10-414/714 – Deep Learning Systems: Algorithms and Implementation

Generative Models

Fall 2023

J. Zico Kolter and Tianqi Chen (this time)
Carnegie Mellon University

Outline

Generative adversarial training (GAN)

Diffusion models

Outline

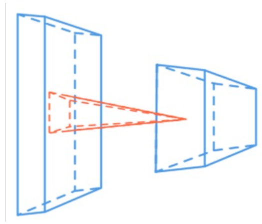
Generative adversarial training

Diffusion models

From classifier to generator

Class probability predictor

x



$h_\theta(x)$

Prediction

$\hat{y}$

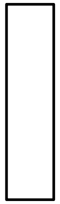
Label

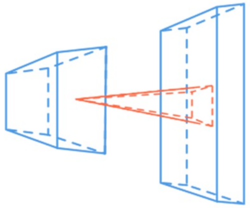
$y = 6$

Goal: predicted class get close to the label

Digits generator

Random vector





Run multiple times

Generated samples

Target data distribution

Goal: make generated the distribution of generated samples “close” to target data distribution

Define “distance” of distributions

Unlike supervised classification setting, the “goal” is less obvious

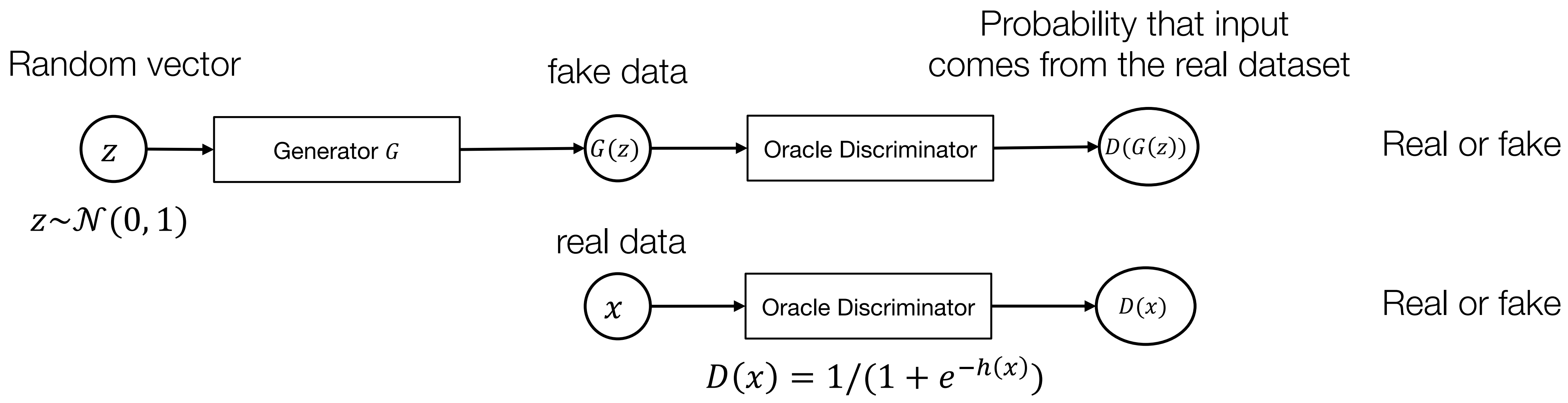
To build effective training mechanism, we need to define a “distance” between generated and real datasets and use that to drive the training.

What we really wanted, in text: make sure that the generated samples “looks real”.

Generated samples

Target data distribution

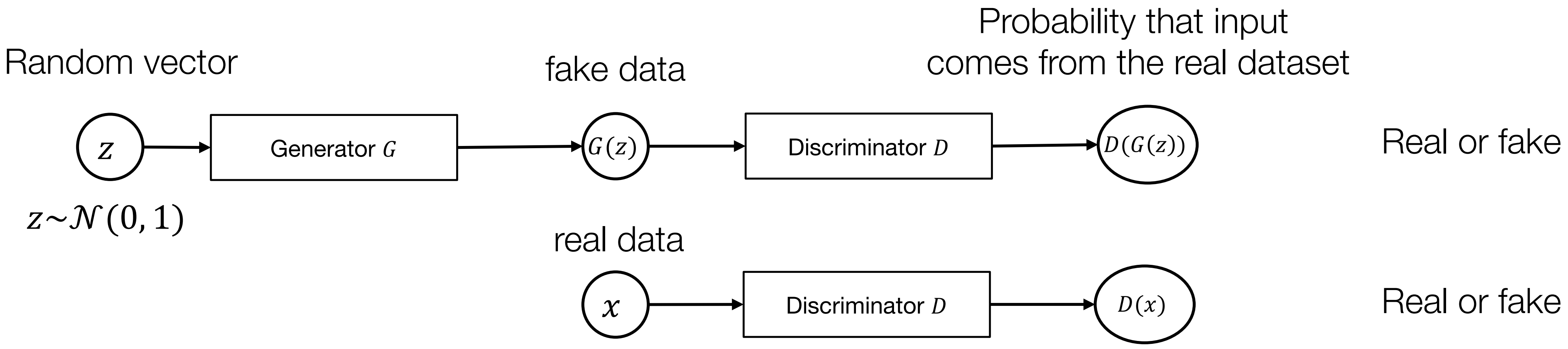
Learn generator through an oracle discriminator



Assume that we have an oracle discriminator that can tell the difference between real and fake data. Then we need train the generator to "fool" the oracle discriminator.
We need to maximize the discriminator loss

Generator objective: $max_G\{-\mathrm{E}_{z \sim Noise} \log(1 - D\big(G(z)\big)\}$

Learning the discriminator

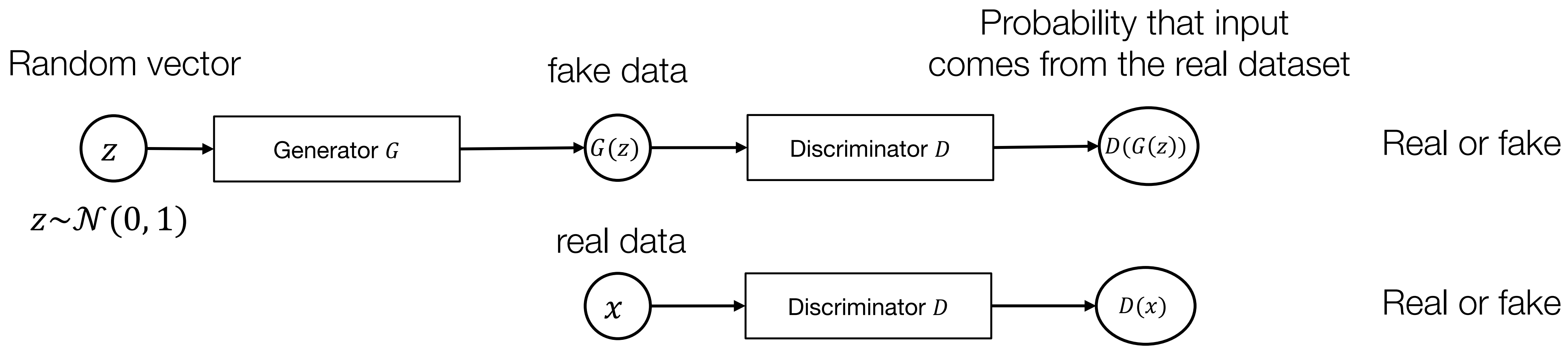


$$D(x) = 1/(1 + e^{-h(x)})$$

We do not have an oracle discriminator, but we can learn it using the real and generated fake data.

Discriminator objective $min_D\{-E_{x\sim Data}\log D(x) - \mathrm{E}_{z\sim Noise}\log(1 - D(G(z))\}$

Generative adversarial network



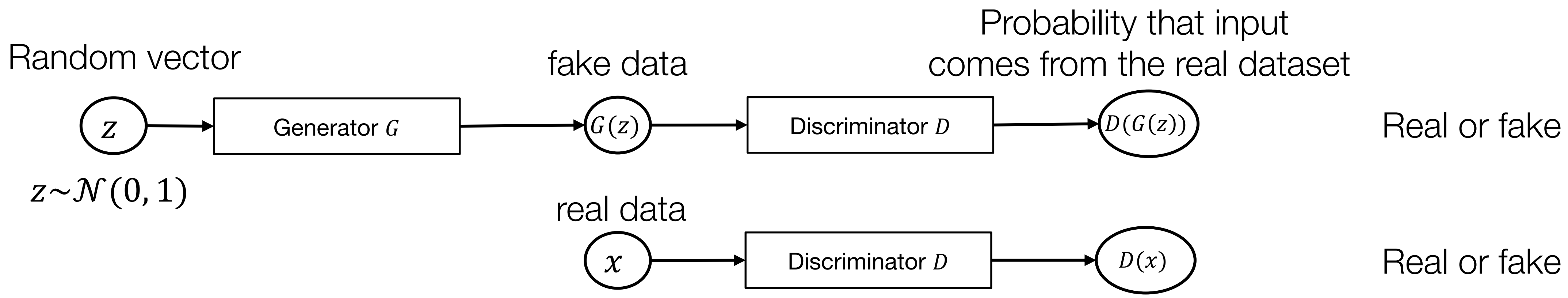
$$D(x) = 1/(1 + e^{-h(x)})$$

Putting it together, it becomes an “minimax” game between D and G

$$min_D max_G \{-E_{x\sim Data} \log D(x) - \mathrm{E}_{z\sim Noise} \log(1 - D(G(z))\}$$

In practice, we usually optimize G using $min_G\{-\mathrm{E}_{z\sim Noise} \log(D(G(z))\}$, maximize the probability that discriminator predicts generated image is real

Generative adversarial training in practice



$$D(x) = 1/(1 + e^{-h(x)})$$

Iterative process

- Discriminator update
  - Sample minibatch of $D\big(G(z)\big)$, get a minibatch of $D(x)$
  - Update D to minimize $min_D\{-E_{x \sim Data} \log D(x) - \mathrm{E}_{z \sim Noise} \log(1 - D\big(G(z)\big)\}$
- Generator update
  - Sample minibatch of $D\big(G(z)\big)$
  - Update G to minimize $min_G\{-\mathrm{E}_{z \sim Noise} \log\big(D\big(G(z)\big)\big)\}$, this can be done by feeding label=1 to to the model

Outline

Generative adversarial training

Diffusion models

Stochastic Differential Equation

Describes a stochastic movement of x_t in the space

Wiener process (gaussian white noise, see the discrete view)

Stochastic differential equation (SDE): $$dx_t = f(x_t)\,dt + \sqrt{2D(x_t,t)}\,dW_t$$

Drift term, pulls towards modes

Diffusion term, inject noise

Discrete simulation of the process, with small step size η_t

$$x_{t+1} \leftarrow x_t + \eta_t f(x_t) + \mathcal{N}(0, 2\eta_t D(x_t,t))$$

Forward Diffusion Process

Data distribution

$q(x_0)$

Close to standard normal distribution

$q(x_T)$

Gradually add gaussian noise

Forward SDE: $$dx_t = -\frac{1}{2}\beta(t)x_t\,dt + \sqrt{\beta(t)}\,dW_t$$

Discretized form: $$x_{t+1} \leftarrow (1 - \frac{1}{2}\eta_t\beta(t))x_t + \mathcal{N}(0, \eta_t\beta(t))$$

Drifts toward 0

Add noise

Forward diffusion process takes image x_0 and generate white noise x_T

Image source: https://cvpr2022-tutorial-diffusion-models.github.io/

Reverse (Denoising) Process

$q(x_0)$

$q(x_T)$

Forward SDE: $dx_t = -\frac{1}{2}\beta(t)x_t\, dt + \sqrt{\beta(t)}\, dW_t$

It is easy to sample from $q(x_0, \dots, x_t \dots x_T | x_0)$

The reverse process $x_T, \dots, x_t, \dots x_0$ can described by the following SDE

Reverse SDE: $dx_t = -[\frac{1}{2}\beta(t)x_t - \beta(t)\nabla_{x_t} \log q(x_t)]\, dt + \sqrt{\beta(t)}\, dW_t$

Score function

If we simulate this reverse process, we can generate data distribution from noise x_T

Image source: https://cvpr2022-tutorial-diffusion-models.github.io/

Score Matching the Reverse Process

$q(x_0)$

$q(x_T)$

Forward SDE: $dx_t = -\frac{1}{2}\beta(t)x_t\, dt + \sqrt{\beta(t)}\, dW_t$

Reverse SDE: $dx_t = -[\frac{1}{2}\beta(t)x_t \; - \beta(t)\boxed{\nabla_{x_t} \log q(x_t)}]\, dt + \sqrt{\beta(t)}\, dW_t$

Approximate with
neural network $s_\theta(x_t, t)$

$\min_\theta E_{t\sim U(0,T),\, x_t \sim\, q(x_t)}\; ||s_\theta(x_t, t) \text{ - } \nabla_{x_t} \log q(x_t)\, ||^2$

Issue: we don't have close-form formula for $\nabla_{x_t} \log q(x_t)$

Image source: https://cvpr2022-tutorial-diffusion-models.github.io/

Score Matching the Reverse Process

$q(x_0)$  $q(x_T)$

Forward SDE: $dx_t = -\frac{1}{2}\beta(t) x_t \, dt + \sqrt{\beta(t)} \, dW_t$ $\xrightarrow{\text{can derive}}$ $\boxed{\begin{array}{l} q(x_t \mid x_0) = \mathcal{N}(\gamma_t x_0, \sigma_t^2 \mathrm{I}) \\ \gamma_t = e^{-\frac{1}{2}\int_0^t \beta(s) ds}, \sigma_t^2 = 1 - e^{-\int_0^t \beta(s) ds} \end{array}}$

Modified matching objective: $\min_\theta E_{t \sim U(0,T),\, x_0 \sim q(x_0),\, x_t \sim q(x_t|x_0)} \, ||s_\theta(x_t, t) - \nabla_{x_t} \log q(x_t \mid x_0)||^2$

Sample x_t: $x_t = \gamma_t x_0 + \sigma_t \epsilon \quad \epsilon \sim \mathcal{N}(0, \mathrm{I})$

Simplify the score: $\nabla_{x_t} \log q(x_t \mid x_0) = -\nabla_{x_t} \frac{(x_t - \gamma_t x_0)^2}{2\sigma_t^2} = -\frac{x_t - \gamma_t x_0}{\sigma_t^2} = -\frac{\epsilon}{\sigma_t}$

Parameterize: $s_\theta(x_t, t) = -\frac{\epsilon_\theta(x_t, t)}{\sigma_t}$ $\Rightarrow$ $\min_\theta E_{t \sim U(0,T), x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,\mathrm{I})} \frac{1}{\sigma_t^2} ||\epsilon_\theta(x_t, t) - \epsilon \,||^2$

Image source: https://cvpr2022-tutorial-diffusion-models.github.io/

Training Diffusion Model

$q(x_0)$  $q(x_T)$

$$q(x_t \mid x_0) = \mathcal{N}(\gamma_t x_0, \sigma_t^2 \mathrm{I})$$
$$\gamma_t = e^{-\frac{1}{2}\int_0^t \beta(s)ds}, \sigma_t^2 = 1 - e^{-\int_0^t \beta(s)ds}$$

Forward SDE: $dx_t = -\frac{1}{2}\beta(t)x_t\, dt + \sqrt{\beta(t)}\, dW_t$

Sample $t \sim U(0, T)$

$x_t = \gamma_t x_0 + \sigma_t\, \epsilon \quad \epsilon \sim \mathcal{N}(0, \mathrm{I})$

Time step weighting, usually set $\lambda_t = \sigma_t^2$

Update $\epsilon_\theta(x_t, t)$ to minimize $\frac{\lambda_t}{\sigma_t^2} ||\epsilon_\theta(x_t, t) - \epsilon\, ||^2$

Intuition: we are predicting the noise ϵ !

Image source: https://cvpr2022-tutorial-diffusion-models.github.io/

Generation Process in Diffusion Model

$q(x_0)$  $q(x_T)$

Forward SDE: $dx_t = -\frac{1}{2}\beta(t)x_t\, dt + \sqrt{\beta(t)}\, dW_t$

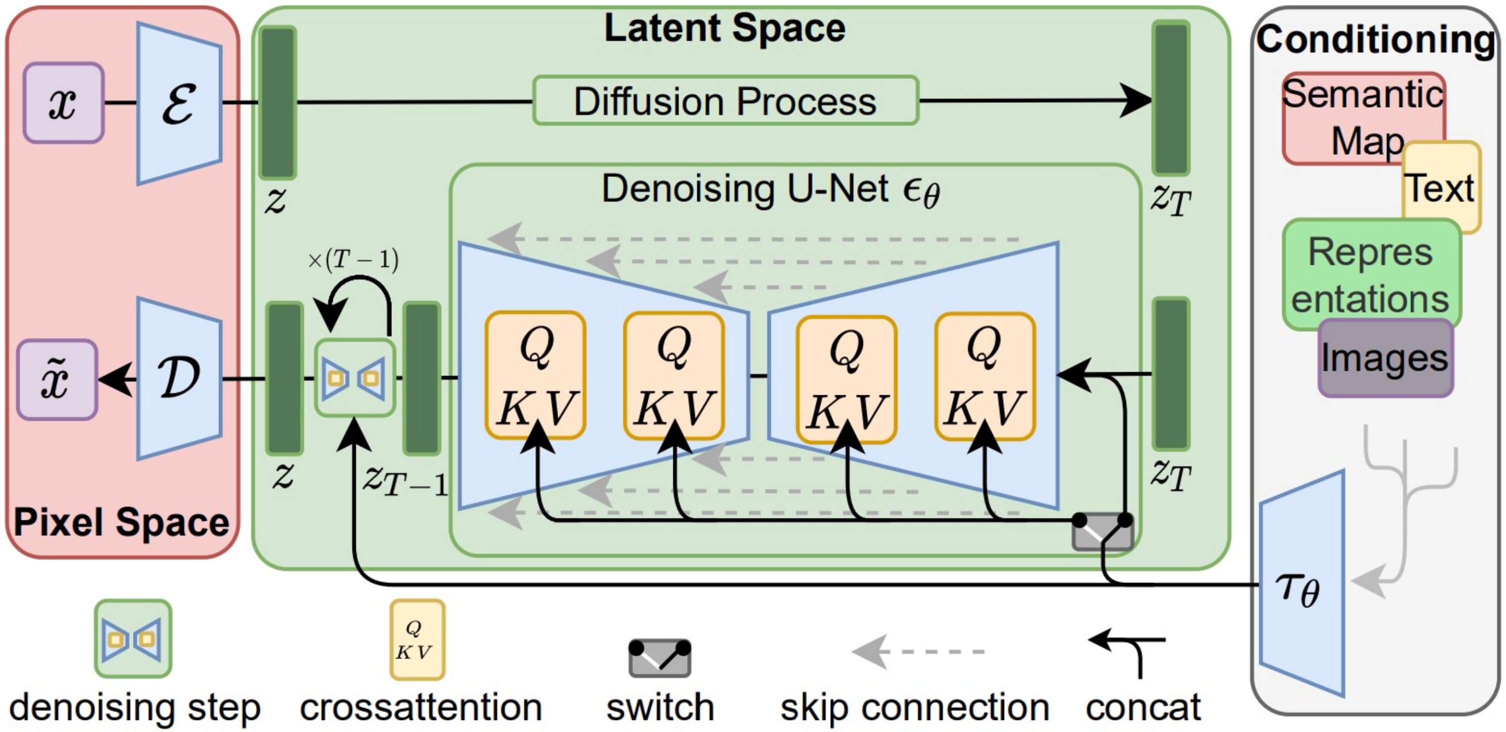
Generation process: $dx_t = -[\frac{1}{2}\beta(t)x_t - \beta(t)\epsilon_\theta(x_t, t)]\, dt + \sqrt{\beta(t)}\, dW_t$

Intuition take a small step in reverse direction of predicted noise

Different ways to numerically run the generation process

Image source: https://cvpr2022-tutorial-diffusion-models.github.io/

Latent Space Diffusion Models

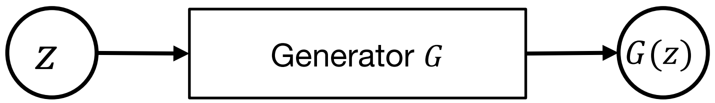


Running diffusion process in latent space
Decode back to higher resolution pixel space

Rombach et al., “High-Resolution Image Synthesis with Latent Diffusion Models”, CVPR 2022

Comparing GAN and Diffusion Models

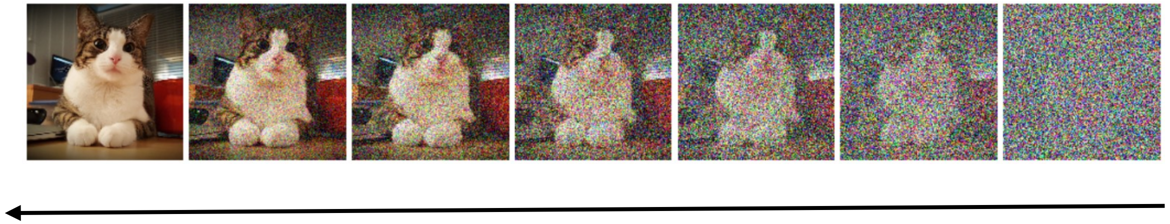
GAN



$z \sim \mathcal{N}(0, 1)$

Generate output by single step G

Diffusion Models



Learning iterative refinement instead of single step generation

Outline

Generative adversarial training (GAN)

Diffusion models