Deterministic Degradation Process for Diffusion GAN and Its Inversion

Jeongik Cho
jeongik.jo.01@gmail.com

Abstract

Recently, diffusion models have shown impressive generative performance. However, they have the disadvantage of having a high latent dimension and slow sampling speed. To increase the sampling speed of diffusion models, diffusion GANs have been proposed. But the latent dimension of diffusion GANs using non-deterministic degradation is still high, making it difficult to invert the generative model. In this paper, we introduce an invertible diffusion GAN that uses deterministic degradation. Our proposed method performs inverse diffusion using deterministic degradation without a model, and the generator of the GAN is trained to perform the diffusion process with the latent random variable. The proposed method uses deterministic degradation, so the latent dimension is low enough to be invertible.

1 Introduction

Generative models such as GANs [1] and VAEs [2] are trained to transform an easy latent random variable into a complex data random variable. Generative model inversion is a method of transforming a complex data random variable into an easy latent random variable. It can be used for techniques such as feature embedding and principal component analysis (PCA). One can find many GAN inversion methods and applications in [4].

On the other hand, the diffusion model [3] has recently demonstrated high generative performance. Since the diffusion model is also a generative model, we can consider of inversion of it as well. However, the diffusion model has a critical disadvantage for inversion, as its latent dimension is very high. For a basic diffusion model that uses Gaussian noise on data, the latent dimension is the product of the data dimension and maximum time step. For example, in a diffusion model that generates 32 × 32 resolution grayscale images with 100-time steps, the practical latent dimension would be 32 × 32 × 100 = 102,400. This is much higher than the latent dimension of GANs or VAEs. High dimensional latent random variable not only makes the inversion of the diffusion model difficult, but even if successful, the resulting inversion may not be meaningful.

Some methods [5], [6] have applied GANs to the diffusion model to achieve faster sampling speeds. However, these methods still use non-deterministic degradations such as data-dimensional Gaussian noise, so the practical latent dimension remains very high.
Instead, in this paper, we introduce DDDG (Deterministic Degradation Diffusion GAN), a method using deterministic degradation as the degradations [7] of the diffusion process and injecting noise through a GAN.

In the basic diffusion model, the diffusion process occurs non-deterministically without the model, and the model is trained to perform reverse diffusion. Therefore, during the diffusion process, the data undergoes degradation, and its entropy increases. In contrast, in DDDG, the reverse diffusion process occurs deterministically without the model, and the GAN is trained to perform diffusion with a low-dimensional latent random variable. Therefore, during the reverse diffusion process in DDDG, the data undergoes degradation, and its entropy decreases.

The overall training process of DDDG is similar to DDGAN [5]. The generator is trained to generate data for the previous time step (equal to the next time step in DDDG), while the discriminator is trained to distinguish data at each time step. However, there are some differences between DDGAN and DDDG.

The first difference is that DDGAN uses non-deterministic degradation, while DDDG uses deterministic degradation. Since DDGAN uses both data dimensional non-deterministic degradation and random latent variables, latent dimension is still high \((d_x + d_z) \times t_{\text{max}}\), where \(d_x\), \(d_z\), and \(t_{\text{max}}\) represents data dimension, latent dimension, and maximum time step). In contrast, DDDG uses only \(d_z\)-dimensional latent random variables at each time step, resulting in a very low latent dimension \((d_z \times t_{\text{max}})\).

The second difference is that the discriminator of DDGAN takes a pair of data as input, while DDDG takes only one data as input. Since DDGAN uses non-deterministic degradation, the discriminator needs to determine whether the generator input data and generator output data correspond to each other. On the other hand, since DDDG uses deterministic degradation, it can enforce the generator’s output data to correspond to the input data. DDDG enforces the correspondence between the generator network output data and input data by normalizing the generator’s network output. As a result, the discriminator does not need to check whether the generator input data and output data corresponding to each other. This means that the discriminator only needs to take in a single data point.

Furthermore, DDDG uses deterministic degradation, and the discriminator only takes in one data point, allowing the latent encoder that inverts the generator to be integrated into the discriminator.

2 Deterministic Degradation Process for Diffusion GAN

The Algo. 1 shows the training procedure of DDDG. The basic procedure is similar to that of DDGAN [5]. In Algo. 1, DLSSGAN [8] was used for GAN inversion.

In line 1, \(x_0\) is a constant. Therefore, \(x'_t\) is initialized as a constant.

In line 2, \(t_{\text{max}}\) represents the maximum time step.

In lines 3 and 4, \(Z\) and \(X\) represent a latent random variable and data random variable, respectively. \(Z\) is \(d_z\)-dimensional i.i.d. random variable with mean 0 and variance 1. The \(\text{sample}\) function represents a function that samples values from a random variable. \(z_t\) and \(x\) represent \(t\)-th latent code and data point, respectively.

In line 5, “⋄” represents element-wise multiplication. \(v_t\) and \(s_t\) represent \(t\)-th latent
Algorithm 1 Algorithm to train DDDG one cycle

Require: $D, G, Z, X, v$

1: $x_t' \leftarrow x_0$
2: for $t$ in $0, 1, ..., t_{\text{max}} - 1$:
3:     $z_t \leftarrow \text{sample}(Z)$
4:     $x \leftarrow \text{sample}(X)$
5:     $s_t \leftarrow \sqrt{\frac{d_t v_{t+1}/2}{\|v_{t+1}/2\|_2}}$
6:     $x_{t+1} \leftarrow \text{reduce info}(x, t + 1)$
7:     $x_{t+1}' \leftarrow G(x_t', z_t \circ s, t)$
8:     $a_r, - \leftarrow D(x_{t+1}, t + 1)$
9:     $a_f, z_t' \leftarrow D(x_{t+1}', t + 1)$
10:    $L_{\text{enc}} \leftarrow \frac{1}{d_t} \| (z_t - z_t') \circ s \|_2^2$
11:    $L_d \leftarrow \text{adv}(a_r, a_f) + \lambda_{\text{enc}} L_{\text{enc}}$
12:    $L_g \leftarrow \text{adv}(a_f) + \lambda_{\text{enc}} L_{\text{enc}}$
13:    $x_t' \leftarrow x_{t+1}'$
14:    $v_t \leftarrow \text{update}(v_t, z_t^m 2)$
15:    apply gradient($D, L_d$)
16:    apply gradient($G, L_g$)
variance vector and latent scale vector, respectively. \( v_t \) and \( s_t \) correspond to the latent variance vector, and latent scale vector of DLSGAN [8]. One other difference is that in DDDG, \( v \) is a matrix of shape \( t_{\text{max}} \times d_z \). The latent variance vector of each time step can be different, so \( t_{\text{max}} \) latent variance vectors are required. \( d_z \) represents a dimension of the latent random variable for each time step.

In line 6, \( \text{reduce info} \) represents the degrading process. The entropy of data point \( x \) increases after the degrading process in a typical diffusion model. However, the information of the data point \( x \) decreases after the degrading process in DDDG because DDDG uses deterministic degrading. Therefore, \( \text{reduce info}(X, 0) \) has minimum entropy (i.e., constant), and \( \text{reduce info}(X, t_{\text{max}}) \) equals to \( X \). \( x_t \) refers to the data point \( x \) at the \( t \)-th time step. Therefore, one can infer that \( x_0 \) in line 1 is constant.

In line 7, \( x'_{t+1} \) represents generated data by the generator \( G \). One can see that \( G \) takes previously generated data \( x'_t \) as input and generates \( x'_{t+1} \). In the case where the model converges perfectly, \( x'_t \) in the generator input can be replaced with \( x_t = \text{reduce info}(x, t) \). However, in consideration of the case where the model may not fully converge, we use \( x'_t \) instead. Also, note that the output of generator \( G \) is always be normalized to ensure that \( x'_{t+1} \) corresponds to \( x'_t \) (see Eqs. 1, 2, and 3).

In lines 8 and 9, \( D \) represents the integrated discriminator with the latent encoder. Therefore, integrated \( D \) has 2 outputs. The first output \( a_r \) or \( a_f \) represents adversarial value for real/fake discrimination. The second output \( z'_t \) represents the predicted latent code of latent code \( z_t \). "\( \_ \)" represents not using value. Like DLSGAN, DDDG does not use the predicted latent code of the real data.

In line 10, \( L_{\text{enc}} \) represents encoder loss. \( L_{\text{enc}} \) is the same as the encoder loss of DLSGAN.

In lines 11 and 12, \( \text{adv} \) represents the adversarial loss function for GAN. One can find several adversarial losses in [9]. \( \lambda_{\text{enc}} \) represents encoder loss weight.

In line 13, \( x'_t \) is updated to \( x'_{t+1} \) for the next training step.

In line 14, "\( \text{update} \)" represents the update function of DLSGAN. Note that only \( v_t \) (\( t \)-th vector of \( v \)) is updated.

DDDG uses deterministic degradation. It means that there are constraints on the representation of \( x'_{t+1}, x'_{t+2}, \ldots, x'_{t_{\text{max}}} \) when given \( x'_t \). For example, assume \( 2 \times 2 \) average pooling was used for image DDDG degradation. It means that average pooling on \( x'_{t+1} \) or 2 times average pooling on \( x'_{t+2} \) should be equal to \( x'_t \). To satisfy such constraints, DDDG normalizes the generator network output.

\[
x_{t+1}^{\text{net}} = G_{\text{net}}(x'_t, z_t \odot s, t) \tag{1}
\]
\[
x_{t+1}^{\text{norm}} = x_{t+1}^{\text{net}} - \text{reduce info}(x_{t+1}^{\text{net}}, t) + x'_t \tag{2}
\]
\[
x'_{t+1} = \text{reduce info}(x_{t+1}^{\text{norm}}, t+1) \tag{3}
\]

In Eq. 1, \( G_{\text{net}} \) and \( x_{t+1}^{\text{net}} \) represent the network part of the generator and network output, respectively.

Eq. 2 shows an example of normalizing the generator output. \( x_{t+1}^{\text{norm}} \) is normalized network output. DDDG enforces the correspondence between the generator output and input data by using Eq. 2. The normalization method can vary depending on the degradation method used. However, Eq. 2 can be used for most degradation methods that use typical filters.
Eq. 3 shows applying `reduce info` to $x_{t+1}^{norm}$ so that $x'_{t+1}$ corresponds to time step $t + 1$.

**Algorithm 2** DDDG sampling process

**Require:** $G, z, v$
1: $x' \leftarrow x_0$
2: for $t$ in $0, 1, ..., t_{max} - 1$
3: \hspace{1em} $s_t \leftarrow \sqrt{\frac{t}{t+1}} \cdot \frac{v}{\|v\|_2}$
4: \hspace{1em} $x' \leftarrow G(x', z_t \circ s_t, t)$
5: return $x'$

**Algorithm 3** DDDG inversion process

**Require:** $D, x$
1: $z \leftarrow []$
2: for $t$ in $0, 1, ..., t_{max} - 1$
3: \hspace{1em} $x_{t+1} \leftarrow reduce info(x, t + 1)$
4: \hspace{1em} $z_t \leftarrow D(x_{t+1}, t + 1)$
5: \hspace{1em} $z.append(z_t)$
6: return $z$

Algos. 2, 3 show sampling and inversion algorithms for DDDG. In Algo. 2, $x_0$ is a constant.

In Algo. 3, `[]` represents empty list.

One can find several differences between DDDG and DDGAN [5] in Algo. 1.

The first difference is that DDGAN uses non-deterministic degradation, while DDDG uses deterministic degradation. Since DDGAN uses both data dimensional non-deterministic degradation and random latent variables, it uses a very high latent dimension ($(d_x + d_z) \times t_{max}$, where $d_x$ represents data dimension). Therefore, it is difficult to invert the generator in DDGAN, and even if inverted, it may not be meaningful. In contrast, DDDG uses only $d_z$-dimensional latent random variables at each time step, resulting in a very low latent dimension ($d_z \times t_{max}$). This makes it easy and meaningful to invert the generator.

The second difference is that the discriminator of DDGAN takes a pair of data as input, while DDDG takes only one data as input. Since DDGAN uses non-deterministic degradation, the discriminator needs to determine whether the generator input data and output data corresponding to each other. In contrast, DDDG uses deterministic degradation, so it can enforce the generator’s output data to correspond to the input data. DDDG enforces the correspondence between the generator network output data and input data by normalizing the generator’s network output (Eq. 2).

Furthermore, DDDG’s use of deterministic degradation and discriminator receiving only one data input allows the latent encoder for inverting the generator to be integrated with the discriminator. In the case of DDGAN, since the discriminator takes a pair of data with non-deterministic noise as input, a separate encoder that does not share layers with the discriminator is required to invert the generator.

Due to these differences, DDDG can efficiently and meaningfully invert diffusion GANs. Also, DDDG retains the advantage of fast sampling speed compared to a
general diffusion model like another diffusion GANs.

3 Experiments

We trained DDDG with MNIST dataset [13]. The following hyperparameters were used for model training.

\[
\begin{align*}
\lambda_{r1} &= 0.1 \\
\lambda_{enc} &= 1.0 \\
d_z &= 64 \\
t_{max} &= 6 \\
Z &\sim N(0, I_{d_z}) \\
optimizer &= \text{Adam} \left( \begin{array}{c} \\
\text{learning rate} = 0.002 \\
\beta_1 = 0.0 \\
\beta_2 = 0.99 \\
\end{array} \right) \\
\text{trainable weights ema decay rate} &= 0.999 \\
\text{latent variance vector ema decay rate} &= 0.999 \\
\text{batch size} &= 16 \\
\text{epochs} &= 100
\end{align*}
\]

NSGAN [1] with R1 regularization [10] were used for adversarial losses. \( \lambda_{r1} \) represents R1 regularization weight. Equalized learning rate [11] was used for all trainable weights. DDDG is applicable for class-conditional GANs as the number of time steps is very low. Therefore, we used CAGAN [14] for class-conditional GAN loss. We also utilized the idea of CAGAN for the latent encoder of the discriminator. We set the latent encoder output to \( d_z \times t_{max} \)-dimensional and only activate \( d_z \)-dimensional latent encoder output corresponding to each time step.

We used Gaussian blur for the degradation process. Fig. 1 shows the degradation process of real images with Gaussian blur.

After training, the FID score [12] of DDDG was 4.196431, and the real image reconstruction performance was a PSNR of 14.816616 and an SSIM of 0.468145.

Fig. 2 visualizes the sampling process of DDDG. One can observe that the data is progressively generated through the diffusion process in Fig. 2.

Fig. 3 visualizes the reconstruction process of DDDG. \( d_z = 64 \) and \( t_{max} = 6 \), so practical latent dimension of each sample is \( 64 \times 6 = 384 \). It means that the latent encoder of DDDG encodes input data into a 384-dimensional latent code.

In DDDG, the error in the predicted latent code at low \( t \) continues to have a persistent impact on the subsequent time steps. We can see from Fig. 3 that some samples are incorrectly reconstructed due to the poorly predicted latent code when the time step is low. However, it still demonstrates that DDDG can properly reconstruct many images through generative model inversion.

4 Conclusion

In this paper, we propose a diffusion GAN that uses deterministic degradation. Our proposed method performs inverse diffusion using deterministic degradation without a
Figure 1: Degradation process with Gaussian blur in real images. First column: constant (reduce info($x, 0$)). Second column: blurred image with Gaussian kernel $\sigma = 6$ (reduce info($x, 1$)). Third to sixth column: $\sigma = 5, 4, 3, 2$. Last column: original image (reduce info($x, t_{\text{max}}$)).

model, and the generator of the GAN is trained to perform the diffusion process with the latent random variable. The proposed method utilizes deterministic degradation, allowing for a very low latent dimension, making it possible to invert the generator. In addition, by using generator output normalization, the proposed method enforces the generator to produce data that corresponds to the input data. As a result, the discriminator does not need to take in a pair of data. In addition, because DDDG uses deterministic degradation and the discriminator only needs to take in one data point, the latent encoder and discriminator can be integrated efficiently.

References


Figure 2: Visualization of DDDG sampling process. Samples in top part are $x'_0, x'_1, ..., x'_{t_{\text{max}}-1}$. Samples in bottom part are $x_{\text{norm}}^1, x_{\text{norm}}^2, ..., x_{\text{norm}}^{t_{\text{max}}}$. Final output $x'_{t_{\text{max}}} = \text{reduce info}(x_{\text{norm}}^{t_{\text{max}}}, t_{\text{max}}) = x_{\text{norm}}^{t_{\text{max}}}$.
Figure 3: Visualization of DDDG reconstruction process. Images in the last column are inputs. The left part shows the progressive reconstruction using the latent code obtained through the encoder.


