GANs and Diffusion Models

Generative models

thispersondoesnotexist.com

thiscitydoesnotexist.com

thischairdoesnotexist.com

thiscatdoesnotexist.com

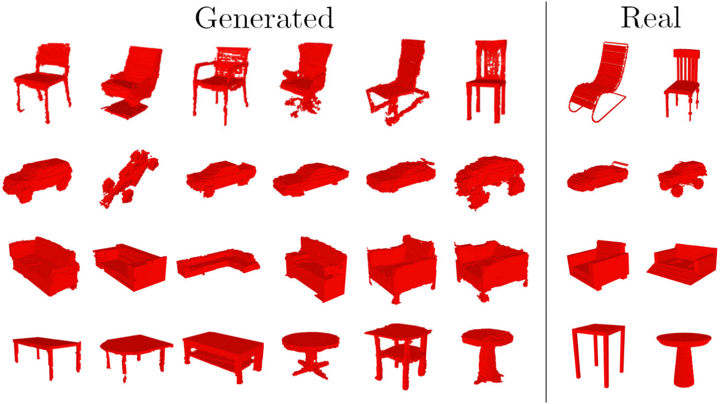
thisstartupdoesnotexist.com

Why Generative Models?

https://arxiv.org/abs/1609.04802

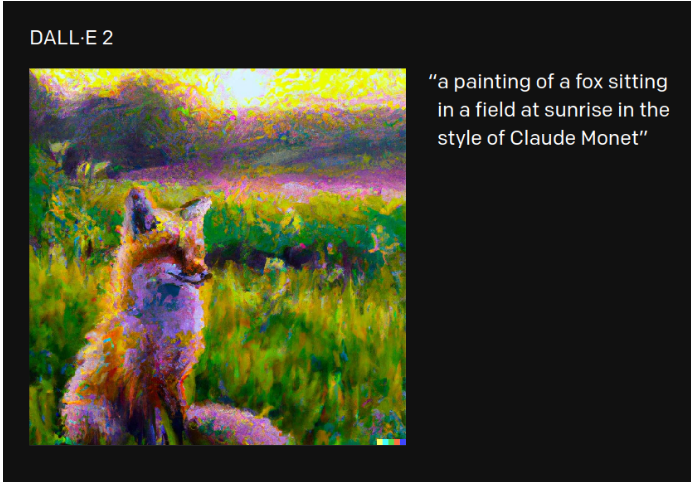
Image super-resolution

Why Generative Models?



Generative Design

Why Generative Models?



Text-to-Image generation

Generative Modeling

Assume that a dataset $\mathcal{D}$ is generated from a **probability distribution** p

Generative Models estimate p from the dataset $\mathcal{D}$

Generative Modeling

Assume that a dataset $\mathcal{D}$ is generated from a **probability distribution** p

Generative Models estimate p from the dataset $\mathcal{D}$

- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ supervised generative models learn the joint distribution $p(x, y)$, often to compute $p(y \mid x)$

Generative Modeling

Assume that a dataset $\mathcal{D}$ is generated from a **probability distribution** p

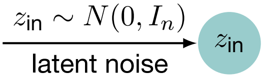
Generative Models estimate p from the dataset $\mathcal{D}$

- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ supervised generative models learn the joint distribution $p(x, y)$, often to compute $p(y \mid x)$
- $\mathcal{D} = \{x^{(1)}, \ldots, x^{(n)}\}$ unsupervised generative models learn the distribution $p(x)$, often to generate a new sample $x \sim p(x)$

Generative Adversarial Networks

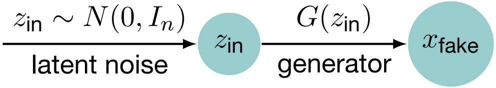
GANs

Step 1: Sample a noise vector z_{in} from the standard normal distribution



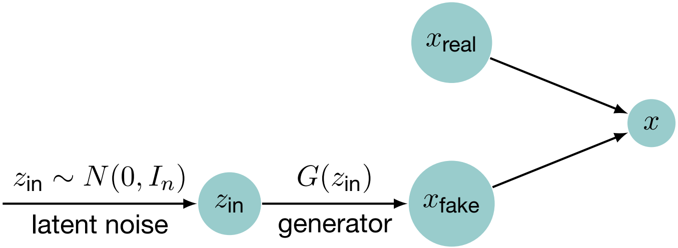
GANs

Step 2: Use a **Generator** to transform z_{in} to a fake image



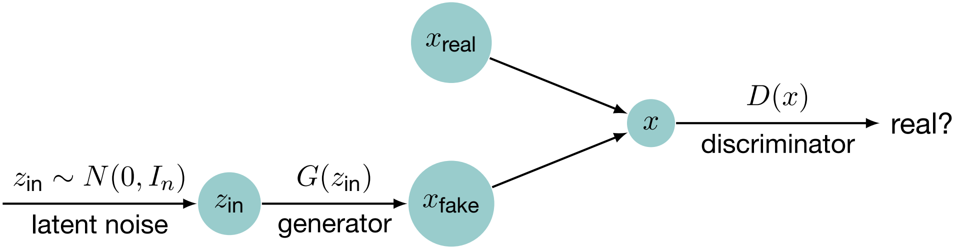
GANs

Step 3: Mix fake images and real images together

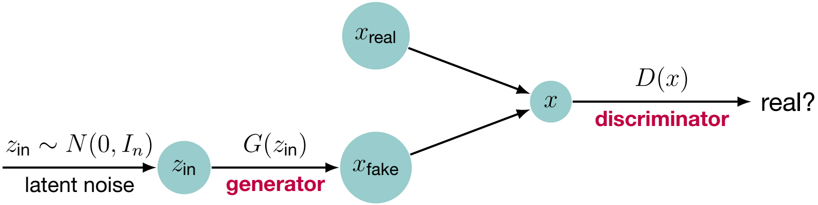


GANs

Step 4: Use a **Discriminator** to classify between real and fake images



Two models in GANs



- Two models compete against each other:
- **Generator** tries to fool the discriminator by making realistic fake images
- **Discriminator** tries to distinguish between real and fake images
- This feedback loop results in **fake images that are similar to real images**

GANs Loss

To train both models, we need loss functions

Let $\mathrm{BCE} =$ Binary Cross-Entropy Loss

Discriminator (D) First we obtain a pair of $x =$ real image and $G(z) =$ fake image

GANs Loss

To train both models, we need loss functions

Let $\mathrm{BCE}=$ Binary Cross-Entropy Loss

Discriminator (D) First we obtain a pair of $x=$ real image and $G(z)=$ fake image

- The label of x is 1 (real) and the label of $G(z)$ is 0 (fake)
- The prediction of D on x is $D(x)$, and that on $G(z)$ is $D(G(z))$

GANs Loss

To train both models, we need loss functions

Let $\mathrm{BCE} =$ Binary Cross-Entropy Loss

Discriminator (D) First we obtain a pair of $x =$ real image and $G(z) =$ fake image

- The label of x is 1 (real) and the label of $G(z)$ is 0 (fake)
- The prediction of D on x is $D(x)$, and that on $G(z)$ is $D(G(z))$
- Thus the loss of the discriminator D is:

$$\mathrm{BCE}(D(x), 1) + \mathrm{BCE}(D(G(z)), 0)$$

GANs Loss

To train both models, we need loss functions

Let $\mathrm{BCE} =$ Binary Cross-Entropy Loss

Generator (G) Consider $G(z) =$ fake image

- Label $G(z)$ as 1 if the discriminator classified it as a real image
- In other words, the generator wants is $D(G(z)) = 1$

GANs Loss

To train both models, we need loss functions

Let $\mathrm{BCE} =$ Binary Cross-Entropy Loss

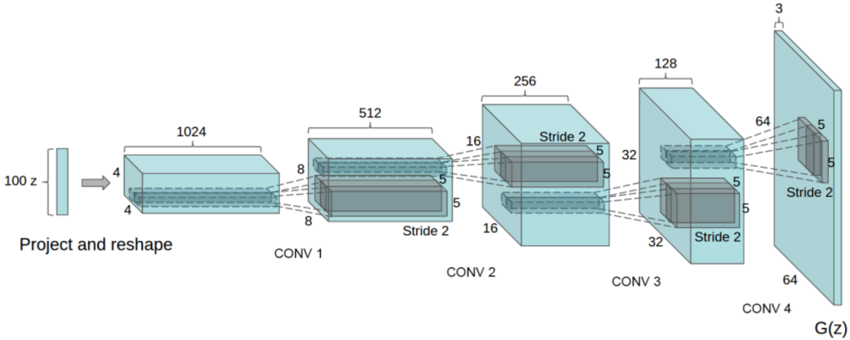
Generator (G) Consider $G(z) =$ fake image

- Label $G(z)$ as 1 if the discriminator classified it as a real image
- In other words, the generator wants is $D(G(z)) = 1$
- Thus the loss of the generator G is:

$$\mathrm{BCE}(D(G(z)), 1)$$

DCGAN

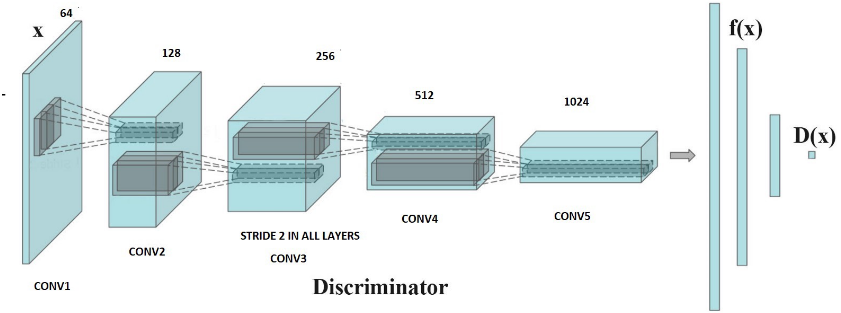
Deep Convolutional GAN



Generator

DCGAN

Deep Convolutional GAN



Diffusion Models

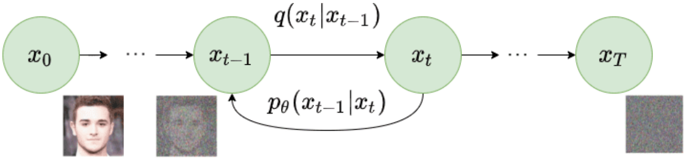
Text-to-Image Demos

- DALL·E 2: `https://openai.com/dall-e-2`
- Stable Diffusion: `https://huggingface.co/spaces/stabilityai/stable-diffusion`

https://dreamingcomputers.com/ai-images/stable-diffusion-ai-art

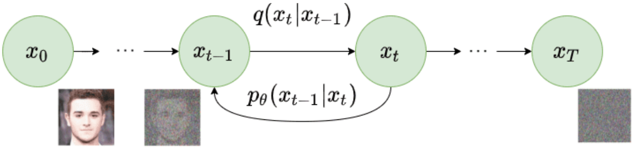
"A fine detail concept art of a one steampunk narwhal, by tyler edlin trending on artstation hd, glowing colorful intricate wires"

Diffusion Models



- **Forward diffusion:** Iteratively add noises to the image
- **Backward diffusion:** Revert the process, transforming noises into an image

Forward diffusion

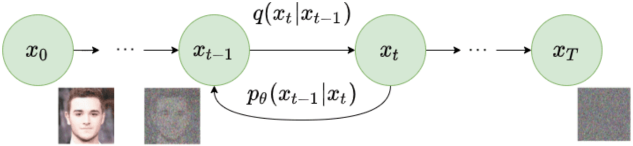


1. Choose **Diffusion Parameters** $\beta_1, \beta_2, \ldots, \beta_T$

2. At step $t = 1, \ldots, T$, add noises to image:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\varepsilon_{t-1}, \quad \varepsilon_{t-1} \sim \mathcal{N}(0, I_n)$$

Forward diffusion



1. Choose **Diffusion Parameters** $\beta_1, \beta_2, \ldots, \beta_T$

2. At step $t = 1, \ldots, T$, add noises to image:

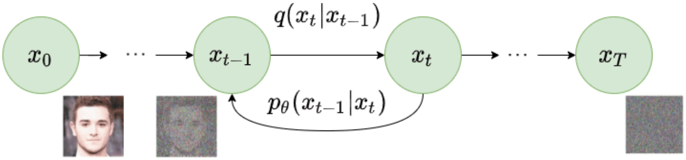
$$x_t = \sqrt{1-\beta_t} x_{t-1} + \sqrt{\beta_t} \varepsilon_{t-1}, \quad \varepsilon_{t-1} \sim \mathcal{N}(0, I_n)$$

For all t, x_t can be written in terms of x_0:

$$x_t = \sqrt{\bar{\alpha}_t} x_{t-1} + \sqrt{1-\bar{\alpha}} \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I_n),$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \alpha_1 \alpha_2 \ldots \alpha_t$

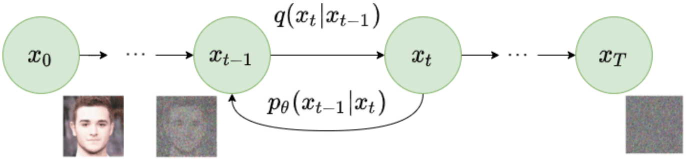
Backward diffusion



Learn the probability distribution(s)

$$p(x_0, \ldots, x_T) = p(x_0 \mid x_1) \times \ldots \times p(x_{T-1} \mid x_T) \times p(x_T)$$

Backward diffusion



Learn the probability distribution(s)

$$p(x_0, \ldots, x_T) = p(x_0 \mid x_1) \times \ldots \times p(x_{T-1} \mid x_T) \times p(x_T)$$

To generate a new image:

1. Sample a noise image $x_T \sim \mathcal{N}(0, I)$
2. For $t = T, \ldots, 1$, generate $x_{t-1} \sim p(x_{t-1} \mid x_t)$

Backward diffusion

Assume that $p(x_{t-1} \mid x_t)$ is a **normal distribution**

$$p(x_{t-1} \mid x_t) = \mathcal{N}(\mu_t(x_t), \beta_t I)$$

We want to **learn** the distribution, which is the same as learning the parameter μ_t

Backward diffusion

Assume that $p(x_{t-1} \mid x_t)$ is a **normal distribution**

$$p(x_{t-1} \mid x_t) = \mathcal{N}(\mu_t(x_t), \beta_t I)$$

We want to **learn** the distribution, which is the same as learning the parameter μ_t

A common technique to learn μ_t is to use a neural network:

- x_t is the features
- μ_t is the target

But we don't know μ_t...

Backward diffusion

Fortunately, we can write μ_t in terms of ε_t, which is the noise that we sampled during the forward diffusion!

Backward diffusion

Fortunately, we can write μ_t in terms of ε_t, which is the noise that we sampled during the forward diffusion! Thanks to Bayes's rule:

$$p(x_{t-1} \mid x_t) = q(x_t \mid x_{t-1}) \times \ldots$$

and good properties of normal distributions, one can derive that

$$\mu_t(x_t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon_t \right),$$

Backward diffusion

$$p(x_{t-1} \mid x_t) = \mathcal{N}(\mu_t(x_t), \beta_t I)$$

We want to **learn** $\mu_t(x_t)$, and we have

$$\mu_t(x_t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon_t \right),$$

Idea: use a neural network to <u>learn the noise</u> instead

$$\varepsilon_t = \varepsilon_t(x_t)$$

- x_t is the features
- ε_t is the target

Training the model

1. Initialize T neural networks: $\mathrm{NN}_1, \ldots, \mathrm{NN}_T$
2. For many epochs
   2.1 Randomly choose t from $\{1, \ldots, T\}$
   2.2 Sample a noise $\varepsilon_t \sim N(0, I)$
   2.3 Train NN_t with data: $(x_t, \varepsilon_t) = (\sqrt{\bar{\alpha}_t} x_{t-1} + \sqrt{1 - \bar{\alpha}} \varepsilon_t, \varepsilon_t)$

Sampling a new image

1. Sample $x_T \sim N(0, I)$
2. For $t = T, \ldots, 1$ do
   - 2.1 $\varepsilon_t = \mathrm{NN}_t(x_t)$
   - 2.2 $\mu_t = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_t \right)$
   - 2.3 Sample $x_{t-1} \sim \mathcal{N}(\mu_t, \beta_t I)$

Diffusion model for Text-to-Image

- DALL·E 2: `https://openai.com/dall-e-2`
- Stable Diffusion: `https://huggingface.co/spaces/stabilityai/stable-diffusion`