Inverting generator of GAN

through direction embedding discriminator

Jeongik Cho

jeongik.jo.01@gmail.com

Abstract

Generators in generative adversarial networks map latent distributions into data distributions. GAN inversion is mapping data distribution to latent distribution by inverting the generator of GAN.

In this paper, I introduce a direction embedding discriminator GAN in which the discriminator learns the inverse mapping of the generator. In the suggested method, when the latent vector is sampled from an i.i.d. (independent and identically distributed) random variable, the latent vector is considered as angular coordinates of spherical coordinates. Thus, the latent vector can be transformed into a point on the surface of the hypersphere in cartesian coordinates.

Discriminator embeds the generated data point into cartesian coordinates. The direction of embedded coordinates represents predicted cartesian coordinates of latent vector, and the log of magnitude represents an adversarial value (real/fake). The generator and discriminator are trained cooperative to decrease the angle between the embedded cartesian coordinates from the discriminator and the cartesian coordinates converted from the latent vector considered as angular coordinates of spherical coordinates. The suggested method can be applied during GAN training, does not require additional encoder training, and does not use a reconstruction loss.

1. Introduction

It is very useful to learn the inverse transform of the generator of GAN. It can perform the role of feature learning or can be used for various useful applications such as data manipulation. Many useful applications and methods for GAN inversion are introduced in the GAN inversion survey paper [1].

In this paper, I introduce direction embedding discriminator GAN that a discriminator learns the inverse mapping of a generator during the GAN training. Unlike the previous methods of training an additional encoder with a reconstruction loss after training GAN, the suggested method does not require an additional encoder and does not use a reconstruction loss.

2. Training of direction embedding discriminator GAN

In this section, I introduce a method for the discriminator D to learn the inverse mapping of the generator G.

In suggested method, when the latent vector z_s is sampled from i.i.d. random variable, the latent vector z_s is considered as angular coordinates of spherical coordinates. When the radius is set to 1, and the dimension of the latent vector is d_{z_s}, the latent vector z_s can be considered as spherical coordinates and can be transformed into a point on the surface of the hypersphere defined in $d_{z_s}+1$ dimension cartesian coordinates, whose center is 0 and the radius is 1. Discriminator D embeds the generated data $G(z_s)$ into $d_{z_s}+1$ dimensional cartesian coordinates. The direction of embedded coordinates represents cartesian coordinates of latent vector, and the log of magnitude represents an adversarial value (real/fake).

Both the generator G and the discriminator D are trained to decrease the angle between the cartesian latent vector z_c and the vector embedded by the discriminator.

The discriminator is trained to lower the log magnitude of the embedded vector when the input is real data and to raise it when input is generated data. The generator is trained to lower the log magnitude of the embedded vector of generated data.

The following algorithm shows the process of obtaining the generator and discriminator loss of the proposed method.

$function\ GetLoss(D, G, x)$:

1 $z_s \leftarrow sample\left(U\left(0, \frac{\pi}{2}\right)^{d_{z_s}}\right)$

2 $z_c \leftarrow spherical\ to\ cartesian(z_s)$

3 $z' \leftarrow D\big(G(z_s)\big)$

4 $v_g \leftarrow \log(\|z'\|_2)$

5 $v_r \leftarrow \log(\|D(x)\|_2)$

6 $z_c' \leftarrow \frac{z'}{\|z'\|_2}$

7 $L_{emb} \leftarrow tan\big(arccos(z_c \cdot z_c')\big)$

8 $L_d \leftarrow f_r^d(v_r) + f_g^d(v_g) + \lambda_{emb} L_{emb}$

9 $L_g \leftarrow f_g(v_g) + \lambda_{emb} L_{emb}$

10 $return\ L_d, L_g$

Algorithm 1. Loss function to train direction embedding discriminator GAN

In the above algorithm, D, G, and x represent discriminator, generator, and real data, respectively.

In line 1, $U\left(0, \frac{\pi}{2}\right)^{d_{z_c}}$ is a d_{z_s} dimensional i.i.d. random variable following uniform distribution in range $\left(0, \frac{\pi}{2}\right)$. $sample$ is a function that samples a single value from a random variable. z_s is a (spherical) latent vector. Also, z_s is the input of generator G.

In line 2, z_c is cartesian latent vector.

$spherical\ to\ cartesian$ is a function that converts spherical coordinates to cartesian coordinates. All input values of this function exist within the range $\left(0,\frac{\pi}{2}\right)$, and the radius for the function is set to 1. That is, all elements in cartesian latent vector z_c are positive, and the L2 norm of the vector is always 1. Also, when the dimension of the input vector is $d_{z_s'}$ the dimension of the output vector d_{z_c} is $d_{z_s}+1$.

In line 3, The discriminator embeds the generated data $G(z_s)$ to the embedded vector z'. The input of the generator G is the spherical latent vector z_s. Since z_s follows a uniform distribution, by adding a quantile function to preprocessing of the generator, the neural network part of the generator can be trained with any desired distribution such as $U(-1,1)$ or $N(0,1)$.

In lines 4 and 5, $\|D(x)\|_2$ and $\|z'\|_2$ represents the magnitude (L2 norm) of embedded vectors of real data and generated data, respectively. v_g and v_r are adversarial values for generated data and real data, respectively. Since the magnitude of the vector is always positive, log is added so that the adversarial value can have a negative value.

In line 6, z_c' is the predicted cartesian latent vector by the discriminator. As previously defined, all elements of a cartesian latent vector z_c are positive. Therefore, the predict cartesian latent vector z_c' must also have all elements positive. I used the $softplus$ function ($y=\log(1+e^x)$) for discriminator output activation so that discriminator always outputs positive vectors.

In lines 7, "$\cdot$", $arccos$, and tan represents an inner product, arccos function, and tangent function, respectively. Since the L2 norm of both cartesian latent vector z_c and predicted cartesian latent vector z_c' are 1, the dot product of the two vectors is cosine similarity. Therefore, embedding loss L_{emb} is the tangent of the angle between the two vectors. Embedding loss L_{emb} makes predicted cartesian latent vector z_c' in the same direction as cartesian latent vector z_c. The following figure shows the visualization of embedding loss L_{emb}.

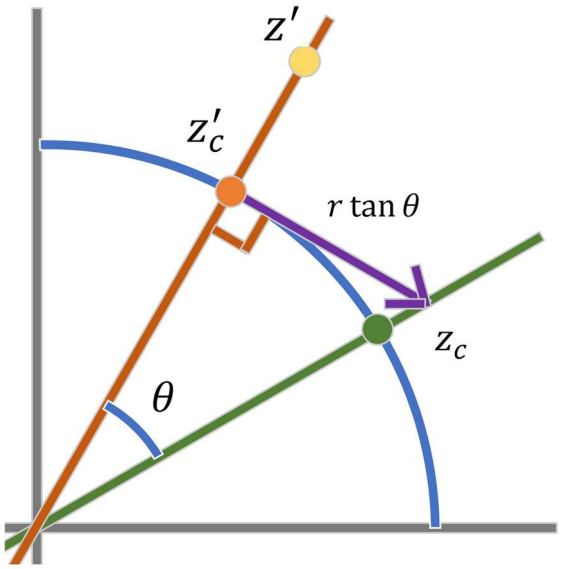

Figure 1. Embedding loss visualization.

In the figure above, the purple arrow represents embedding loss L_{emb}. Embedding loss L_{emb} makes predicted cartesian latent vector z_c' to move in the same direction as cartesian latent vector z_c. θ is the angle between two vectors z_c' and z_c. Since the radius is assumed to be 1, r in the figure is 1. Also, since all elements of cartesian latent vector z_c and predicted cartesian latent vector

z_c' are always positive, L_{emb} is always greater than or equal to zero.

In lines 8 and 9, f_r^d, f_g^d, and f_g are adversarial loss functions. There are several adversarial loss functions such as Original GAN [2], NSGAN [3], LSGAN [4], WGAN-gp [5], etc. Note that the discriminator is trained to lower v_r and raise v_g, and the generator is trained to lower v_g. This is because the latent vector of real data is unknown, so if real data is mapped to a vector with a very large magnitude in the wrong direction, it is difficult to move it to the correct direction. Also, if the magnitude of the embedded vector for the generated data is too low, the direction changes too quickly, make training difficult. L_d and L_g represent discriminator loss and generator loss, respectively. λ_{emb} is the embedding loss weight. One can see that the embedding loss L_{emb} is applied to both the generator and the discriminator.

The direction embedding discriminator GAN is trained to minimize the losses L_d and L_g. This algorithm allows the discriminator to learn the inversion mapping of the generator.

When performing inversion, one can simply obtain a spherical latent vector from the input data through the $cartesian\ to\ spherical$ function, which is an inverse function of $spherical\ to\ cartesian$. Additionally, since the $spherical\ to\ cartesian$ function assumes the radius is 1, the input vector of the $cartesian\ to\ spherical$ function should be normalized. Thus, the spherical latent vector of the input data x is $cartesian\ to\ spherical\left(\frac{D(x)}{\|D(x)\|_2}\right)$.

3. Coordinate transform

There are several ways to convert spherical coordinates to cartesian coordinates. However, when transforming spherical coordinates to cartesian coordinates through sequential transformation [6], underflow inevitably occurs because the latent vector dimension is very big.

The following equations show the sequential transformation of spherical coordinates into cartesian coordinates.

$$x_1 = rcos(\varphi_1)$$
$$x_2 = rsin(\varphi_1)cos(\varphi_2)$$
$$x_3 = rsin(\varphi_1)sin(\varphi_2)cos(\varphi_3)$$
$$\dots$$
$$x_{n-1} = rsin(\varphi_1)\dots sin(\varphi_{n-2})cos(\varphi_{n-1})$$
$$x_n = rsin(\varphi_1)\dots sin(\varphi_{n-2})sin(\varphi_{n-1})$$

x_1 to x_n represent cartesian coordinates. r is the radius of the spherical coordinates, and φ_1 to φ_{n-1} are the angular coordinates of the spherical coordinates.

When transforming cartesian coordinates to spherical coordinates through sequential transformation, underflow occurs at x_n with big n because it has many sin and cos multiplication—also, the error caused by continuous functions increases.

Instead, I recommend converting the spherical coordinates into cartesian coordinates with a perfect binary tree [7]. For example, the transformation from an 8-dimensional spherical coordinate to cartesian coordinates through a

perfect binary tree is as follows.

$$x_{000} = r\sin(\varphi)\sin(\varphi_0)\sin(\varphi_{00})$$
$$x_{001} = r\sin(\varphi)\sin(\varphi_0)\cos(\varphi_{00})$$
$$x_{010} = r\sin(\varphi)\cos(\varphi_0)\sin(\varphi_{01})$$
$$x_{011} = r\sin(\varphi)\cos(\varphi_0)\cos(\varphi_{01})$$
$$x_{100} = r\cos(\varphi)\sin(\varphi_1)\sin(\varphi_{10})$$
$$x_{101} = r\cos(\varphi)\sin(\varphi_1)\cos(\varphi_{10})$$
$$x_{110} = r\cos(\varphi)\cos(\varphi_1)\sin(\varphi_{11})$$
$$x_{111} = r\cos(\varphi)\cos(\varphi_1)\cos(\varphi_{11})$$

Transformation in this paper always assumes a radius of 1, so r is omitted in practice. With the sequential method, real numbers less than or equal to 1 are multiplied $n-1$ times in x_n, but when using a perfect binary tree method, real numbers less than or equal to 1 are multiplied only $\log_2 n$ times. Therefore, underflow hardly occurs. In the above transformation, when the range of φ is set to $\left(0, \frac{\pi}{2}\right)$, all x are always positive.

Transformation of cartesian coordinates to spherical coordinates can be obtained through the proper expansion of equations. The following function shows a practical cartesian to spherical function.

$function\ cartesian\ to\ spherical(z_c)$:

$z_s \leftarrow [\,]$

$t \leftarrow z_c$

$repeat\ \log_2(d_{z_c})\ times$:

$a \leftarrow t[even\ indexes]$

$b \leftarrow t[odd\ indexes]$

$z_s.insert\ front\left(arctan\left(\frac{a}{b}\right)\right)$

$t \leftarrow \sqrt{a \odot a + b \odot b}$

$z_s \leftarrow z_s.flatten()$

$return\ z_s$

Algorithm 3. Cartesian to spherical function

'[ ]' is an empty list, and '$\odot$' is element-wise multiplication. $flatten()$ is function that reshaping list to vector.

In the above algorithm, it is assumed that all elements of z_c are positive values.

5. Experimental results and discussion

5.1 Experiment settings

I trained the direction embedding discriminator GAN to generate the celeb A dataset [8] resized to 128x128 resolution. As the model architecture, a partially reduced version of styleGAN2 [9] was used. For the preprocessing of the generator, a quantile function of normal distribution was used so that the neural network part of the generator receives $N(0,1)$ as input. Batch operations (minibatch stddev layer) of StyleGAN2 are removed so that one data is embedded as one latent vector. As an adversarial loss, NSGAN [3] with r1 loss was used. The model was initially trained with an embedding loss weight $\lambda_{emb} = 1.0$, and then trained with $\lambda_{emb} = 10.0$ after some training. The FID [10] of the model was about 25.

5.2 Experiment results

Figure 2. Generated images

The above figure shows the data generated by the generator of the trained GAN.

Figure 3. Reconstructed images with generated images

The above figure shows the reconstructed images from generated images. In each column separated by a thick white vertical line, the left

images show the generated images, and the right images show the reconstructed images. One can see that the generated images are almost completely reconstructed.

Figure 4. Reconstructed images with real (test) images

The above figure shows the reconstructed images from real (test) images. In each column separated by a thick white vertical line, the left images show the real images, and the right images show the reconstructed images. As the FID of the model is about 25, the performance of GAN is relatively poor, so one can see that the real images are not completely reconstructed. However, one can see that many features are preserved even for real images.

6. Conclusion

In this paper, I proposed a method to make the discriminator learn the inversion mapping of the generator. Discriminator and generator are trained in cooperation to decrease angle between cartesian latent vector and predicted cartesian latent vector. In this way, the discriminator can learn the inversion mapping of the generator. Also, the proposed method can be used during GAN training and does not use additional encoder training or reconstruction loss.

7. References

[1] https://arxiv.org/abs/2101.05278

[2] https://arxiv.org/abs/1406.2661

[3] https://arxiv.org/abs/1801.04406v4

[4] https://arxiv.org/abs/1611.04076

[5] https://arxiv.org/abs/1704.00028

[6] https://en.wikipedia.org/wiki/N-sphere

[7] https://aip.scitation.org/doi/10.1063/1.527088

[8] https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[9] https://arxiv.org/abs/1912.04958

[10] https://arxiv.org/abs/1706.08500

Appendix

Big size image of figure 2. Generated images

Big size image of figure 3. Reconstructed images with generated images

Big size image of figure 4. Reconstructed images with test images