CMSC740
Advanced Computer Graphics

Matthias Zwicker

Fall 2025

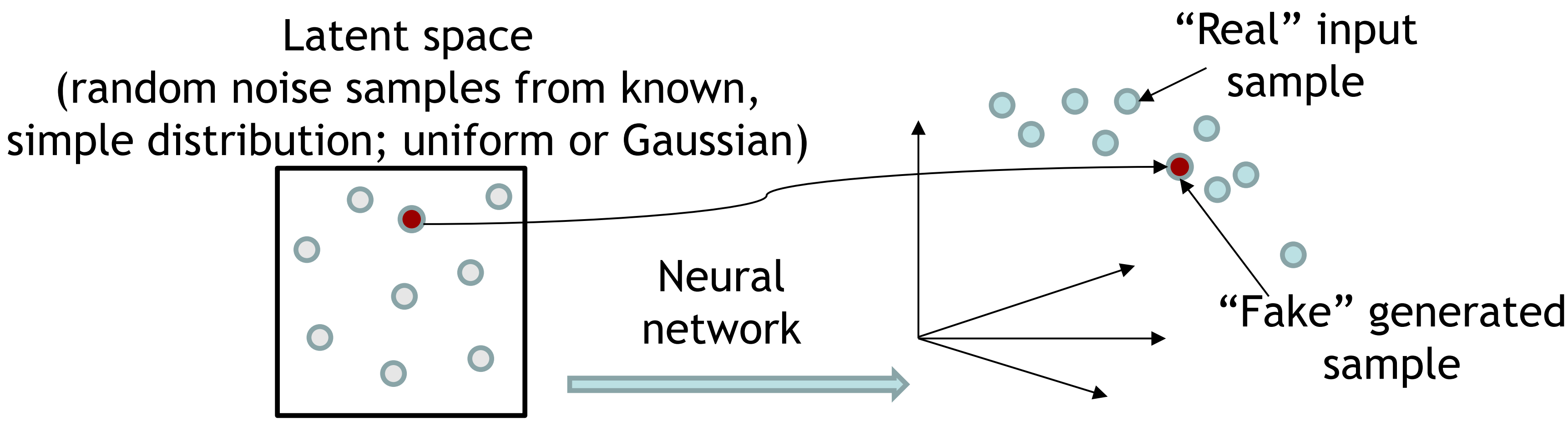
Unsupervised, generative models

- Given: database of objects
- Goal: mechanism that produces random new objects that “look just like objects from the database”
- “Random (Gaussian, uniform) noise -> neural network -> object”, where object can be almost anything (image, video, 3D shape, text, etc.)

Random noise vector → Neural network to compute non-linear function → Output 3D shape

- Applications in graphics: automatically generate images, videos, shapes, textures, animations, …

Abstract point of view

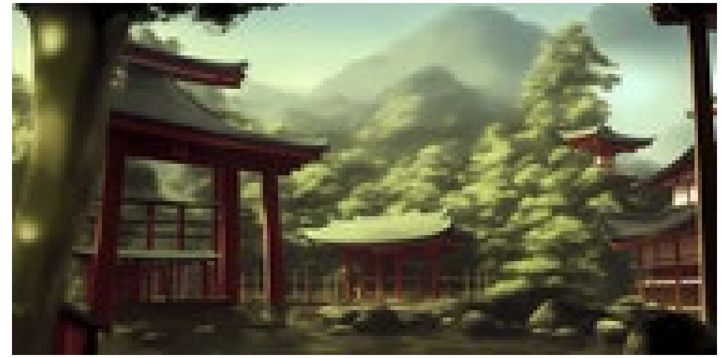


Objects in database interpreted as points in high dimensional space (images, shapes, etc.), i.e., samples of non-uniform probability density

- Generative model: mechanism (neural network) that maps random samples from latent space to points in high dimensional space, such that distribution (density) of generated points matches distribution of given data
- Applications: generate arbitrary amounts of new data samples that “look like” samples from given data distribution (images, video, 3D shapes, text, etc.)

Examples

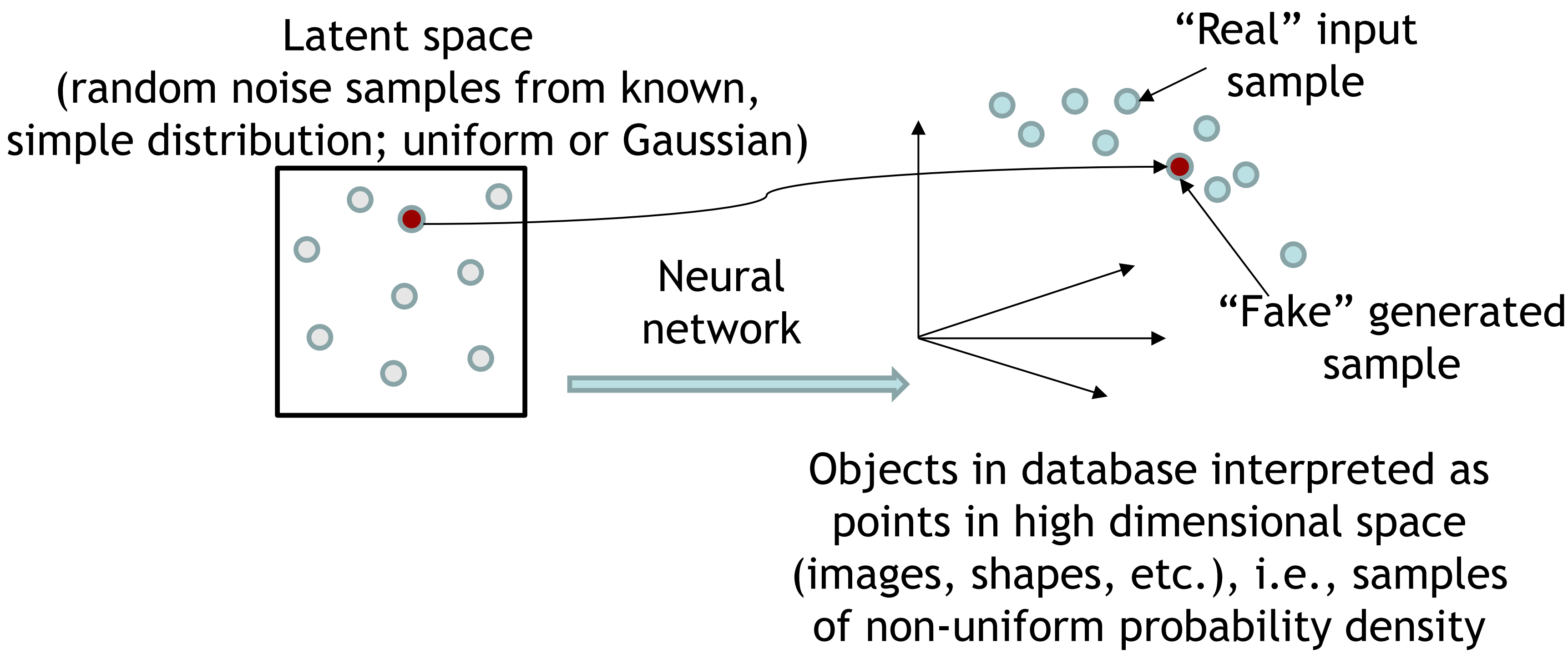
StyleGan (2018) samples, input distribution given by millions of facial images
https://en.wikipedia.org/wiki/StyleGAN



Stable diffusion (2022), input distribution given by hundreds of millions of internet images
https://en.wikipedia.org/wiki/Stable_Diffusion

In practice, often **conditional** generative modeling: provide additional input (condition, such as text label) to model, which then samples from conditional density; same techniques generally apply to unconditional and conditional generative modeling

Abstract point of view



- Challenge: how to formulate the training objective (loss function)?
- Notes
  - Difference between generated and true data density quantified using divergence (similar to metric, but to quantify similarities/distances of probability distributions)
  - Large amount of training data (millions of images, etc.) typically necessary to obtain high-quality generative models (empirical neural scaling laws)

Training objectives

Alternatives

- Estimate probability density of true and generated data as continuous functions, then minimize difference (divergence) between two densities
  - Various divergence measures as generalizations of squared Euclidean distance
  - "Learning Generative Models using Denoising Density Estimators", https://github.com/siavashBigdeli/DDE
- Design learning objective that minimizes the divergence between generated and input density without explicitly estimating the densities at all
  - Generative adversarial networks, https://en.wikipedia.org/wiki/Generative_adversarial_network
- Estimate generated density at any location in high dimensional data space, then maximize log-likelihood of input data samples produced by generator
  - Maximum likelihood estimation, https://en.wikipedia.org/wiki/Maximum_likelihood_estimation
  - Flow-based generative models, https://en.wikipedia.org/wiki/Flow-based_generative_model
  - Variational autoencoders (ELBO approximation), https://en.wikipedia.org/wiki/Variational_autoencoder
- Estimate gradient of density (score) of input data, then use an iterative procedure to sample from it
  - Langevin sampling, https://en.wikipedia.org/wiki/Metropolis-adjusted_Langevin_algorithm, https://github.com/ermongroup/ncsn
- Design process that iteratively maps input density to simple (Gaussian density), then learn to invert it using (approximate) maximum likelihood objective
  - Diffusion models, https://lilianweng.github.io/posts/2021-07-11-diffusion-models/ (has nice overview of other models also)
  - Highly related to Langevin sampling

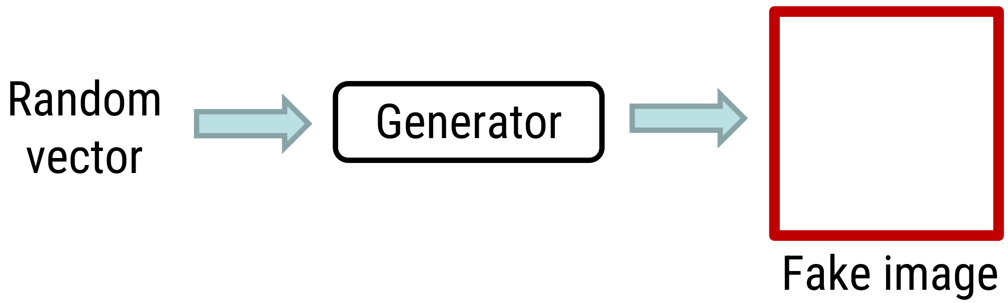
Generative Adversarial Nets

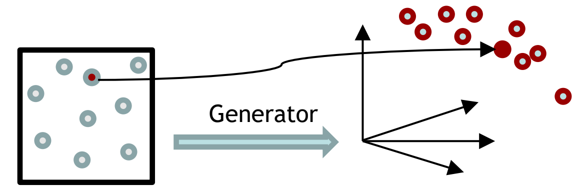
Basic mechanism introduced by Goodfellow et al., 2014

https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

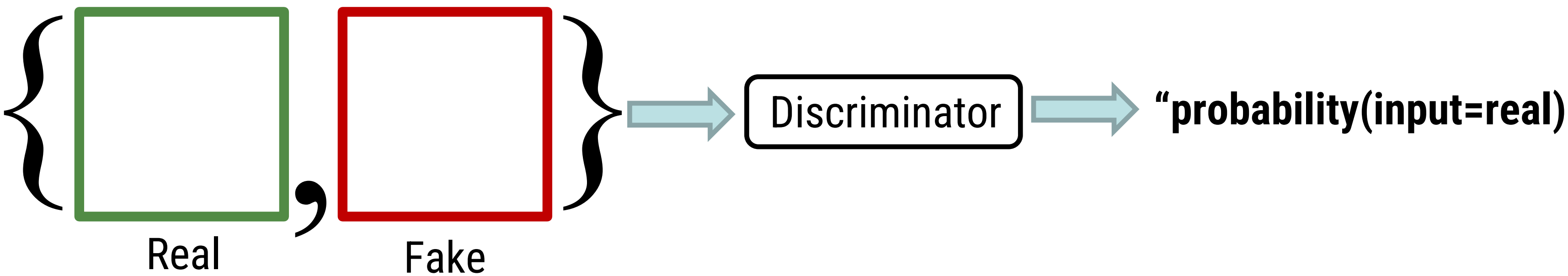
Generative adversarial networks

- Generator network to synthesize objects





- Discriminator network to predict whether object is real



- Train generator and discriminator simultaneously
  - Generator training objective: maximize "**probability(fake input=real)**" of discriminator
  - Discriminator training objective: minimize "**probability(fake input=real)**", maximize "**probability(real input = real)**"

Mathematical formulation

https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

- Generator G, generated object $G(z)$, random input z from known density p_z (uniform, Gaussian)
- Discriminator D, probability that object x is real $D(x)$
- Given data density p_{data}, training objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Two-player minimax game,
optimize parameters of neural networks implementing D, G

Training

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- Expected values in GAN objectives are integrals, evaluate using Monte Carlo sampling
  - Integrals (expected values) turn into sums over samples
- Sampling p_{data} simply by randomly selecting input sample
- Sampling $p_{\mathbf{z}}$ simple, since $p_{\mathbf{z}}$ uniform or Gaussian
- Switch between gradient descent steps to optimize generator ($\min$ in training objective), discriminator training ($\max$)

Theoretical analysis

https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

- GAN training minimizes Jensen-Shannon divergence between generated and given data distribution https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence
- Training converges such that generated = given distribution under some assumptions
  - Unlimited capacity of networks
  - Unlimited amount of training data
- Annotated proof

  https://srome.github.io/An-Annotated-Proof-of-Generative-Adversarial-Networks-with-Implementation-Notes/

KL divergence

- Kullback-Leibler (KL) divergence (discrete case) given by relative entropy

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

  - Set of discrete events X, two different probability distributions P, Q for the events x
- Interpretation
  - Assume we have optimal code for X under distribution $\boldsymbol{Q}$ (optimal code uses fewer bits for events with higher probability)
  - KL divergence is number of extra (“wasted”) bits used when encoding events distributed according to $\boldsymbol{P}$, in average
  - Not symmetric
- Continuous version: replace sum with integral

JS divergence

- Jensen-Shannon (JS) divergence is symmetrized version of KL divergence

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(Q \parallel M)$$

$$M = \frac{1}{2}(P + Q)$$

- Square root of JSD is a metric
  - Distance between identical points is zero, otherwise always positive, symmetric, triangle inequality
- GAN objective can be modified to minimize other metrics, such as Wasserstein distance (Kantorovich-Rubinstein metric, earth mover's distance) https://en.wikipedia.org/wiki/Wasserstein_GAN

Proof steps

- For a given generator width PDF p_{G}, optimal discriminator is

$$D(x) = \frac{p_{data}}{p_{data}+p_G}$$

- Plugging D_G into $V(D,G)$ get $C(G)$ defined as

$$C(G) = -\log 4 + 2 \cdot JSD\left(p_{data} | p_G\right)$$

which is 0 only when $p_G = p_{data}$

- When p_G converges to p_{data}, optimal discriminator becomes

$$D_G^* = \frac{p_{data}}{p_{data} + p_G} = \frac{1}{2}$$

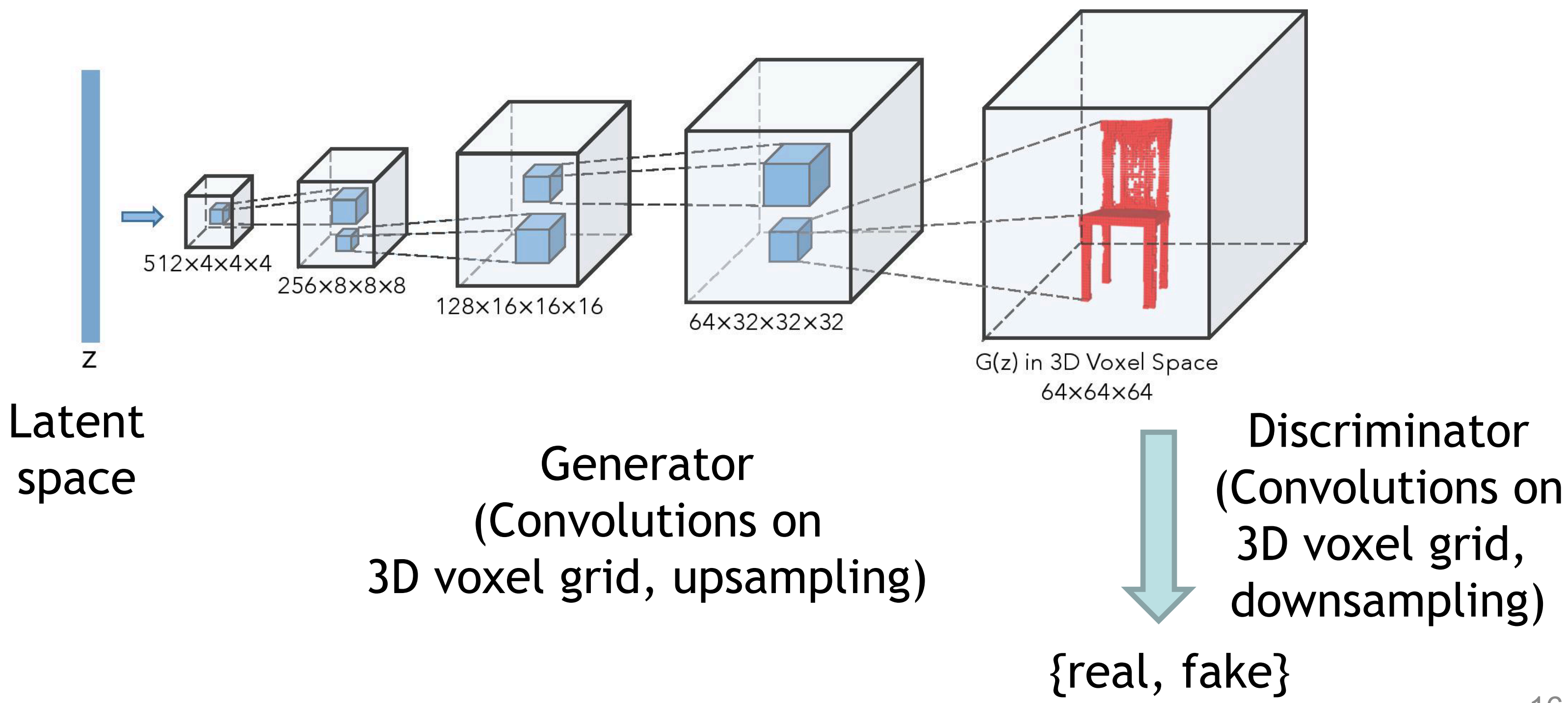
- Hence, at optimum of $V(D,G)$, $p_G = p_{data}$, $D^*_G = 1/2$
- Separate proof for convergence of training procedure

GAN applications

- Original GAN paper [Goodfellow et al. 2014] showed image generation
- Applications to many other types of data
  - Video
  - Shapes/3D models
  - Audio
  - Text
  - Medical data
  - Etc.

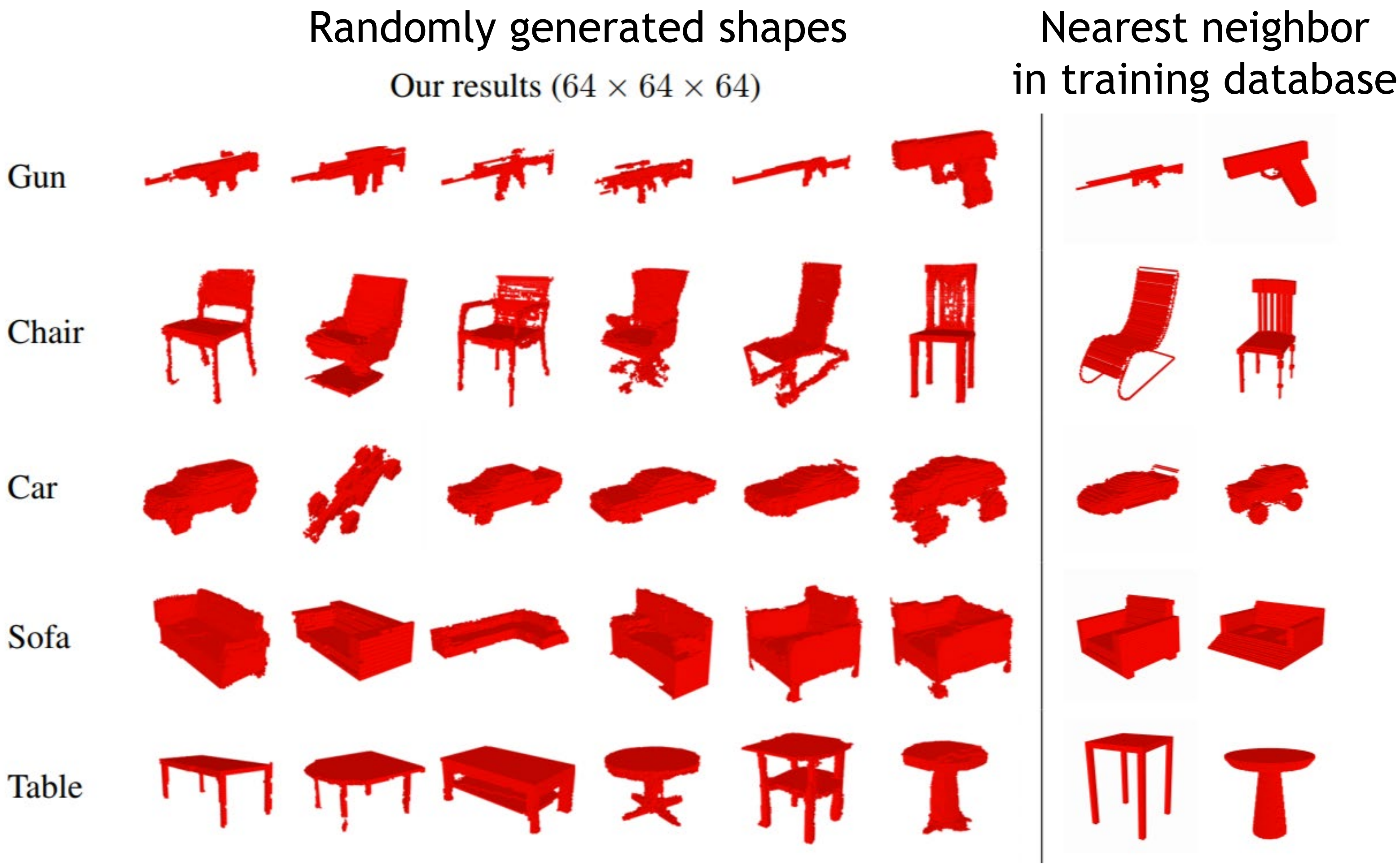
ShapeGAN

- “Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling”, Wu et al., NIPS 2016, http://3dgan.csail.mit.edu/



Results http://3dgan.csail.mit.edu/

- Trained on ShapeNet



http://3dgan.csail.mit.edu/

GAN pros/cons

- Pros
  - Theoretical guarantees to obtain generator for input data distribution
  - Conceptually simple formulation and training
  - Fast sample generation (no iteration required)
- Cons
  - Training convergence unstable in practice
  - Requires manual hyper-parameter tuning
  - Tends to produce “mode collapse” (favors sample quality over diversity of samples)
- Note: theory of GANs isn’t specific to using neural networks as generators, discriminators
- Much recent research on various improvements (different discriminator/generator architectures, ways to feed random input into generator, other divergence metrics)
  - For example, StyleGAN3, NeurIPS 2021 https://github.com/NVlabs/stylegan3
- Illustrative animations of GAN training: https://poloclub.github.io/ganlab/

Diffusion Models

Basic mechanism introduced by Sohl-Dickstein et al., 2015

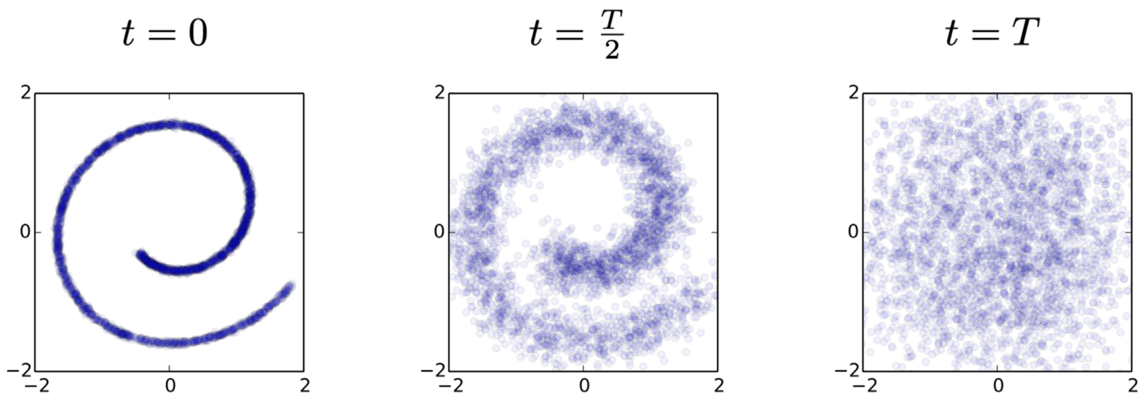
https://arxiv.org/pdf/1503.03585.pdf

Diffusion models

- Basic idea: define forward diffusion process that iteratively maps input density to simple density (Gaussian) by adding noise, then learn to invert it

Forward diffusion, iteratively adding Gaussian noise to data points

Input data distribution q

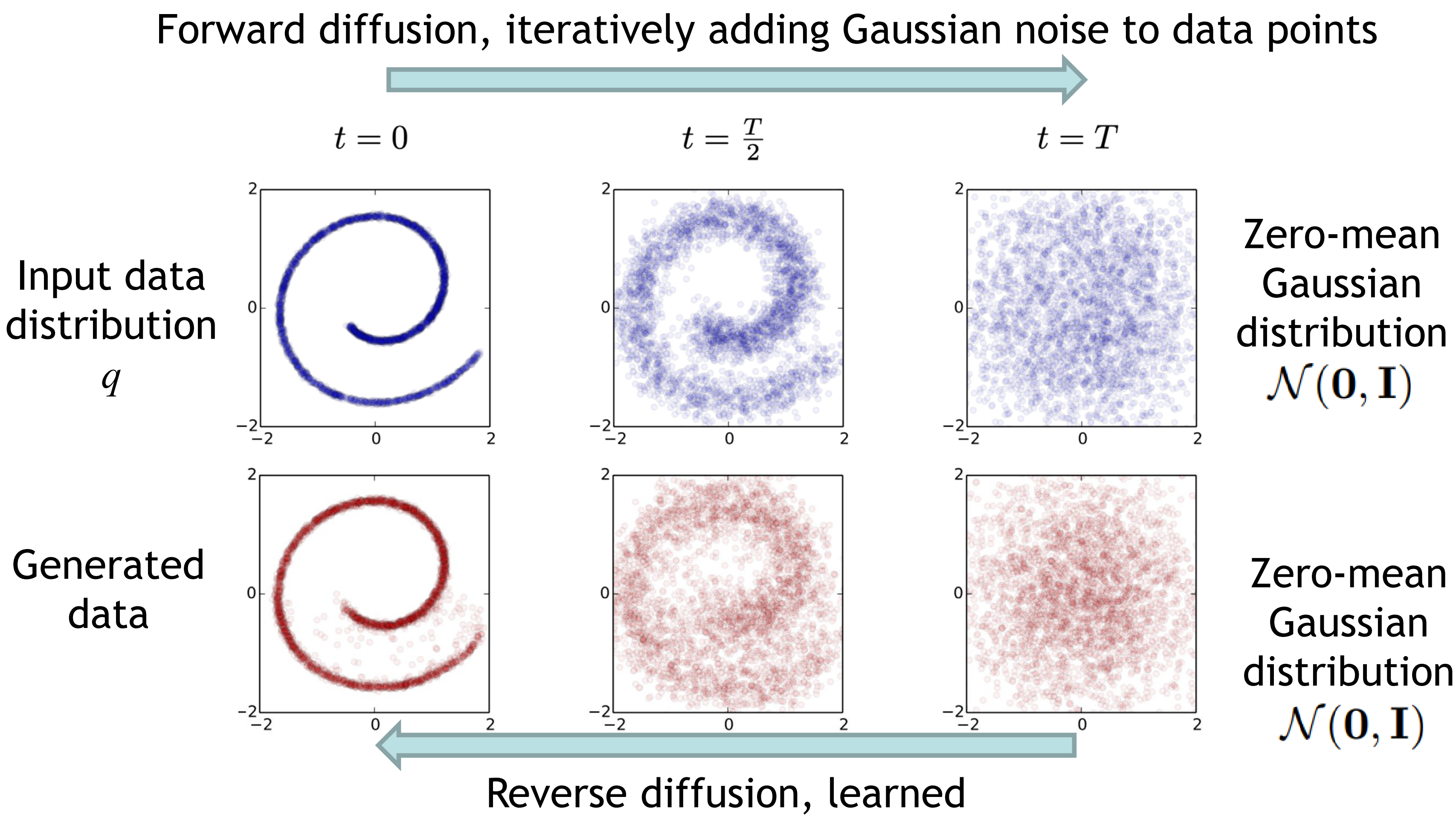


Zero-mean Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$

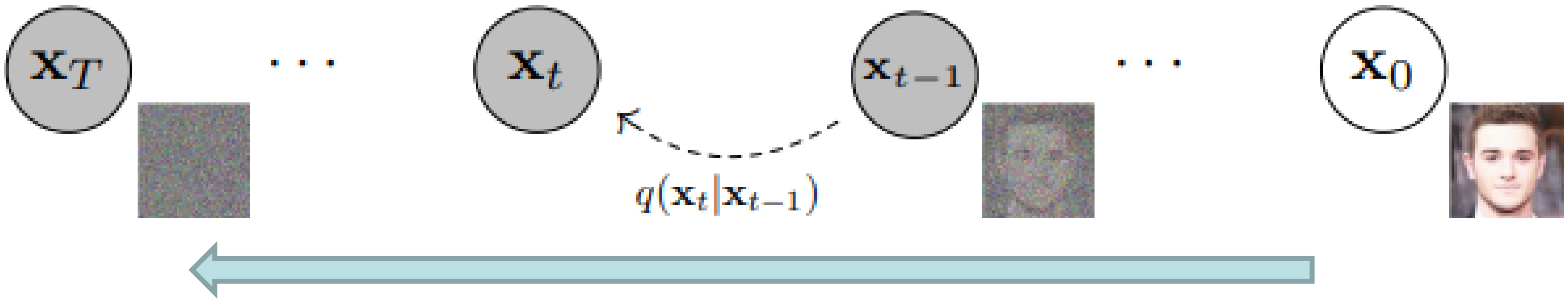
Input data density $q(x_0)$

Diffusion models

- Basic idea: define forward diffusion process that iteratively maps input density to simple density (Gaussian) by adding noise, then learn to invert it



Forward diffusion



Forward diffusion, iteratively adding Gaussian noise to data points (e.g., images)

- Forward diffusion defined as adding Gaussian noise N with given variance β_t in each time step t

mean covariance

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

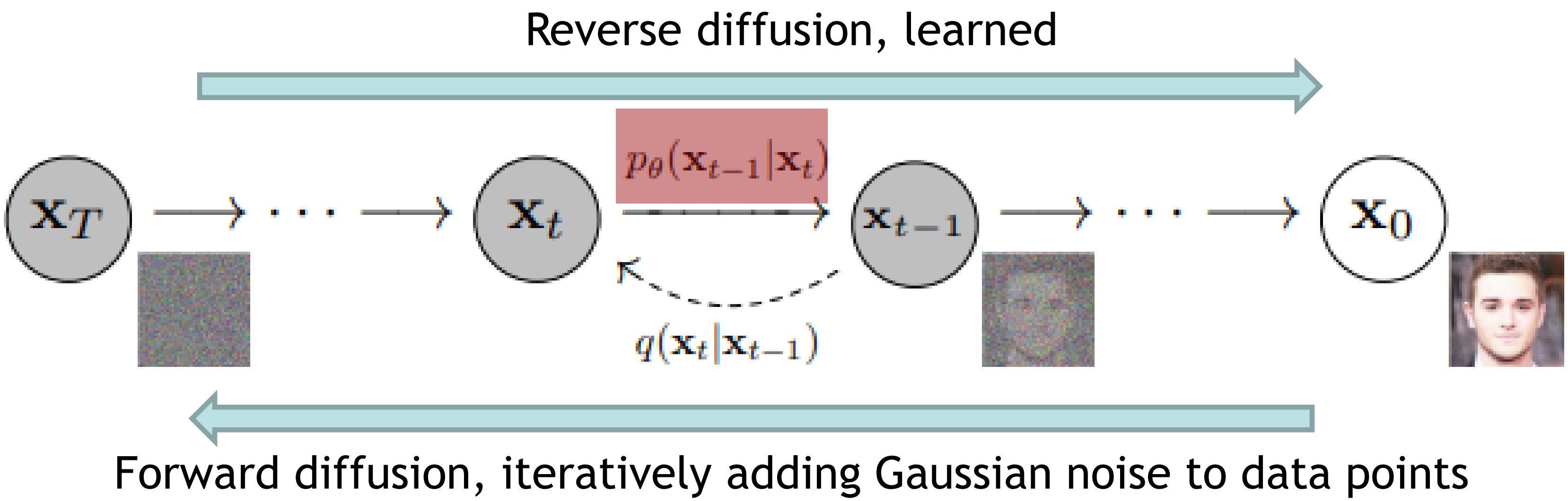
Shrink mean towards 0

Diagonal covariance, independent noise at each pixel

- Joint distribution of entire forward trajectory $\mathbf{x}_{1:T}$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

Reverse diffusion



- Need to know conditional probability $p_\theta(\mathbf{x}_{t-1}|\, \mathbf{x}_t)$ of reverse diffusion step; by sampling $p_\theta(\mathbf{x}_{t-1}|\, \mathbf{x}_t)$ we can make backward steps
- Key observation: if noise β_t small enough, reverse diffusion $p_\theta(\mathbf{x}_{t-1}|\, \mathbf{x}_t)$ has same form has forward step, here Gaussian (Feller, 1949)

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

- But mean μ_θ is unknown at this point (will be able to compute variance Σ_θ explicitly)
- **Goal of training**: generator network with parameters θ will be trained to predict mean μ_θ in each step t

Reverse diffusion

- Joint density of entire reverse trajectory using generator with parameters θ

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

For T -> infinity, isotropic Gaussian distribution

- Marginal density $p_\theta(\mathbf{x}_0)$ of generated output data $\mathbf{x}_0$ using generator with parameters θ

$$p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T})d\mathbf{x}_{1:T} = \int p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\, d\mathbf{x}_{1:T}$$

Integrate over all steps along reverse diffusion

- Given generator with fixed parameters θ, $p_\theta(\mathbf{x}_0)$ is well defined, but integral is intractable to compute in practice (extremely high dimensional)

Generated output density

- Intuition: (marginal) output density $p_\theta(\mathbf{x}_0)$ for each output $\mathbf{x}_0$ is integral over all backward trajectories starting from Gaussian distributed $\mathbf{x}_T$ and ending up at $\mathbf{x}_0$

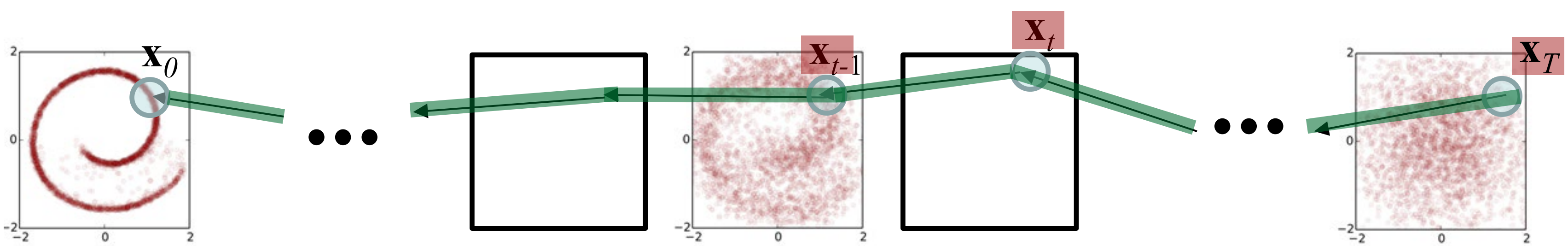
$$p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T})d\mathbf{x}_{1:T} = \int p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\, d\mathbf{x}_{1:T}$$

Trajectory from time T back to 0

Generated output density

- Intuition: (marginal) output density $p_\theta(\mathbf{x}_0)$ for each output $\mathbf{x}_0$ is integral over all backward trajectories starting from Gaussian distributed $\mathbf{x}_T$ and ending up at $\mathbf{x}_0$

$$p_\theta(\mathbf{x}_0) = \int \underbrace{p_\theta(\mathbf{x}_{0:T})}_{\text{Trajectory from time } T \text{ back to } 0} d\mathbf{x}_{1:T} = \int p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\, d\mathbf{x}_{1:T}$$



t=0 t-2 t-1 ⟸ t t=T

Output density $p_\theta(\mathbf{x}_0)$

Reverse diffusion step $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$

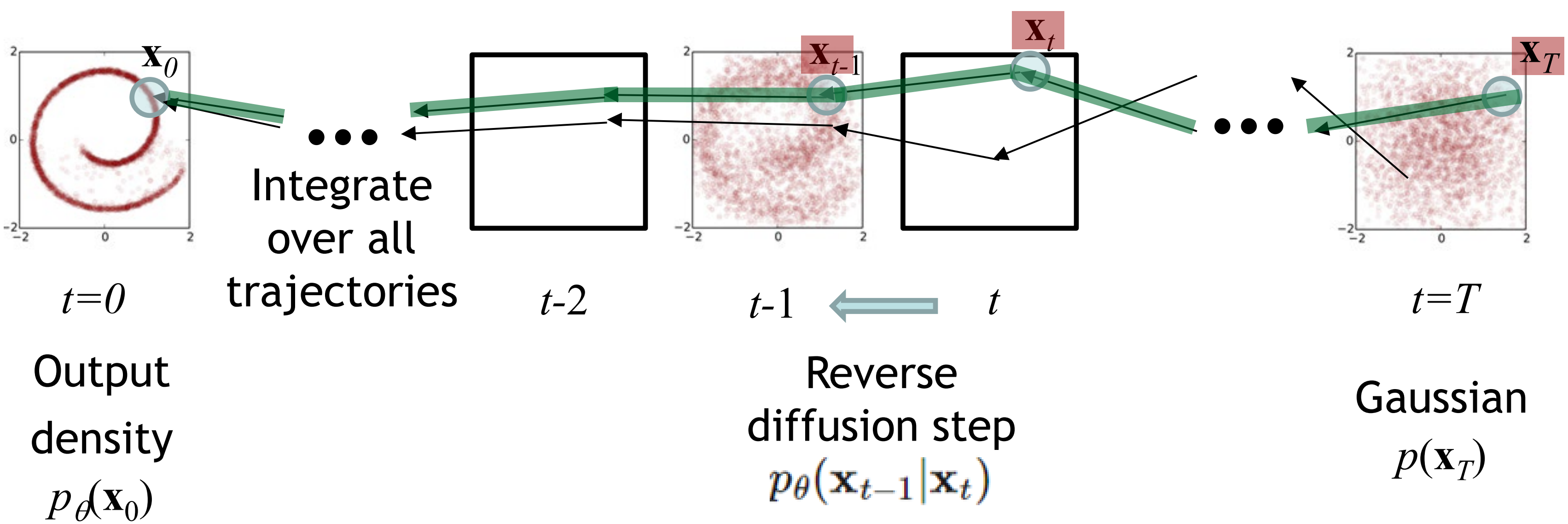
Gaussian $p(\mathbf{x}_T)$

Generated output density

- Intuition: (marginal) output density $p_\theta(\mathbf{x}_0)$ for each output $\mathbf{x}_0$ is integral over all backward trajectories starting from Gaussian distributed $\mathbf{x}_T$ and ending up at $\mathbf{x}_0$

$$p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T})\,d\mathbf{x}_{1:T} = \int p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\, d\mathbf{x}_{1:T}$$

Trajectory from time T back to 0



Training objective

- Minimize negative log-likelihood (i.e., maximize likelihood) of generated output data $-\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0)$
  - "The density of generated data, evaluated where the input data is [to estimate expected value], should be as high as possible" (density of input data denoted q)
  - "Randomly select set of training data points $\mathbf{x}_0$ [for Monte Carlo estimation of $\mathbf{E}_{q(\mathbf{x}0)}$], evaluate their density under the current generator $p_\theta(\mathbf{x}_0)$, and minimize sum of negative logs" (equivalent to maximizing sum of densities)

Training objective

- Goal: Train generator θ to predict Gaussian distributions (means μ_θ) in each step t

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

to minimize negative log-likelihood

$$-\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0)$$

$$= -\mathbb{E}_{q(\mathbf{x}_0)} \log\left(\boxed{\int p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\, d\mathbf{x}_{1:T}} \right)$$

In theory, could optimize this directly over generator parameters θ, but integral **not tractable in practice**

Practical training objective

- For practical training, need to approximate training objective based on negative log-likelihood
- Here, approach based on “Denoising Diffusion Probabilistic Models”, 2020
  https://arxiv.org/pdf/2006.11239.pdf

Practical training objective

- After some math and approximations, find that minimizing negative log-likelihood can be achieved by minimizing KL divergence

Desired inverse diffusion step

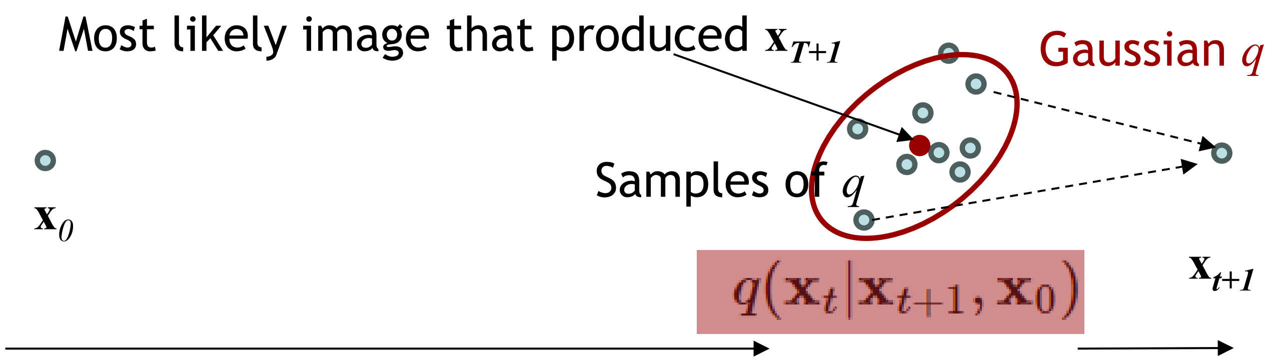
$$D_{\mathrm{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}))$$

where $q(\mathbf{x}_t \mid \mathbf{x}_{t+1}, \mathbf{x}_0)$ is Gaussian with known parameters $\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}$

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I})$$

- "If we knew original image $\mathbf{x}_0$, then we could know which image $\mathbf{x}_{t-1}$ most likely produced $\mathbf{x}_t$ in forward diffusion step"; would allow us to make inverse step
- However, during reverse diffusion, $\mathbf{x}_0$ unknown, cannot use $q(\mathbf{x}_t \mid \mathbf{x}_{t+1}, \mathbf{x}_0)$ directly in reverse diffusion

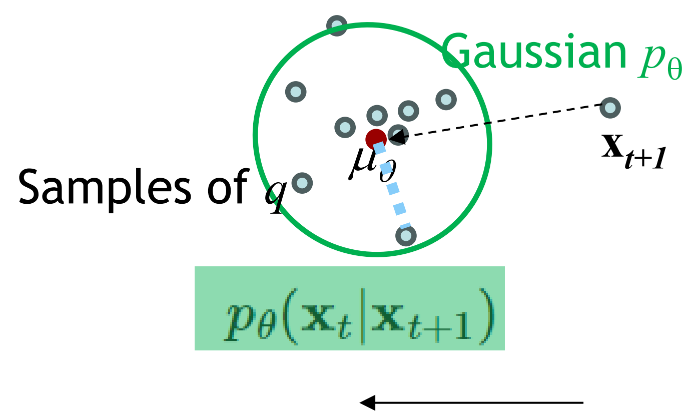
Visualization



Forward diffusion, known distributions

Minimize $D_{\mathrm{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}))$

Key observation: KL is minized when mean μ_θ of p_θ minimizes L2 distance to mean of q; achieved by minizing L2 distance between samples of q and mean μ_θ of p_θ



Reverse diffusion, distribution p_θ of reverse step to be learned

Minimizing KL divergence

$D_{\mathrm{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}))$

- Minimizing KL divergence means minimizing prediction error for mean, over all input data $\mathbf{x}_0$ and noise vectors ε

Loss to be minimized

$$L_t = \mathbb{E}_{\mathbf{x}_0,\epsilon}\Big[\frac{1}{2\|\boldsymbol{\Sigma}_\theta(\mathbf{x}_t,t)\|_2^2}\|\underbrace{\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t,\mathbf{x}_0)}_{\text{Known}} - \underbrace{\boldsymbol{\mu}_\theta(\mathbf{x}_t,t)}_{\text{Predicted by generator}}\|^2\Big]$$

- "For all input images $\mathbf{x}_0$ and noise ε, train generator μ_θ to denoise image $\mathbf{x}_t$, which predicts mean $\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0)$ "

Minimizing KL divergence

$D_{\mathrm{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}))$

- Paper shows: predicting denoised image (i.e., sample of q) is equivalent to predicting noise ε_t that was added

Known noise ε_t applied in forward diffusion

Known noise ε_t applied to input data $\mathbf{x}_0$

$$L_t = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\Big[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t}, t)\|^2\Big]$$

Minimize loss wrt. parameters θ

Noise predicted by generator network, parameters θ

Noisy input image at time t

$$\alpha_t = 1 - \beta_t \qquad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i \qquad \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \dots \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Variances used in forward diffusion

Gaussian noise

Summary

- Approximation of training objective (minimizing negative log-likelihood of generated data density) can be achieved by training noise prediction network ε_θ
- Training
  - Construct noisy data by adding known noise vectors to clean data points, noise magnitudes (variance) given by time t
  - Given noisy inputs, time t, train network $\varepsilon_\theta(\text{noisy image}, t)$ to predict noise

Training & sampling algorithms

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ // randomly select input image
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Simplified training objective,
works better in practice

$$\alpha_t = 1 - \beta_t \qquad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

Variances used in forward diffusion

Training & sampling algorithms

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ // randomly select input image
3: $t \sim \text{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ // start with Gaussian noise image
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \underbrace{\frac{1-\alpha_t}{\sqrt{1-\bar\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}_{\text{subtract predicted noise}}\right) \underbrace{+ \sigma_t \mathbf{z}}_{\text{add new noise}}$$
5: **end for**
6: **return** $\mathbf{x}_0$

Simplified training objective, works better in practice

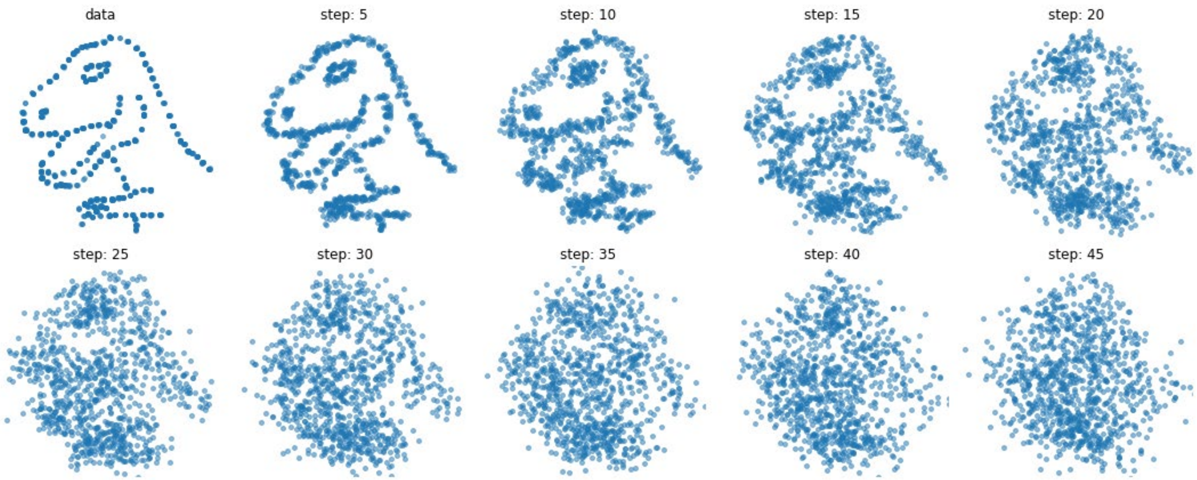
Sample Gaussian $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$, different choices for σ_t in practice, such as $\sigma_t^2 = \beta_t$

$$\alpha_t = 1 - \beta_t \qquad \bar\alpha_t = \prod_{i=1}^{t} \alpha_i$$

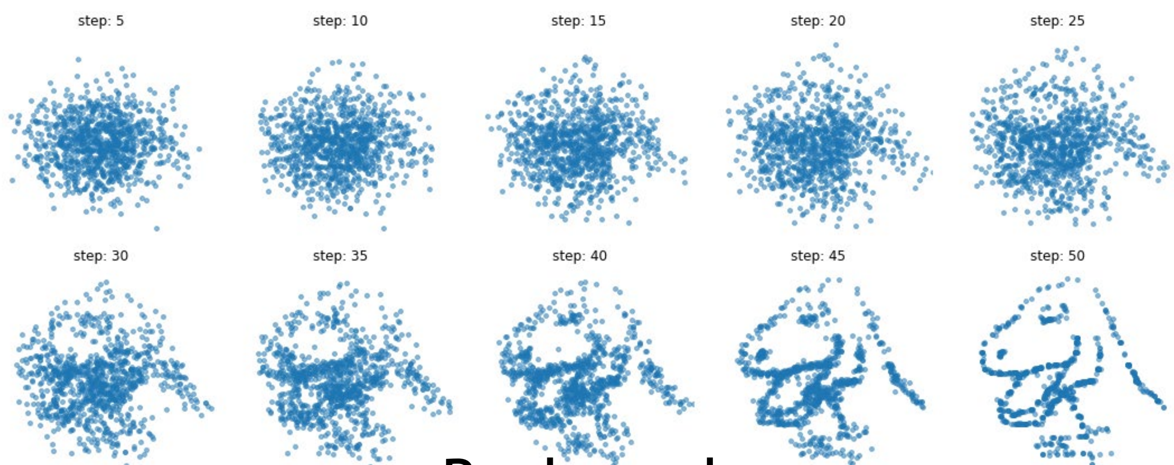
Variances used in forward diffusion

Educational 2D model

https://github.com/tanelp/tiny-diffusion



Forward



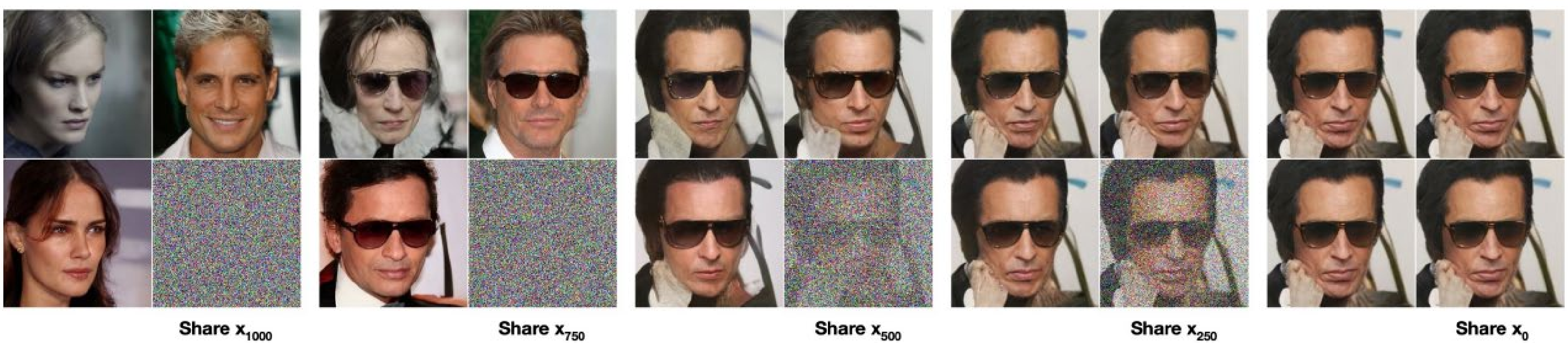
Backward

Results

Gaussian noise

Reverse diffusion steps by sampling $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

Input data distribution



Different end points of reverse trajectories starting at different t