GAN — Ways to improve GAN performance

Jonathan Hui · Follow

Published in Towards Data Science · 13 min read · Jun 19, 2018

Photo by Andy Beales

GAN models can suffer badly in the following areas comparing to other deep networks.

- **Non-convergence**: the models do not converge and worse they become unstable.
- **Mode collapse**: the generator produces limited modes, and
- **Slow training:** the gradient to train the generator vanished.

As part of the GAN series, this article looks into ways on how to improve GAN. In particular,

- Change the cost function for a better optimization goal.
- Add additional penalties to the cost function to enforce constraints.
- Avoid overconfidence and overfitting.
- Better ways of optimizing the model.
- Add labels.

But be aware that this is a dynamic topic as research remains highly active.

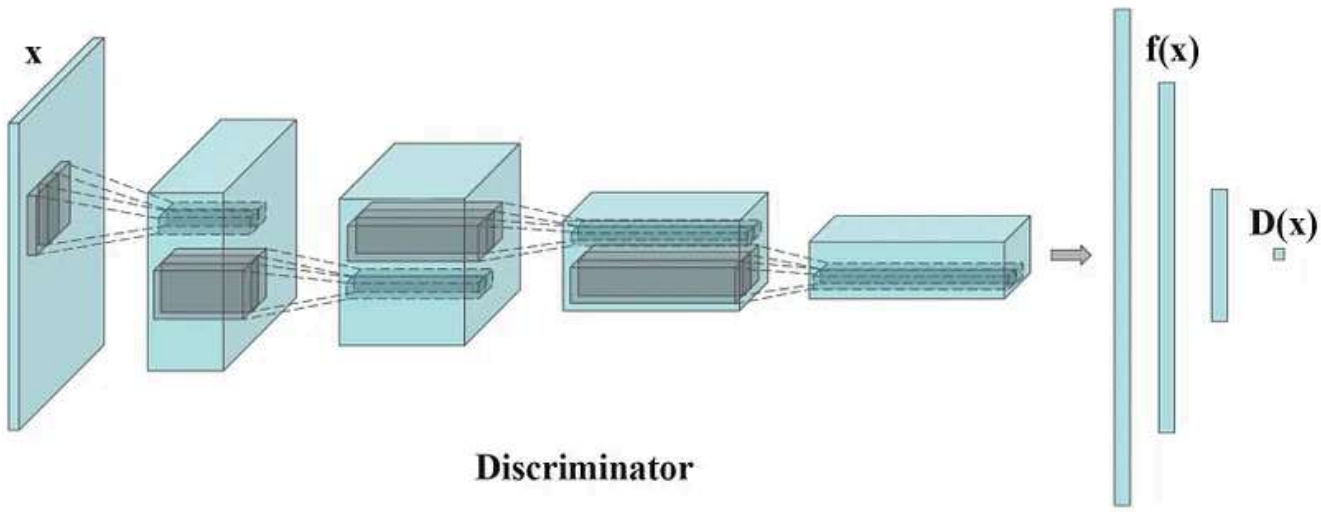
Feature Matching

The generator tries to find the best image to fool the discriminator. The "best" image keeps changing when both networks counteract their opponent. However, the optimization can turn too greedy and fall it into a never-ending cat-and-mouse game. This is one of the scenarios that the model does not converge and mode collapses.

Feature matching changes the cost function for the generator to minimizing the statistical difference between the features of the real images and the generated images. Often, we measure the L2-distance between the means of their feature vectors. Therefore, feature matching expands the goal from beating the opponent to matching features in real images. Here is the new objective function:

$$||\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\mathbf{f}(G(\boldsymbol{z}))||_2^2$$

where *f(x)* is the feature vector extracted in an immediate layer by the discriminator.



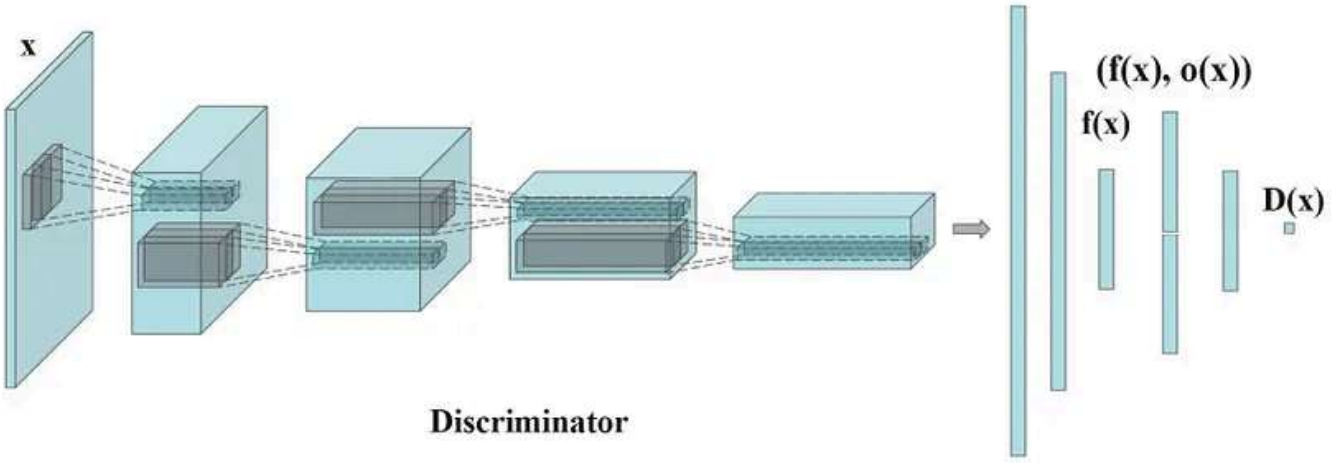
The means of the real image features are computed per minibatch which fluctuate on every batch. It is good news in mitigating the mode collapse. It introduces randomness that makes the discriminator harder to overfit itself.

> *Feature matching is effective when the GAN model is unstable during training.*

Minibatch discrimination

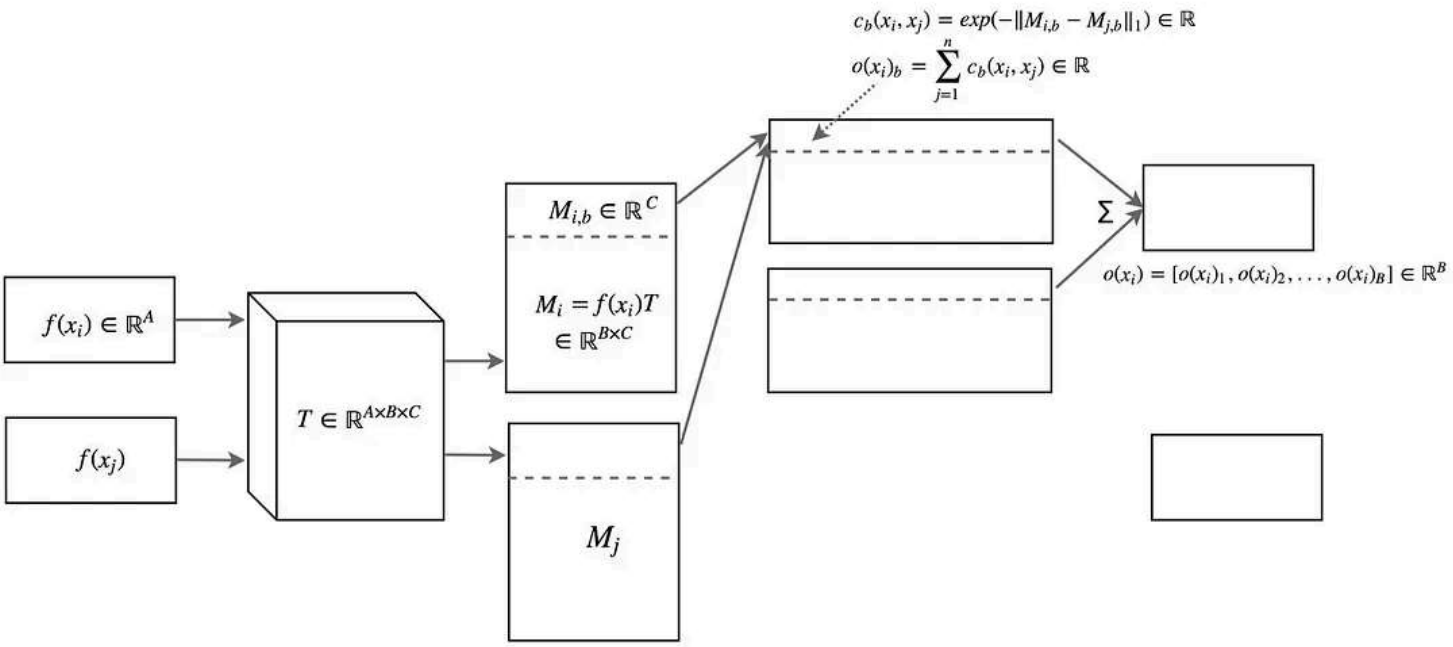
When mode collapses, all images created looks similar. To mitigate the problem, we feed real images and generated images into the discriminator separately in different batches and compute the similarity of the image x with images in the same batch. We append the similarity $o(x)$ in one of the dense layers in the discriminator to classify whether this image is real or generated.

$$[f(x_i), o(x_i)]$$



If the mode starts to collapse, the similarity of generated images increases. The discriminator can use this score to detect generated images and penalize the generator if mode is collapsing.

The similarity $o(xi)$ between the image xi and other images in the same batch is computed by a transformation matrix T. The equations are a little bit hard to trace but the concept is pretty simple. But feel free to skip to next section if you want.



In the figure above, ***xi*** is the input image and ***xj*** is the rest of the images in the same batch. We use a transformation matrix ***T*** to transform the features ***xi*** to ***Mi*** which is a B×C matrix.

$$M_i = f(x_i)T$$

We derive the similarity ***c(xi, xj)*** between image ***i*** and ***j*** using the L1-norm and the following equation.

$$c_b(x_i, x_j) = exp(-\|M_{i,b} - M_{j,b}\|_1) \in \mathbb{R}$$

The similarity ***o(xi)*** between image ***xi*** and the rest of images in the batch is

$$o(x_i)_b = \sum_{j=1}^{n} c_b(x_i, x_j) \in \mathbb{R}$$
$$o(x_i) = [o(x_i)_1, o(x_i)_2, \ldots, o(x_i)_B] \in \mathbb{R}^B$$

Here is the recap:

$$f(x_i) \in \mathbb{R}^A$$
$$T \in \mathbb{R}^{A \times B \times C}$$
$$M_i = f(x_i)T \in \mathbb{R}^{B \times C}$$
$$c_b(x_i, x_j) = exp(-\|M_{i,b} - M_{j,b}\|_1) \in \mathbb{R}$$
$$o(x_i)_b = \sum_{j=1}^{n} c_b(x_i, x_j) \in \mathbb{R}$$
$$o(x_i) = [o(x_i)_1, o(x_i)_2, \ldots, o(x_i)_B] \in \mathbb{R}^B$$

As a quote from the paper “Improved Techniques for Training GANs”

> *Minibatch discrimination allows us to generate visually appealing samples very quickly, and in this regard it is superior to feature matching.*

One-sided label smoothing

Deep networks may suffer from overconfidence. For example, it uses very few features to classify an object. To mitigate the problem, deep learning uses regulation and dropout to avoid overconfidence.

In GAN, if the discriminator depends on a small set of features to detect real images, the generator may just produce these features only to exploit the discriminator. The optimization may turn too greedy and produces no long term benefit. In GAN, overconfidence hurts badly. To avoid the problem, we penalize the discriminator when the prediction for any real images go

beyond 0.9 (*D(real image)>0.9*). This is done by setting our target label value to be 0.9 instead of 1.0. Here is the pseudo code:

```
p = tf.placeholder(tf.float32, shape=[None, 10])

# Use 0.9 instead of 1.0.
feed_dict = {
  p: [[0, 0, 0, 0.9, 0, 0, 0, 0, 0, 0]] # Image with label "3"
}

# logits_real_image is the logits calculated by
# the discriminator for real images.
d_real_loss = tf.nn.sigmoid_cross_entropy_with_logits(
                    labels=p, logits=logits_real_image)
```

Historical averaging

In historical averaging, we keep track of the model parameters for the last t models. Alternatively, we update a running average of the model parameters if we need to keep a long sequence of models.

We add an L2 cost below to the cost function to penalize model different

$$\|\theta - \frac{1}{t}\sum_{i=1}\theta[i]\|_2$$

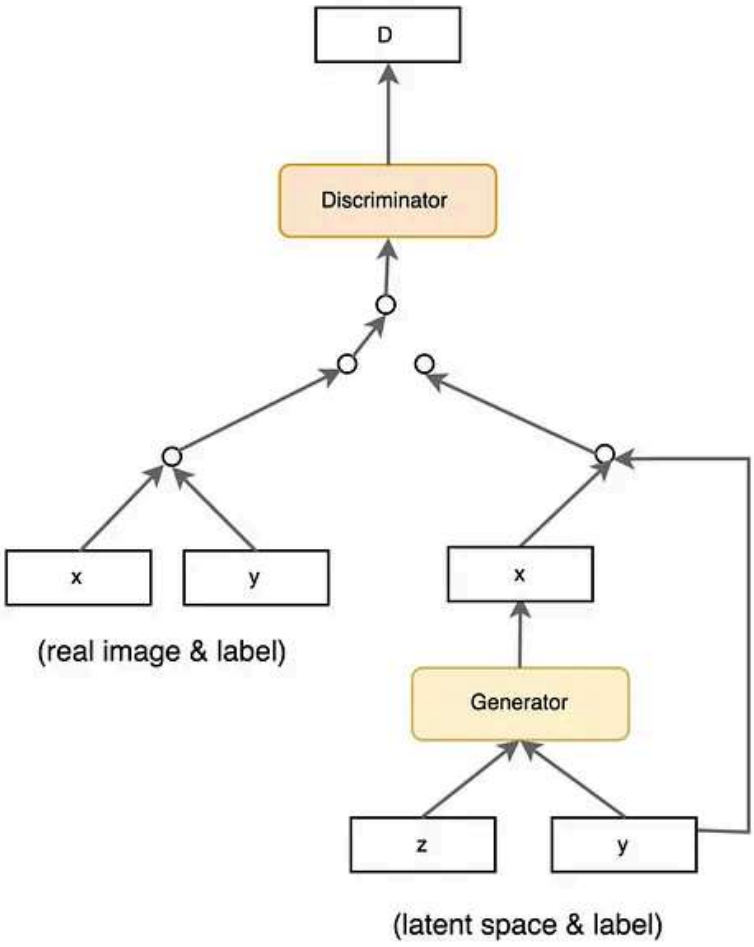
For GANs with non-convex object function, historical averaging may stop models circle around the equilibrium point and act as a damping force to converge the model.

Experience replay

The model optimization can be too greedy in defeating what the generator is currently generating. To address this problem, experience replay maintains the most recent generated images from the past optimization iterations. Instead of fitting the models with current generated images only, we feed the discriminator with all recent generated images also. Hence, the discriminator will not be overfitted for a particular time instance of the generator.

Using labels (CGAN)

Many datasets come with labels for the object type of their samples. Training GAN is already hard. So any extra help in guiding the GAN training can improve the performance a lot. Adding the label as part of the latent space z helps the GAN training. Below is the data flow used in CGAN to take advantage of the labels in the samples.



Cost functions

Do cost functions matter? It must be otherwise all those research efforts will be a waste. But if you hear about a 2017 Google Brain paper, you will definitely have doubts. But pushing image quality is still a top priority. Likely, we will see researchers trying different cost functions before we have a definite answer for the merit.

The following figure lists the cost functions for some common GAN models.

Name	Value Function
GAN	$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$ $L_G^{GAN} = E[\log(D(G(z)))]$
LSGAN	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN_GP	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha G(z)))\rvert - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha x_p))\rvert - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	$L_D^{CGAN} = E[\log(D(x, c))] + E[\log(1 - D(G(z), c))]$ $L_G^{CGAN} = E[\log(D(G(z), c))]$
infoGAN	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c, c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c, c')$
ACGAN	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c\lvert x)] + E[P(class = c\lvert G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c\lvert G(z))]$
EBGAN	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

Table modified from here.

We decide not to detail these cost functions in this article. Here are the articles that covers some common cost functions in details: WGAN/WGAN-

GP, EBGAN/BEGAN, LSGAN, RGAN and RaGAN. At the end of this article, we list an article that studies all these cost functions in more details. Since cost function is one major research area in GAN, we do encourage you to read that article later.

Here is some of the FID score (the lower the better) on some of the datasets. This is one reference point but be warned that it is still too early to make any conclusion on what cost functions perform the best. Indeed, there is no single cost function that performs the best among all different datasets yet.

	MNIST	FASHION	CIFAR	CELEBA
MM GAN	9.8 ± 0.9	29.6 ± 1.6	72.7 ± 3.6	65.6 ± 4.2
NS GAN	6.8 ± 0.5	26.5 ± 1.6	58.5 ± 1.9	55.0 ± 3.3
LSGAN	7.8 ± 0.6*	30.7 ± 2.2	87.1 ± 47.5	53.9 ± 2.8*
WGAN	6.7 ± 0.4	21.5 ± 1.6	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	20.3 ± 5.0	24.5 ± 2.1	55.8 ± 0.9	30.0 ± 1.0
DRAGAN	7.6 ± 0.4	27.7 ± 1.2	69.8 ± 2.0	42.3 ± 3.0
BEGAN	13.1 ± 1.0	22.9 ± 0.9	71.4 ± 1.6	38.9 ± 0.9
VAE	23.8 ± 0.6	58.7 ± 1.2	155.7 ± 11.6	85.7 ± 3.8

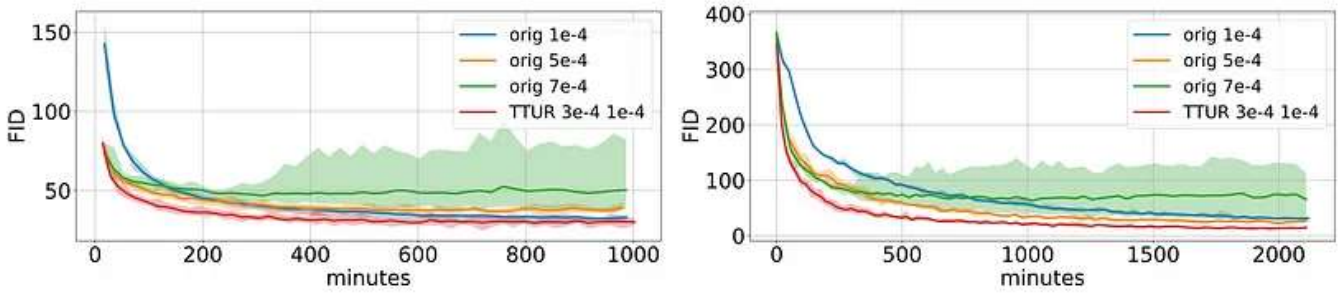
Source

(MM GAN is the GAN cost function in the original paper. NS GAN is the alternative cost functions addressing the vanishing gradients in the same paper.)

But no model performs well without good hyperparameters and tuning GANs takes time. Be patient in the hyperparameters optimization before randomly testing different cost functions. Some researchers had suggested that tunning the hyperparameters may ripe a better return than changing the cost functions. A carefully tunned learning rate may mitigate some

serious GAN's problems like mode collapse. In specific, lower the learning rate and redo the training when mode collapse happens.

We can also experiment with different learning rates for the generator and the discriminator. For example, the following graph use the learning rate of 0.0003 for the discriminator and 0.0001 for the generator in the WGAN-GP training.



Source

Implementation tips

- Scale the image pixel value between -1 and 1. Use tanh as the output layer for the generator.
- Experiment sampling z with Gaussian distributions.
- Batch normalization often stabilizes training.
- Use PixelShuffle and transpose convolution for upsampling.
- Avoid max pooling for downsampling. Use convolution stride.
- Adam optimizer usually works better than other methods.
- Add noise to the real and generated images before feeding them into the discriminator.

The dynamics of the GAN models are not well understood yet. So some of the tips are just suggestions and the mileage may vary. For example, the LSGAN paper reports RMSProp has more stable training in their experiments. This is kind of rare but demonstrates the challenges of making generic recommendations.

The discriminator and the generator are constantly competing with others. Be prepared that the cost function value may go up and down. Don't stop the training pre-maturely even the cost may seem to trend up. Monitor the results visually to verify the progress of the training.

Virtual batch normalization (VBN)

Batch normalization BM becomes a de facto standard in many deep network design. The mean and the variance of BM is derived from the current minibatch. However, it creates a dependency between samples. The generated images are not independent of each other.

$$x_i = G(z_i, z_1, z_2, \dots, z_n) \text{ instead of } x_i = G(z_i)$$

This is reflected in experiments which generated images show color tint in the same batch.

Orange tone on the top batch and greenish tone on the second. Source

Originally, we sample z from a random distribution that gives us independent samples. However, the bias created by the batch normalization overwhelm the randomness of z.

Virtual batch normalization (VBN) samples a reference batch before the training. In the forward pass, we can preselect a reference batch to compute the normalization parameters (μ and σ) for the BN. However, we will overfit the model with this reference batch since we use the same batch over the whole training. To mitigate that, we can combine a reference batch with the current batch to compute the normalization parameters.

Random seeds

The random seeds used to initialize the model parameters impact the performance of GAN. As shown below, the FID scores in measuring the GAN performance vary in 50 individual runs (training). But the range is relatively small and likely to be done in later fine tuning only.

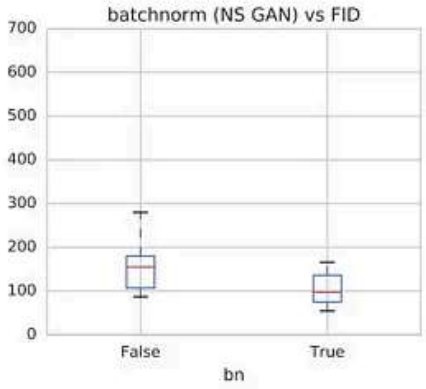
	MNIST	FASHION	CIFAR	CELEBA
MM GAN	9.8 ± 0.9	29.6 ± 1.6	72.7 ± 3.6	65.6 ± 4.2
NS GAN	6.8 ± 0.5	26.5 ± 1.6	58.5 ± 1.9	55.0 ± 3.3
LSGAN	7.8 ± 0.6*	30.7 ± 2.2	87.1 ± 47.5	53.9 ± 2.8*
WGAN	6.7 ± 0.4	21.5 ± 1.6	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	20.3 ± 5.0	24.5 ± 2.1	55.8 ± 0.9	30.0 ± 1.0
DRAGAN	7.6 ± 0.4	27.7 ± 1.2	69.8 ± 2.0	42.3 ± 3.0
BEGAN	13.1 ± 1.0	22.9 ± 0.9	71.4 ± 1.6	38.9 ± 0.9
VAE	23.8 ± 0.6	58.7 ± 1.2	155.7 ± 11.6	85.7 ± 3.8

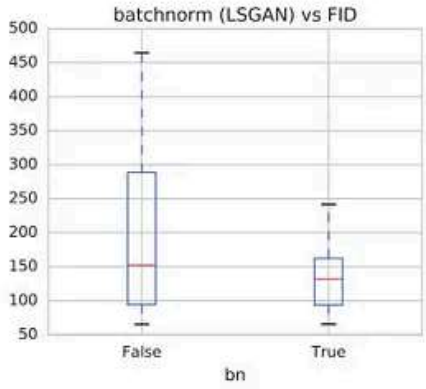
Source

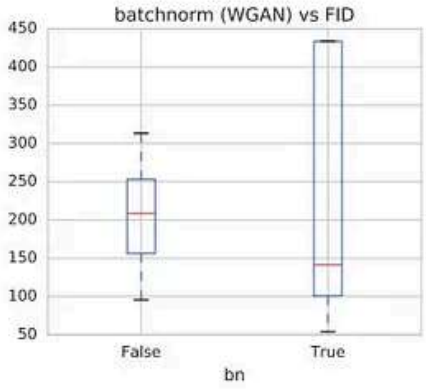
A Google Brain paper indicates LSGAN occasionally fails or collapses in some dataset, and training needs to be restarted with another random seed.

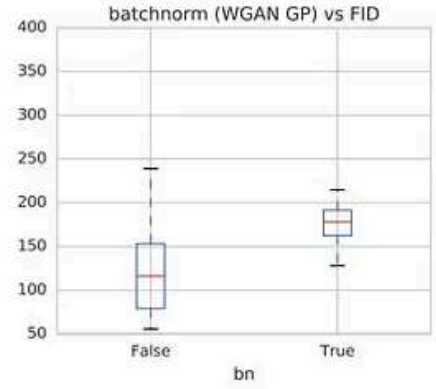
Batch normalization

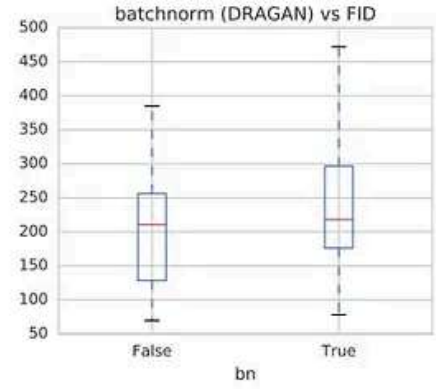
DGCAN strongly recommends adding BM into the network design. The use of BM also become a general practice in many deep network model. However, there will be exceptions. The following figure demonstrates the impact of BN on different dataset. The y-axis is the FID score which the lower the better. As suggested by the WGAN-GP paper, BN should be off when it is used. We suggest readers to check the cost function used and the corresponding FID performance on BN, and verify the setting with experiments.

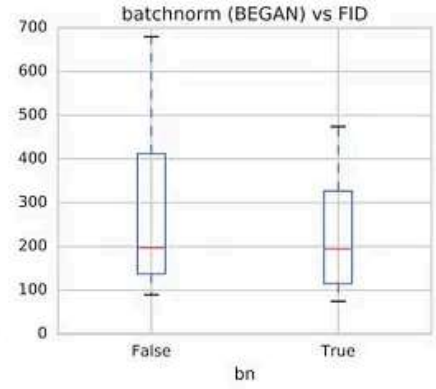

batchnorm (NS GAN) vs FID
False
True
bn


batchnorm (LSGAN) vs FID
False
True
bn

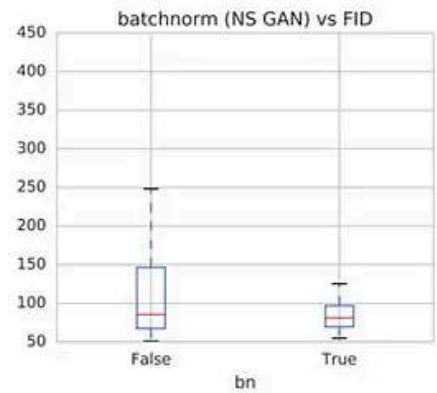

batchnorm (WGAN) vs FID
False
True
bn

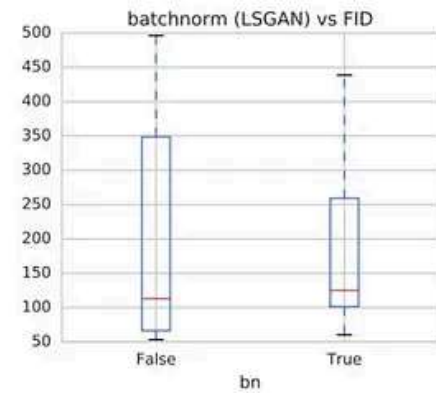

batchnorm (WGAN GP) vs FID
False
True
bn

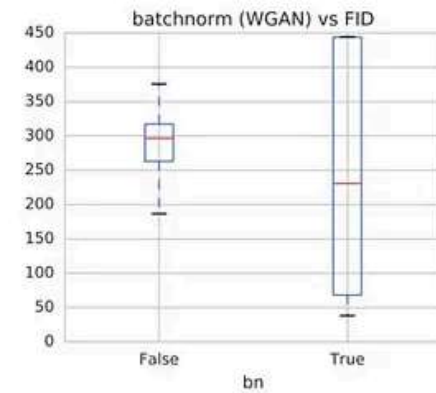

batchnorm (DRAGAN) vs FID
False
True
bn

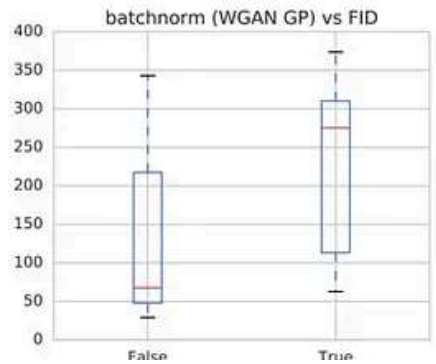

batchnorm (BEGAN) vs FID
False
True
bn

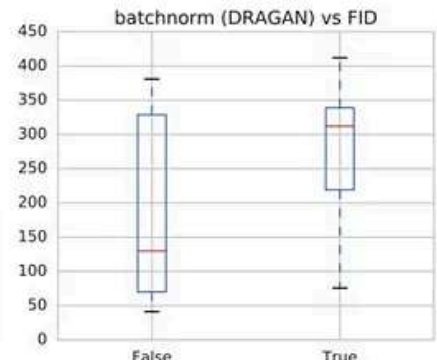
CIFAR10

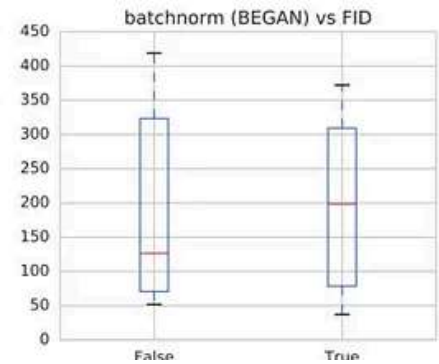

batchnorm (NS GAN) vs FID
False
True
bn

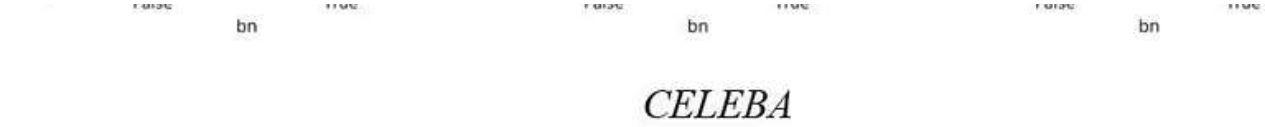

batchnorm (LSGAN) vs FID
False
True
bn


batchnorm (WGAN) vs FID
False
True
bn


batchnorm (WGAN GP) vs FID
False
True


batchnorm (DRAGAN) vs FID
False
True


batchnorm (BEGAN) vs FID
False
True



CELEBA

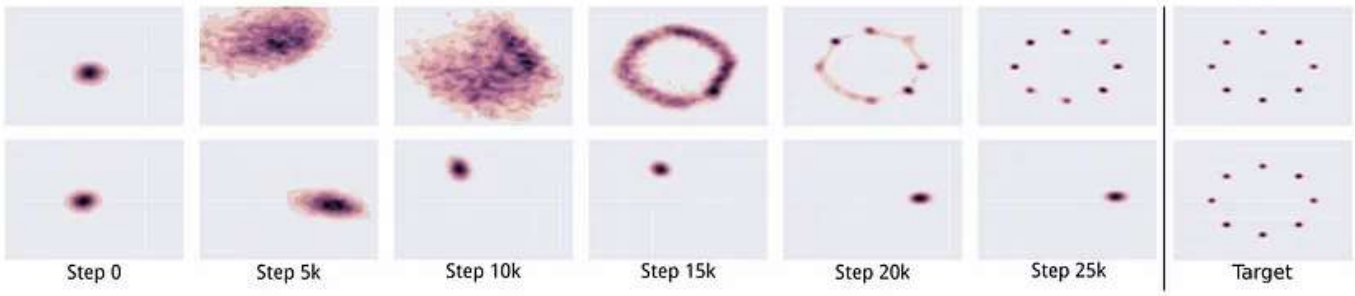
Modified from source.

Spectral Normalization

Spectral Normalization is a weight normalization that stabilizes the training of the discriminator. It controls the Lipschitz constant of the discriminator to mitigate the exploding gradient problem and the mode collapse problem. The concept is based heavily on maths but conceptually, it restricts the weight changes in each iteration and not over depending on a small set of features in distinguishing images by the discriminator. This approach will be computationally light compared with WGAN-GP and achieve good mode coverage that haunts many GAN methods.

Multiple GANs

Mode collapse may not be all bad. The image quality often improves when mode collapses. In fact, we may collect the best model for each mode and use them to recreate different modes of images.



Source

Balance between discriminator & generator

The discriminator and generator are always in a tug of war to undercut each other. Mode collapse and gradient diminishing are often explained as an imbalance between the discriminator and the generator. We can improve GAN by turning our attention in balancing the loss between the generator and the discriminator. Unfortunately, the solution seems elusive. We can maintain a static ratio between the number of gradient descent iterations on the discriminator and the generator. Even this seems appealing but many doubt its benefit. Often, we maintain a one-to-one ratio. But some researchers also test out a ratio of 5 discriminator iterations per generator update. Balancing both networks with dynamic mechanics is also proposed. But not until recent years, we get some traction on it.

On the other hand, some researchers challenge the feasibility and desirability of balancing these networks. A well-trained discriminator gives quality feedback to the generator anyway. Also, it is not easy to train the generator to always catch up with the discriminator. Instead, we may turn the attention into finding a cost function that does not have a close-to-zero gradient when the generator is not performing well.

$$-\nabla_{\theta_g} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \rightarrow 0 \quad \textit{change to} \quad \nabla_{\theta_g} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$$

Nevertheless, issues remain. Many cost function proposals are made and the debates on what is the best remain.

Discriminator & generator network capacity

The model for the discriminator is usually more complex than the generator (more filters and more layers) and a good discriminator gives quality information. In many GAN applications, we may run into bottlenecks where

increasing generator capacity shows no quality improvement. Until we identify the bottlenecks and resolve them, increasing generator capacity does not seem to be a priority for many partitioners.

BigGAN

BigGAN was published in 2018 with the goal of pulling together some practices for GAN in generating the best images at that time. In this section, we will study some of the practices that not yet covered.

Larger batch size

Batch	Ch.	Param (M)	Shared	Skip-z	Ortho.	Itr $\times 10^3$	FID	IS
256	64	81.5	SA-GAN Baseline			1000	18.65	52.52
512	64	81.5	✗	✗	✗	1000	15.30	58.77(±1.18)
1024	64	81.5	✗	✗	✗	1000	14.88	63.03(±1.42)
2048	64	81.5	✗	✗	✗	732	12.39	76.85(±3.83)
2048	96	173.5	✗	✗	✗	295(±18)	9.54(±0.62)	92.98(±4.27)
2048	96	160.6	✓	✗	✗	185(±11)	9.18(±0.13)	94.94(±1.32)
2048	96	158.3	✓	✓	✗	152(±7)	8.73(±0.45)	98.76(±2.84)
2048	96	158.3	✓	✓	✓	165(±13)	8.51(±0.32)	99.31(±2.10)
2048	64	71.3	✓	✓	✓	371(±7)	10.48(±0.10)	86.90(±0.61)

Source (The smaller the FID score, the better)

Increase the batch size can have a significant drop in FID as shown above. With a bigger batch size, more modes are covered and provide better gradients for both networks to learn. But yet, BigGAN reports the model reaches better performance in fewer iterations, but become unstable and even collapse afterward. So, save the model constantly.

Truncation Trick

Low probability density region in the latent space z may not have enough training data to learn it accurately. So when generating images, we can avoid

those regions to improve the image quality at the cost of the variation. i.e. the quality of images will increase but those generated images will have lower variance in style. There are different techniques to truncate the input latent space z. The general principle is when values fall outside a range, it will be resampled or squeeze to the higher-probability region.

Increase model capacity

During tuning, consider increasing the capacity of the model, in particular for layers with high-spatial resolutions. Many models show improvement when double the traditional capacity used at the time. But don't do it too early without proofing the model design and implementation first.

Moving averages of Generator weights

The weights used by the generator are computed from an exponential moving average of the weights of the generator.

Orthogonal regularization

The condition of the weight matrix is a heavy studied topic. This is a study on how sensitive a function output is to changes in its input. It has a large impact on training stability. A matrix Q is orthogonal if

$$Q^{\mathrm{T}}Q = QQ^{\mathrm{T}} = I$$

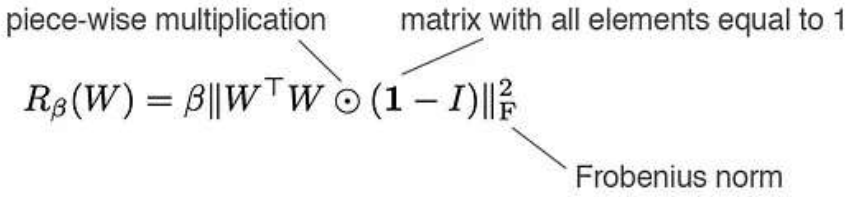
If we multiply x with an orthogonal matrix, the changes in x will not be magnified. This behavior is very desirable for maintaining numerical stability.

$$\|Qx\|_2^2 = (Qx)^T Qx = x^T Q^T Qx = x^T x = \|x\|_2^2$$

With other properties, maintain the orthogonal properties of the weight matrix can be appealing in deep learning. We can add an orthogonal regularization to encourage such properties during training. It penalizes the system if Q deviates from being an orthogonal matrix.

$$R_\beta(W) = \beta \|W^\top W - I\|_{\mathrm{F}}^2$$

Nevertheless, this is known to be too limiting and therefore BigGAN uses a modified term:



The orthogonal regularization also allows the truncation trick to be more successful across different models.

Orthogonal weight initialization

The model weight is initialized to be a random orthogonal matrix.

Skip-z connection

In the vanilla GAN, the latent factor z is input to the first layer only. With skip-z connection, direct skip connections (skip-z) from the latent factor z is

connected to multiple layers of the generator rather than just the first layer.

Further readings

In this article, we do not detail the improvement through the cost function. This is an important topic and we recommend readers to read the article below:

GAN — A comprehensive review into the gangsters of GANs (Part 2)

This article studies the motivation and the direction of the GAN research in improving GANs. By reviewing them in a...

medium.com

To know more cool applications of GANs:

GAN — Some cool applications of GANs.

We make impressive progress in the first few years of GAN developments. No more stamp-size facial pictures like those...

medium.com

All the articles in this series.

GAN — GAN Series (from the beginning to the end)

A full listing of our articles covers the applications of GAN, the issues, and the solutions.

medium.com