribution is the same to the generated data distribution, CA-GAN always trains each condition simultaneously, which means that CA-GAN always produces meaningful gradients, even in the early training stage.

In conditional GANs, training by applying batch normalization [14] to the discriminator induces the generator to distort the input condition distribution.

When batch normalization is applied to the discriminator, and the real data and the generated data condition distribution are different, the discriminator may use the batch condition distribution for real/fake discrimination and the generated data condition distribution follows the real data condition distribution, not the input target condition distribution.

To prevent the generator from ignoring the input target condition distribution, we suggest mixed batch training. Mixed batch training is to always configure each batch for discriminator with the same ratio of real data and generated data so that each batch always has the similar condition distribution.

However, if mixed batch training is applied at the beginning of training, the generated data and the original data within a single batch are trivial for the discriminator to classify and there are hardly anything to be trained. Therefore, the ratio of real data and generated data of each batch is gradually changed to the target ratio.

## II. Analysis of Auxiliary classifier GAN

The loss of AC-GAN is defined as follows:

$$L_d = L_{adv}^d + L_{cls}^r + L_{cls}^g \tag{1}$$

$$L_g = L_{adv}^g + L_{cls}^r + L_{cls}^g \tag{2}$$

$$L_{cls}^r = E_{x,cnd\sim P_r(x,cnd)}[-\log D_{cls}(cnd|x)] \tag{3}$$

$$L_{cls}^g = E_{x',cnd'\sim P_g(x',cnd')}[-\log D_{cls}(cnd'|x')] \tag{4}$$

$$L_{adv}^d = E_{x\sim P_r(x)}[-\log D_{adv}(x)]$$
$$+E_{x\sim P_g(x)}[-\log(1-D_{adv}(x))] \tag{5}$$

$$L_{adv}^g = E_{x\sim P_g(x)}[\log(1-D_{adv}(x))] \tag{6}$$

In (1) and in (2), $L_d$ is the loss of the discriminator and $L_g$ is the loss of the generator. $L_{adv}^d$ is the adversarial loss of the discriminator and $L_{adv}^g$ is the adversarial loss of the generator. In (5), $D_{adv}$ is the probability distribution function of the data in the adversarial module. $D_{adv}(x)$ is the probability distribution of $x$, which is given as the input of the adversarial module. $E$ is the expectation of the given variable. Symbol "~" means "is distributed as". For example, $E_{x\sim P_z(x)}[f(x)]$ is an expectation value of $f(x)$ when $x$ follows the distribution of $P_z(x)$.

In $x, cnd\sim P_r(x, cnd)$ of (3), $x$ is the real data, and $cnd$ is the binary vector that expresses the conditions of real data. In $x', cnd'\sim P_g(x', cnd')$ of (4), $x'$ is the generated data and $cnd'$ is the target binary vector to generate $x'$. $D_{cls}(x)$ is the probability distribution of data $x$ within auxiliary classifier of the discriminator. $-\log D_{cls}(cnd|x)$ is the cross-entropy loss between $cnd$ and $D_{cls}(x)$. Minimizing $-\log D_{cls}(cnd|x)$ means that $D_{cls}$ is trained to estimate the conditions of $x$ ($cnd$) well.

Note that $L_{cls}^r$ in $L_g$ does not play any role because the generator does not affect the calculation of $L_{cls}^r$.

In AC-GAN, when real data distribution and generated data distribution is the same, auxiliary classifier of the discriminator and the generator can be considered as a group of GANs that each GAN trains each condition using cross-entropy adversarial loss, and shares all hidden layers as shown in Fig. 1.

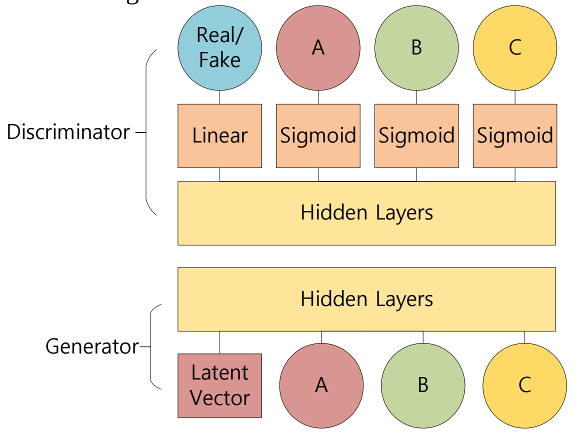

Figure 1. AC-GAN that trains A, B, and C conditions

Suppose that AC-GAN training three independent conditions (A, B, C) trains only with adversarial loss, and the real data distribution and the generated data distribution are the same.

Node A of the discriminator is trained by $L_{cls}^r[A]$ in $L_d$ to output 1 to represent real when it receives real data with condition A, and 0 to represent fake with condition not-A.

When the generator receives 1 as its node A's input, it attempts to generate data by $L_{cls}^g[A]$ in $L_g$ with condition A, and trains the discriminator's node A output to be 1.

If the generator attempts to generate data with condition A but fails, the generated data distribution will be close to the real data distribution with condition not-A since it is assumed that the real data distribution and the generated data distribution are the same.

Thus, the hidden layers of the discriminator and node A, the hidden layers of the generator and the latent vector input, and node A can be thought of as a single GAN A that generates data with condition A trained by $L_{cls}^r[A]$ in $L_d$ and $L_{cls}^g[A]$ in $L_g$. However, $L_{cls}^g[A]$ in $L_d$ trains node A of the discriminator to be 1 representing real when the discriminator receives generated data. Therefore, $L_{cls}^g[A]$ in $L_d$ interferes with the training of GAN A.

Also, when the generator receives 0 as its node A's input, it can be thought of as a GAN that generates data with condition not-A.

AC-GAN uses cross-entropy loss as an adversarial loss. However, in order to use advanced adversarial loss such as LSGAN or WGAN-GP, a hyperparameter is needed to adjust the ratio of adversarial loss and classification loss.

To solve these problems, the loss of the modified AC-GANs used in StarGAN [10] or AttGAN [11] is modified as follows:

$$L_d = L_{adv}^d + \lambda_{cls} L_{cls}^r \tag{7}$$

$$L_g = L_{adv}^g + \lambda_{cls} L_{cls}^g \quad (8)$$

$$L_{cls}^r = E_{x,cnd \sim P_r(x,cnd)}[-\log D_{cls}(cnd|x)] \quad (9)$$

$$L_{cls}^g = E_{x',cnd' \sim P_g(x',cnd')}[-\log D_{cls}(cnd'|x')] \quad (10)$$

In (7) and in (8), $L_d$ is loss of discriminator and $L_g$ is loss of generator. $L_{adv}^d$ is adversarial loss of discriminator and $L_{adv}^g$ is adversarial loss of generator. In $x, cnd \sim P_r(x, cnd)$ of (9), $x$ is real data, and $cnd$ is the binary vector that expresses the conditions of real data. In $x', cnd' \sim P_g(x', cnd')$ of (10), $x'$ is generated data and $cnd'$ is the target binary vector to generate $x'$. $\lambda_{cls}$ is classification loss weight.

As explained above, modified AC-GAN also can be considered as a group of GANs. However, each GAN can only be trained as a GAN for each condition only if the real data distribution and the generated data distribution for the corresponding condition are the same.

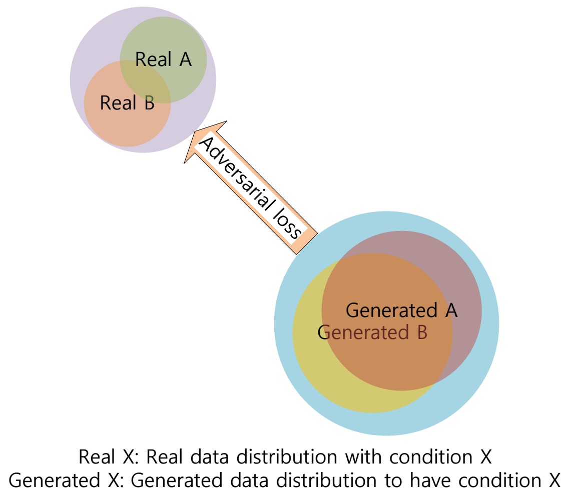
Real A
Real B
Adversarial loss
Generated A
Generated B
Real X: Real data distribution with condition X
Generated X: Generated data distribution to have condition X

Figure 2. Data distribution at the beginning of training using AC-GAN

In other words, if the real data distribution differs from the generated data distribution at the beginning of the training, the training does not proceed with classification loss, but only with adversarial loss, as shown in Fig.2.

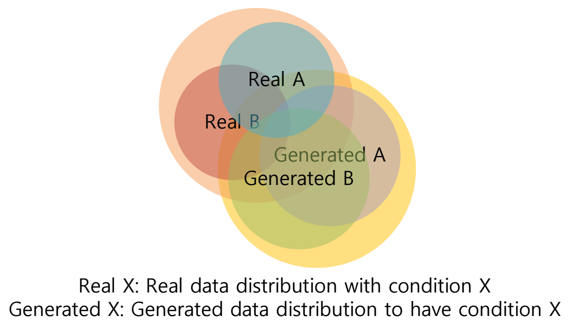
Real A
Real B
Generated A
Generated B
Real X: Real data distribution with condition X
Generated X: Generated data distribution to have condition X

Figure 3. Distribution of generated data after some training using AC-GAN

By training with adversarial loss, the real data distribution and the generated data distribution gets closer. As these distributions get closer to each other, the classification loss gradually acts as the cross-entropy adversarial loss of each GAN, and produces meaningful gradients and training is performed to generate data with each condition.

AC-GAN has the disadvantage of requiring one additional hyperparameter to adjust the ratio of adversarial loss and classification loss in both discriminator and generator and not producing meaningful gradients early stage of training.

## III. Conditional activation GAN (CA-GAN)

To solve these problems of AC-GAN, we propose CA-GAN, which is similar to having multiple GANs each of which is defined to train corresponding condition.

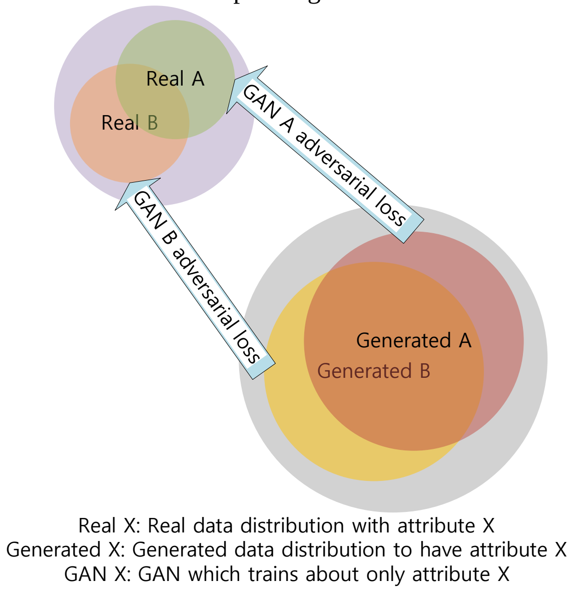
Real A
Real B
GAN A adversarial loss
GAN B adversarial loss
Generated A
Generated B
Real X: Real data distribution with attribute X
Generated X: Generated data distribution to have attribute X
GAN X: GAN which trains about only attribute X

Figure 4. Data distribution at the beginning of training using CA-GAN

Loss of conditional activation GAN is the sum of each GAN's loss where Each GAN trains only one condition as defined in the following equation.

$$L_d = \sum_{for\ each\ c\ in\ S_{cnd}} L_{d_c} \quad (11)$$

$$L_g = \sum_{for\ each\ c\ in\ S_{cnd}} L_{g_c} \quad (12)$$

$$L_{d_c} = E_{x,c \sim P_r(x,c)}\left[f_r^d\left(D_c(x)\right)\right] + E_{x' \sim P_{g_c}(x',1)}\left[f_g^d\left(D_c(x')\right)\right] \quad (13)$$

$$L_{G_c} = E_{x' \sim P_{g_c}(x',1)}\left[f^g\left(D_c(x')\right)\right] \quad (14)$$

In (11) and in (12), $L_d$ and $L_g$ represent the discriminator and the generator losses of conditional activation GAN, respectively. $S_{cnd}$ represents the set of conditions that the given CA-GAN is intended to be trained for. $c$ is one specific condition in $S_{cnd}$. GAN $c$ is an individual GAN that trains for only condition $c$.

$g_c$ and $d_c$ are generator and discriminator of GAN $c$. $g_c$ receives a binary activation value with a latent vector. If $g_c$ receives 1 as an activation value, $g_c$ tries to trick $d_c$, and $d_c$ tries to discriminate generated data from $g_c$ as fake. If $g_c$ receives 0 as the activation value, both $g_c$ and $d_c$ do not care about what has been generated. $d_c$ only cares about discriminating real data, which has condition $c$, and does not care about other real data including real data with condition not-$c$.

In $x, c \sim P_r(x, c)$ of (13), $x$ is the real data which has condition $c$. In $x' \sim P_{g_c}(x', 1)$, $x'$ is generated data by $g_c$ when it receives latent vector with 1 as activation value.

$f_r^d$ is a function that calculates the adversarial loss of the discriminator about real data. $f_g^d$ is a function that calculates the adversarial loss of the discriminator about generated data. In (14), $f^g$ is a function that calculates the adversarial loss of the generator.

The following equation is an example of the adversarial loss of GAN $c$ that uses adversarial loss given in LSGAN [12].

$$L_{d_c} = E_{x,c \sim P_r(x,c)}[(D_c(x) - 1)^2]$$

$$+E_{x' \sim P_{g_c}(x',1)}[D_c(x')^2] \quad (15)$$

$$L_{g_c} = E_{x' \sim P_{g_c}(x',1)}[(D_c(x') - 1)^2] \quad (16)$$

In CA-GAN, since each GAN shares all hidden layers, conditional activation loss can be changed as the following equation.

$$L_d = E_{x,cnd \sim P_r(x,cnd)}\left[f_r^d\big(D(x)\big) \cdot cnd\right]$$

$$+E_{x',cnd' \sim P_g(x',cnd')}\left[f_g^d\big(D(x')\big) \cdot cnd'\right] \quad (17)$$

$$L_g = E_{x',cnd' \sim P_g(x',cnd')}\left[f^g\big(D(x')\big) \cdot cnd'\right] \quad (18)$$

In $x, cnd \sim P_r(x, cnd)$ of (17), $x$ is real data, and $cnd$ is the binary vector that expresses the conditions of real data. In $x', cnd' \sim P_g(x', cnd')$ of (18), $x'$ means generated data, and $cnd'$ is the target binary vector to make $x'$. "·" is an inner product (element-wise product, then sum).

The following equation is the loss of CA-GAN when it is using the adversarial loss of LSGAN.

$$L_d = E_{x,cnd \sim P_r(x,cnd)}[(D(x) - 1)^2 \cdot cnd]$$

$$+E_{x',cnd' \sim P_g(x',cnd')}\left[\big(D(x')\big)^2 \cdot cnd'\right] \quad (19)$$

$$L_g = E_{x',cnd' \sim P_g(x',cnd')}[(D(x') - 1)^2 \cdot cnd'] \quad (20)$$

Likewise, the loss of CA-GAN when using the adversarial loss of WGAN-GP can be defined as the following equation.

$$L_d = E_{x,cnd \sim P_r(x,cnd)}[-D(x) \cdot cnd]$$

$$+E_{x',cnd' \sim P_g(x',cnd')}[D(x') \cdot cnd'] + \lambda_{gp} gp\_loss \quad (21)$$

$$gp\_loss = E_{\hat{x}}\left[\left(\left\|\nabla_{\hat{x}} average\big(D(\hat{x})\big)\right\|_2 - 1\right)^2\right] \quad (22)$$

$$L_g = E_{x',cnd' \sim P_g(x',cnd')}[-D(x') \cdot cnd'] \quad (23)$$

In (21), $\lambda_{gp}$ and $gp\_loss$ represents gradient penalty loss weight and gradient penalty loss, respectively. $gp\_loss$ is an average of each GAN's gradient penalty loss. In (22), $\hat{x}$ is data uniformly sampled from straight line between $x$ and $x'$.

In AC-GAN, GAN A that trains condition A also generates data with condition not-A as well as data with condition A.

However, in CA-GAN, since GAN A, training with condition A, does not care about condition not-A, a new GAN training condition not-A must be added to train condition not-A.

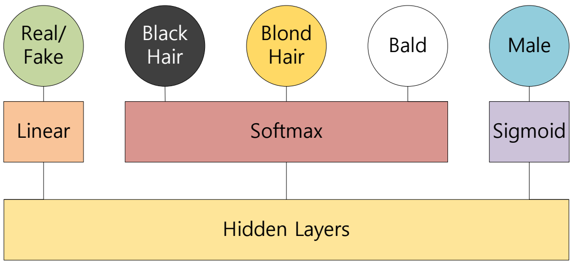

Figure 5. Example of AC-GAN discriminator output part

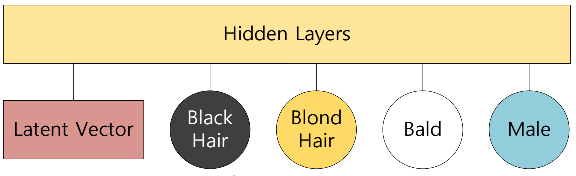

Figure 6. Example of AC-GAN generator input part

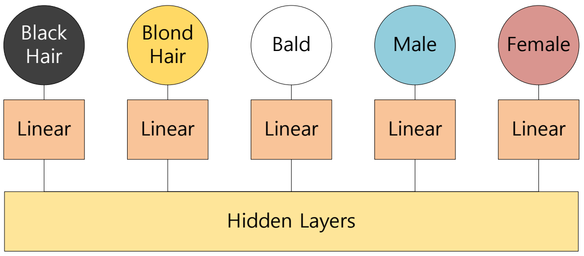

Figure 7. Example of CA-GAN discriminator output part

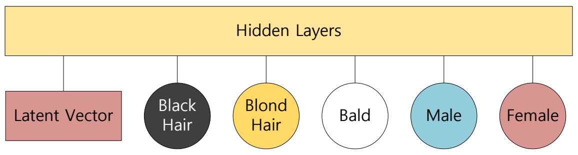

Figure 8. Example of CA-GAN generator input part

(Assume $P(Black\ hair) + P(Blond\ hair) + P(Bald) = 1, P(Male) + P(Female) = 1$)

In CA-GAN, since each GAN can be trained through advanced adversarial loss that generates meaningful gradients even if the real data distribution and the generated data distribution are different, meaningful gradients are generated even at the beginning of the training.

Also, unlike AC-GAN's use of two losses (adversarial loss, classification loss), CA-GAN uses only one loss (conditional activation loss), so there is no need to find the proper ratio of adversarial loss and classification loss. This means that it takes less time to search for an important hyperparameter: the ratio of adversarial loss and classification loss.

## IV. Mixed batch training

In conditional GANs, training by applying batch normalization to the discriminator may induce the generator to distort the input condition distribution.

When batch normalization is applied to the discriminator and the target condition distribution used for training is different from the real data condition distribution, the discriminator may use the batch condition distribution for real/fake discrimination, which leads generated data condition distribution to follow real data condition distribution. To prevent the generator from ignoring the input target condition distribution, we suggest mixed batch training.

Mixed batch training is configuring each batch for discriminator always to have the same ratio of real data and generated data so that each batch always has the similar condition distribution. Since each training batch is always configured to keep the same condition distribution, the discriminator will not discriminate real/fake by condition distribution, and the generator will not attempt to follow the real data condition distribution.

However, if mixed batch training is applied at the beginning of training, for the discriminator to discriminate the generated data from the original data in the batch is arbitrary and the training does hardly proceed. Therefore, the ratio of real data and generated data of each batch is gradually changed to the target ratio. In other words, at the beginning of training, the "real data:generated data" of each batch is "100:0 and 0:100". As training progresses, this ratio changes to" 20:80 and 80:20", "40:60 and 60:40", and finally "50:50 and 50:50". The ratio of real data:generated data that changes for each epoch or iteration is an additional hyperparameter used for mixed batch training.

Also, if you want to train generator and discriminator unbalanced, the target ratio may not be 50:50, but in general, 50:50 is used.

## V. Material and methods

### A. MNIST dataset

In the MNIST experiment, we used a dataset of the MNIST handwriting number dataset [15]. The dataset has 60000 training images and 10000 test images with an image resolution of 28 × 28 pixels, and the channel size is 1.

The basic design of DCGAN [16] with instance normalization is used for the model architecture.

The generator receives a 10-dimensional condition vector and a 256-dimensional latent vector that follows a normal distribution. While AC-GAN uses all 11 outputs of the discriminator, CA-GAN uses only 10 outputs.

Adversarial loss of LSGAN was used for both AC-GAN and CA-GAN. Adam optimizer ($learning\ rate = 10^{-5}, beta1 = 0.9, beta2 = 0.999$) [17], $batch\ size = 32, epochs = 50$ were used for all experiments. $\lambda_{cls}$ for AC-GAN is 0.1, which is the best hyperparameter for AC-GAN we found. For the implementation, tensorflow2.0 is used [18].

For the evaluation of the proposed network, an average of Fréchet Inception Distance (FID) [19] over all conditions is used.

All the experiments were conducted three times and the result of three experiments are averaged. The size of the generated data set is the same as the size of each test dataset in evaluation.

In all generated pictures below, each row has the same latent vector, and each column has the same condition.

## VI. Experimental Results and Discussion

### A. AC-GAN

We compared the performance of modified AC-GAN with or without $L^{g}_{cls}$ in discriminator loss when the adversarial loss exists.

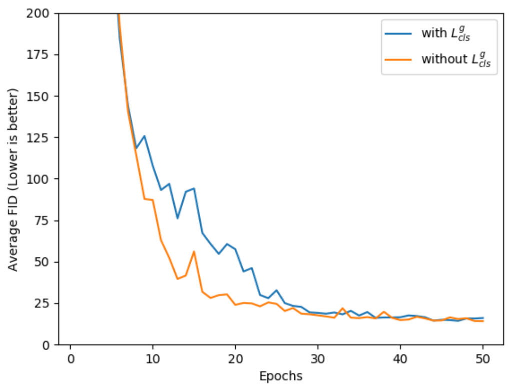

Figure 9. Effect of $L^{g}_{cls}$ in modified AC-GAN performance

In Fig. 9, the blue graph shows the average FID of modified AC-GAN with $L_d = L_{adv}^d + \lambda_{cls}(L_{cls}^r + L_{cls}^g), \lambda_{cls} = 0.1$ and the orange graph shows the average FID of modified AC-GAN with $L_d = L_{adv}^d + \lambda_{cls} L_{cls}^r, \lambda_{cls} = 0.1$. As the graph shows, the performance of the network without $L_{cls}^g$ is better.

The next experiment is to compare the performance when the adversarial loss weight and classification loss weight are different in modified AC-GAN.

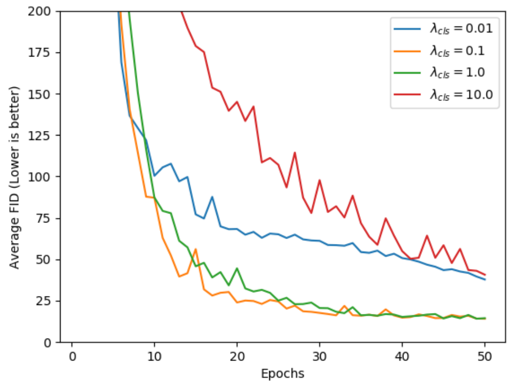

Figure 10. Modified AC-GAN performance comparison with different weight of adversarial loss and classification loss

Fig. 10 shows the FID when the classification loss weight $\lambda_{cls}$ varies from 0.01 to 10.0, with the adversarial loss weight fixed to 1.0. The changes in training speed and the quality of the results as the ratio of the adversarial loss weight and the classification loss weight changes can be easily seen through this graph.

## B. CA-GAN

We compared proposed CA-GAN with modified AC-GAN.

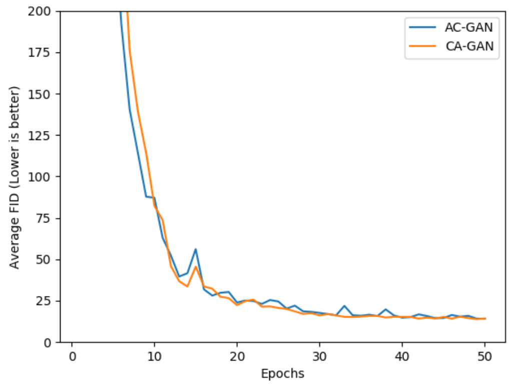

Figure 11. Performance comparison of modified AC-GAN vs CA-GAN

Fig. 11 shows that the performance of CA-GAN is similar to that of the modified AC-GAN when we use a good hyperparameter ($\lambda_{cls}$).

## C. Mixed batch training

In the original MNIST handwriting number training dataset, the number of images for each number is almost the same. For the experiment, we intentionally used a dataset consisting of 5500 of number 0 and 500 of other numbers 1~9 each from the MNIST handwriting number training dataset, to create an unbalanced dataset. The number 0 in the dataset occupies 55% of the total 10000 data, and the remaining numbers 1~9 accounts for 5% each. Since the train data size has been reduced to $\frac{1}{6}$, we used $Adam(learning\ rate = 3 \times 10^{-5}, beta1 = 0.9, beta2 = 0.999), epoch = 100$. FID was measured every 2 epochs.

We applied batch normalization in the discriminator instead of instance normalization in this experiment.

$ratio\ change\ per\ epoch = 0.01$ was used for mixed batch training. That is, for each epoch, real data:generated data changes by 1%p(100:0 and 0:100 in epoch 1, 90:10 and 10:90 in epoch 11, 70:30 and 30:70 in epoch 31, 50:50 after epoch 51).

Fig.12 and Fig.13 show the data generated by modified AC-GAN and CA-GAN without mixed batch training, respectively. The generated data clearly show that both modified AC-GAN and CA-GAN ignore conditional vectors and generate a lot of number zeros following the distribution of the original training dataset.

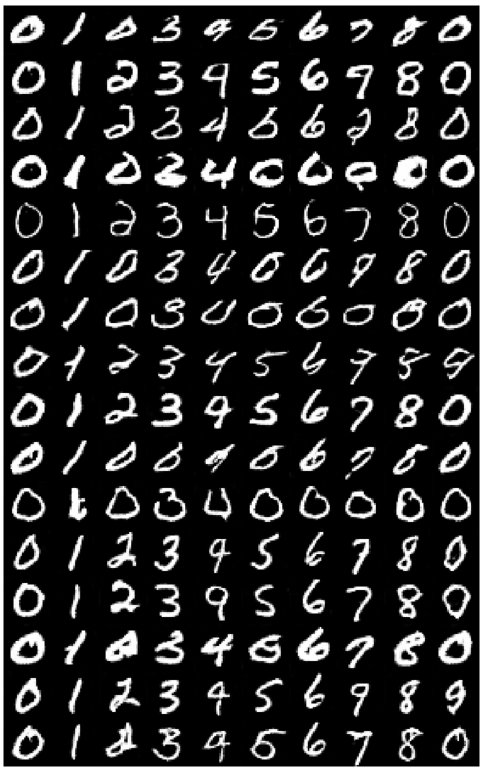

Figure 12. Data generated by modified AC-GAN after 100 epochs without mixed batch training

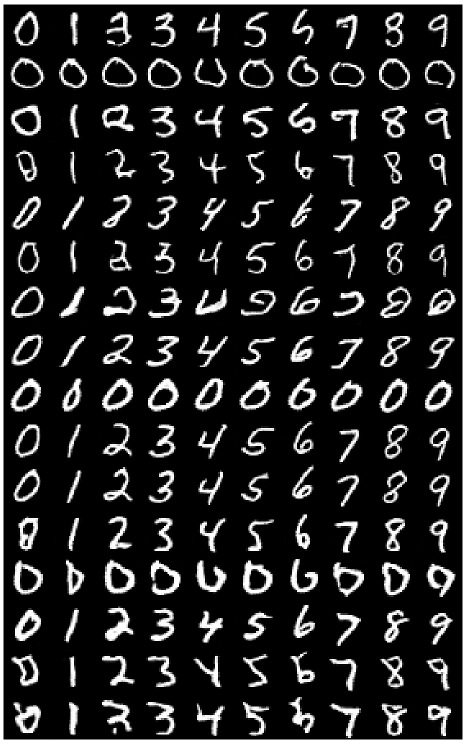

Figure 13. Data generated by CA-GAN after 100 epochs without mixed batch training

Fig. 14 and Fig. 15 show the data generated by the modified AC-GAN and CA-GAN with mixed batch training, respectively. And Fig. 16 shows performance comparison of these two cases based on FID measure.

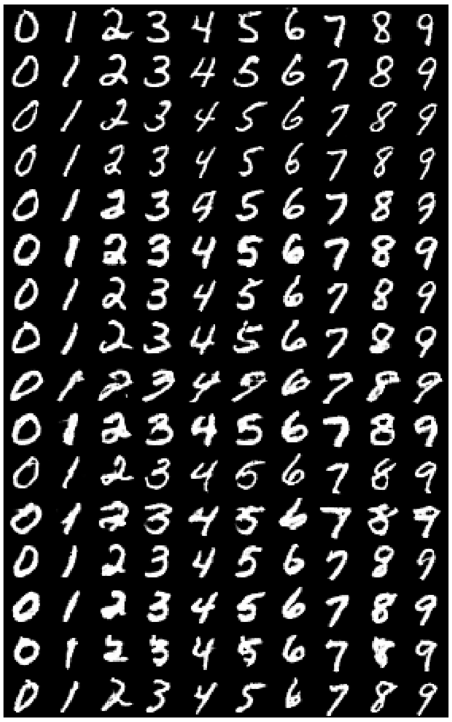

Figure 14. Data generated by modified AC-GAN after 100 epochs with mixed batch training

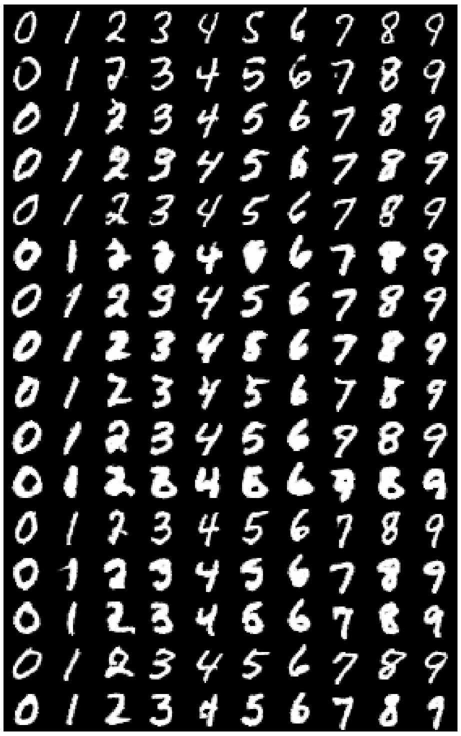

Figure 15. Data generated by CA-GAN after 100 epochs with mixed batch training

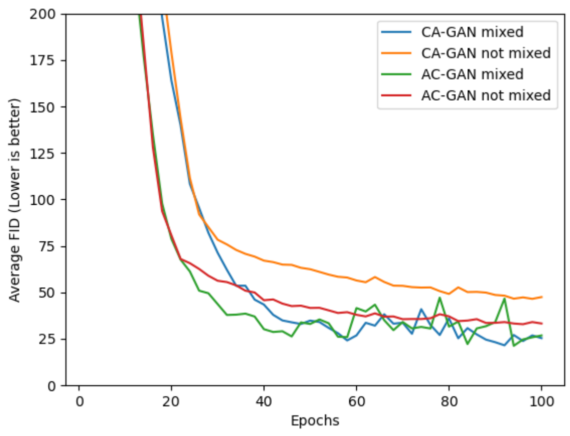

Figure 16. Mixed batch training performance comparison

These results shown in Fig. 12~16 clearly show that the performance of mixed batch training is better than not using it for both modified AC-GAN and CA-GAN.

On the other hand, in this experiment, it can be seen that the performance of CA-GAN is lower than that of modified AC-GAN when mixed batch training is not applied. It seems that AC-GAN works better for data with unbalanced conditions because each GAN trains 'not-c' as well as some condition 'c'. However, when mixed batch training is applied, the difference in performance between modified AC-GAN and CA-GAN almost disappears.

## VII. Conclusion

In this paper, we tried to interpret AC-GAN as a set of GANs and explained why generated data classification loss of discriminator loss in AC-GAN interferes with training and confirmed this theory through the experiments.

Based on this interpretation, we proposed a novel approach of GAN, called Conditional Activation GAN(CA-GAN). CA-GAN can be interpreted as an integration of GANs in which each individual GAN trains only one condition. Unlike modified AC-GAN, CA-GAN generates a meaningful gradient even at the beginning of the training, so that the training speed is fast, as shown in the experiments.

CA-GAN is expected to be used as a replacement for modified AC-GAN in many GAN applications because it has fewer hyperparameters and trains faster than modified AC-GAN, while it is compatible with AC-GAN.

We also predicted that the discriminator with batch normalization might use batch condition distribution to discriminate real/fake, which would cause performance degradation, in conditional GAN.

To prevent this degradation, we proposed mixed batch training. The mixed batch training is configuring each batch for discriminator with the same ratio of real data and generated data so that each batch always has the similar condition distribution. The ratio of real data:generated data gradually changes to target ratio during the training. Through experiments, the performance improvement of conditional GANs: modified AC-GAN and CA-GAN, due to mixed batch training is confirmed. Mixed batch training is expected to help train conditional GANs using batch normalization for discriminators.

In conclusion, CA-GAN, which we propose in this paper, provides better performance than AC-GAN in terms of training speed and hyperparameter search. The mixed batch training also improves performance of conditional GAN by inducing healthy competition between generator and discriminator.

## VIII. Appendix

Fig. 20 and 21 show the architecture of GAN for all experiments.

| |
|---|
| Condition vector ([10]), Latent vector ([256]) |
| Fully connected layer ([7 * 7 * 256], Leaky ReLU) |
| Reshape ([7, 7, 256]) |
| Instance normalization () |
| Up sampling 2D () |
| Convolution layer (128, [3, 3], Leaky ReLU) |
| Instance normalization () |
| Convolution layer (128, [3, 3], Leaky ReLU) |
| Instance normalization () |
| Convolution layer (128, [3, 3], Leaky ReLU) |
| Instance normalization () |
| Up sampling 2D () |
| Convolution layer (64, [3, 3], Leaky ReLU) |
| Instance normalization () |
| Convolution layer (64, [3, 3], Leaky ReLU) |
| Instance normalization () |
| Convolution layer (64, [3, 3], Leaky ReLU) |
| Instance normalization () |
| Convolution layer (1, [1, 1], Tanh) |

Figure 20. Generator architecture

| | |
|---|---|
| Input image ([28, 28, 1]) | |
| Convolution layer (64, [3, 3], Leaky ReLU) | |
| Instance normalization () or Batch normalization () | |
| Average pooling () | |
| Convolution layer (128, [3, 3], Leaky ReLU) | |
| Instance normalization () or Batch normalization () | |
| Convolution layer (128, [3, 3], Leaky ReLU) | |
| Instance normalization () or Batch normalization () | |
| Convolution layer (128, [3, 3], Leaky ReLU) | |
| Instance normalization () or Batch normalization () | |
| Average pooling () | |
| Convolution layer (256, [3, 3], Leaky ReLU) | |
| Instance normalization () or Batch normalization () | |
| Convolution layer (256, [3, 3], Leaky ReLU) | |
| Instance normalization () or Batch normalization () | |
| Convolution layer (256, [3, 3], Leaky ReLU) | |
| Instance normalization () or Batch normalization () | |
| Flatten () | |
| Fully connected layer ([1], Linear),<br>Fully connected layer ([10], Linear) | |
| Linear ([1]),<br>Softmax ([10])<br>for AC-GAN | Not use ([1]),<br>Linear ([10])<br>for CA-GAN |

Figure 21. Discriminator architecture

## REFERENCES

[1] M. Mirza, and S. Osindero, "Conditional Generative Adversarial Nets", 2014, arXiv:1411.1784. [Online]. Available: https://arxiv.org/abs/1411.1784

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets", Advances in Neural Information Processing Systems 27 (NIPS), 2014, pp. 2672-2680. [Online]. Available: https://papers.nips.cc/paper/5423-generative-adversarial-nets

[3] T. Kaneko, K. Hiramatsu, and K. Kashino, "Generative Attribute Controller With Conditional Filtered Generative Adversarial Networks", The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 6089-6098. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2017/html/Kaneko_Generative_Attribute_Controller_CVPR_2017_paper.html

[4] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets", Advances in Neural Information Processing Systems 29 (NIPS), 2016, pp. 2172-2180. [Online]. Available: http://papers.nips.cc/paper/6399-infogan-interpretable-representation

[5] A. Odena, C. Olah, C. Olah, J. B Shlens, and J. Shlens, “Conditional image synthesis with auxiliary classifier GANs”, ICML'17: Proceedings of the 34th International Conference on Machine Learning – Volume 70, 2017, pp. 2642-2651. [Online]. Available: https://dl.acm.org/doi/10.5555/3305890.3305954

[6] L. Zhang, Y. Ji, X. Lin and C. Liu, “Style Transfer for Anime Sketches with Enhanced Residual U-net and Auxiliary Classifier GAN”, 2017 4th IAPR Asian Conference on Pattern Recognition (ACPR), Nanjing, 2017, pp. 506-511. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8575875

[7] X. Xia, R. Togneri, F. Sohel and D. Huang, “Auxiliary Classifier Generative Adversarial Network With Soft Labels in Imbalanced Acoustic Event Detection”, IEEE Transactions on Multimedia, vol. 21, no. 6, pp. 1359-1371, June 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8523637

[8] P. Sattigeri, S. C. Hoffman, V. Chenthamarakshan, and K. R. Varshney, “Gated-GAN: Adversarial Gated Networks for Multi-Collection Style Transfer”, IEEE Transactions on Image Processing, vol. 28, no. 2, pp. 546-560, Feb. 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8463508

[9] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, “GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification”, Neurocomputing, Volume 321, 2018, Pages 321-331, ISSN 0925-2312. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0925231218310749

[10] Z. He, W. Zuo, M. Kan, S. Shan and X. Chen, “AttGAN: Facial Attribute Editing by Only Changing What You Want” IEEE Transactions on Image Processing, vol. 28, no. 11, pp. 5464-5478, Nov. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8718508

[11] Y. Choi, M. Choi, M. Kim, J. Ha, S. Kim, and J. Choo, “StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation”, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 8789-8797. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Choi_StarGAN_Unified_Generative_CVPR_2018_paper.html

[12] X. Mao, Q. Li, H. Xie, R. Y.K. Lau, Z. Wang, S. P. Smolley, “Least Squares Generative Adversarial Networks”, The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2794-2802. [Online]. Available: https://ieeexplore.ieee.org/document/8237566

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved Training of Wasserstein GANs”, Advances in Neural Information Processing Systems 30 (NIPS), 2017, pp. 5767-5777. [Online]. Available: http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans

[14] S. Ioffe, and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:448-456, 2015. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[15] Y. LeCun, C. Cortes, and C. J.C. Burges, “THE MNIST DATABASE of handwritten digits”. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[16] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, arXiv preprint arXiv:1511.06434v2 [cs.LG], 2015. [Online]. Available: https://arxiv.org/abs/1511.06434

[17] D. P. Kingma, and J. Ba, “Adam: A Method for Stochastic Optimization”, arXiv preprint arXiv:1412.6980v9 [cs.LG], 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[18] Tensorflow 2.0. [Online]. Available: http://www.tensorflow.org

[19] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”, Advances in Neural Information Processing Systems 30 (NIPS), 2017, pp. 6626-6637. [Online]. Available: http://papers.nips.cc/paper/7240-gans-trained-by-a-two-t

[20] http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html