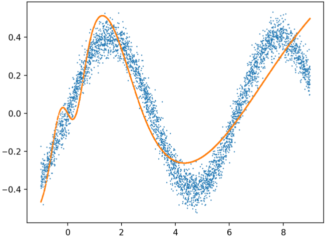
Deep Generative Models

Ju Sun
Computer Science & Engineering
University of Minnesota, Twin Cities

December 13, 2023

Supervised learning

supervised learning: find functional relationship between input $\boldsymbol{x}$ and target $\boldsymbol{y}$



- Underlying true function: f_0
- Training data: $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$ with $\boldsymbol{y}_i \approx f_0(\boldsymbol{x}_i)$
- Choose a family of functions $\mathcal{H}$, so that $\exists f \in \mathcal{H}$ that is close to f_0
- Find f, i.e., optimization

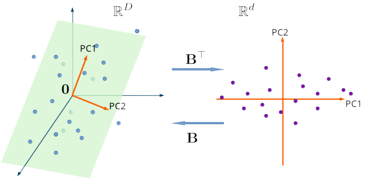
$$\min_{f \in \mathcal{H}} \sum_i \ell\left(\boldsymbol{y}_i, f(\boldsymbol{x}_i)\right) + \Omega(f)$$

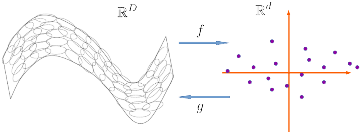
classification ($\boldsymbol{y}$ also called labels) or regression

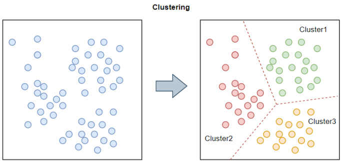
object recognition, semantic segmentation, object detection, machine translation, image captioning, sentiment analysis, etc

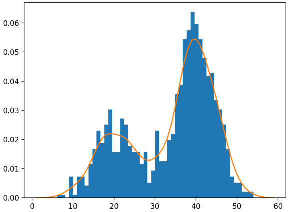
Unsupervised learning

unsupervised learning: discover hidden structure in data, e.g., dimension reduction (subspace/manifold learning/sparse coding), clustering (e.g., k-means, spectral clustering), density estimation (e.g., GMM), etc







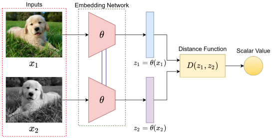


(Credit: https://www.ecloudvalley.com/)

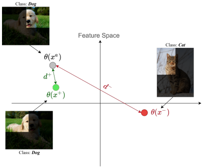
(Credit: https://machinelearningmastery.com/)

Self-supervised learning

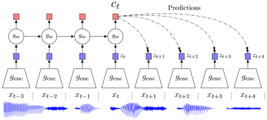
Marriage of unsupervised (no target) & supervised (contrived target) learning



joint embedding



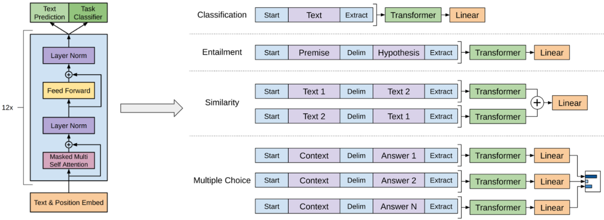
contrastive learning



sequential prediction

Pretraining: (transformer-based) **decoder-only** architectures pretrained on **language modelir** $\mathbb{P}\left[\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)}\right]$

Finetuning: on task-specific supervised data

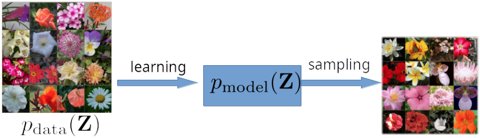


Model distributions

notations: X, Y—RVs or events; $\boldsymbol{x}, \boldsymbol{y}$—realization of RVs;
$\mathbb{P}[A]$—probability of event A; p: probability density function;
$\boldsymbol{\theta}$ in $p(\cdot; \boldsymbol{\theta})$—parameters in parametrized density p

generative model: model $p(Z)$, i.e., distribution of data $\boldsymbol{Z}$, as new samples can be generated

- generate new samples



- quantitative analysis: mean, variance, moments, probability (e.g., rare events)
- Bayesian learning/latent variable models: X—observations, Z—hidden variables

$$\mathbb{P}[Z \mid X] = \mathbb{P}[X \mid Z]\, \mathbb{P}[Z] / \mathbb{P}[X]$$

involving probability density functions

Challenges

given samples $\{\boldsymbol{z}_i\}$ drawn iid from $p_{\text{data}}(Z)$

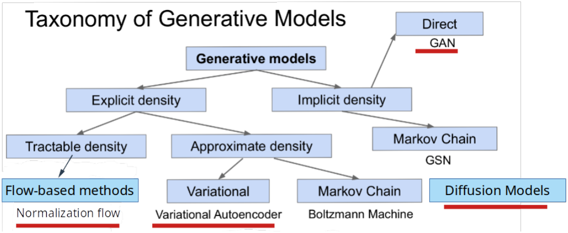
- **modeling**: which $p_{\text{model}}(Z; \boldsymbol{\theta})$ gives good approximation?
- **computation**:
  * parameter estimation: invoking *maximum likelihood estimation*

    $$\max_{\boldsymbol{\theta}} \prod_i p_{\text{model}}(\boldsymbol{z}_i; \boldsymbol{\theta}) \Longleftrightarrow \max_{\boldsymbol{\theta}} \ g(\theta) \doteq \sum_i \log p_{\text{model}}(\boldsymbol{z}_i; \boldsymbol{\theta})$$

    $g(\theta)$ can be highly nonconvex and hence hard to solve globally
  * quantitative analysis: many quantities, e.g., mean, variance, moments, likelihood, involve *high-dimensional integrals*
- **sampling**: *high-dimensional sampling* from even a known $p_{\text{model}}(Z; \boldsymbol{\theta})$ is expensive, e.g., Markov Chain Monte Carlo (MCMC)

classic methods: mixture models (e.g., GMM), kernel density estimation (Parzen window method)

Models we'll cover



(Credit: adapted from Stanford CS231N slides)

models focused on direct data generation

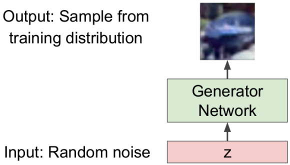
- Adversarial Generative Networks (GANs) [Goodfellow et al., 2014]
- Variational Autoencoder (VAE) [Kingma and Welling, 2013, Kingma and Welling, 2019]
- Diffusion Models [Sohl-Dickstein et al., 2015, Yang et al., 2022]

models focused on density modeling (and data generation)

- Normalization Flow [Rezende and Mohamed, 2015, Papamakarios et al., 2019]

Outline

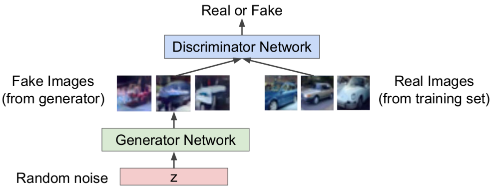
Learning via competition



(Credit: Stanford CS231N)

- **task**: given $\{x_i\}$, generate new samples from the distribution
- **idea**: map samples from a **simple** distribution to samples from the training distribution
- **how to**: measure the difference between **mapped/training** distributions

One solution: introduce a **biased** critic/discriminator



(Credit: Stanford CS231N)

- **Discriminator**: try to distinguish real (training) vs. fake (mapped) samples
- **Generator**: try to "fool" the discriminator by learning to generate realistic-looking images

Hope: the generator learns enough when the competition is stabilized

GAN objective

- **Discriminator**: try to distinguish real (training) vs. fake (mapped) samples
- **Generator**: try to "fool" the discriminator by learning to generate realistic-looking images

Hope: the generator learns enough when the competition is stabilized

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \ \mathbb{E}_{\boldsymbol{x} \sim p_{\text{train}}} \log \underbrace{D_{\boldsymbol{\theta}_d}(\boldsymbol{x})}_{\text{discriminator output for real data}} + \mathbb{E}_{\boldsymbol{z} \sim p(Z)} \log[1 - \underbrace{D_{\theta_d}\left(G_{\theta_g}(\boldsymbol{z})\right)}_{\text{discriminator output for fake data}}]$$

- discriminator output: $(0, 1)$ indicates likelihood of being real image (e.g., by passing sigmoid activations at output)
- discriminate wants to **maximize** the objective so that $D_{\boldsymbol{\theta}_d}(\boldsymbol{x})$ is close to 1 and $D_{\theta_d}\left(G_{\theta_g}(\boldsymbol{z})\right)$ is close to 0
- generators wants to **minimize** the objective so that $D_{\theta_d}\left(G_{\theta_g}(\boldsymbol{z})\right)$ is close to 1

Training GANs: minimax optimization

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \mathbb{E}_{\boldsymbol{x} \sim p_{\text{train}}} \log D_{\boldsymbol{\theta}_d}(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z} \sim p(Z)} \log[1 - D_{\theta_d}\left(G_{\theta_g}(\boldsymbol{z})\right)]$$

minimax (saddle point) optimization—way harder than minimization

heuristic algorithm: alternate between

- maximize wrt $\boldsymbol{\theta}_d$ using **gradient ascent**

$$\max_{\boldsymbol{\theta}_d} \mathbb{E}_{\boldsymbol{x} \sim p_{\text{train}}} \log D_{\boldsymbol{\theta}_d}(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z} \sim p(Z)} \log[1 - D_{\theta_d}\left(G_{\theta_g}(\boldsymbol{z})\right)]$$

- minimize wrt $\boldsymbol{\theta}_g$ using **gradient descent**

$$\min_{\boldsymbol{\theta}_g} \mathbb{E}_{\boldsymbol{z} \sim p(Z)} \log[1 - D_{\theta_d}\left(G_{\theta_g}(\boldsymbol{z})\right)]$$

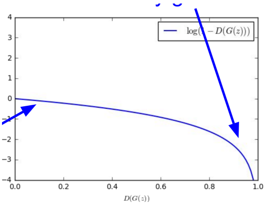
No guarantee this will work... [Razaviyayn et al., 2020]

Training GANs: the trick

minimize wrt $\boldsymbol{\theta}_g$ using **gradient descent**

$$\min_{\boldsymbol{\theta}_g} \mathbb{E}_{\boldsymbol{z}\sim p(Z)} \log[1 - D_{\theta_d}(G_{\theta_g}(\boldsymbol{z}))]$$

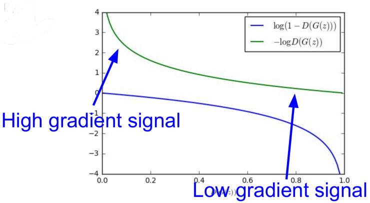
... but bit of numerical issue



(Credit: Stanford CS231N)

– grad. large when $D(G(\boldsymbol{z}))$ is large, i.e., generator is already good

– grad. small when $D(G(\boldsymbol{z}))$ is small, i.e., generator is not good, esp. at the initial stage—bad!



trick:

$\min_{\boldsymbol{\theta}_g} \mathbb{E}_{\boldsymbol{z}\sim p(Z)} -\log D_{\theta_d}(G_{\theta_g}(\boldsymbol{z}))$

instead

GAN training pipeline

for number of training iterations **do**

  for $\boxed{k \text{ steps}}$ **do**

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

  • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Update the generator by ascending its stochastic gradient (improved objective):

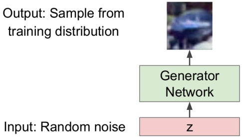
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

(Credit: Stanford CS231N)

– often $k > 1$ in practice to train the discriminator slightly faster

– overall still tricky and involves lots of tuning after training, use the generator to generate new samples



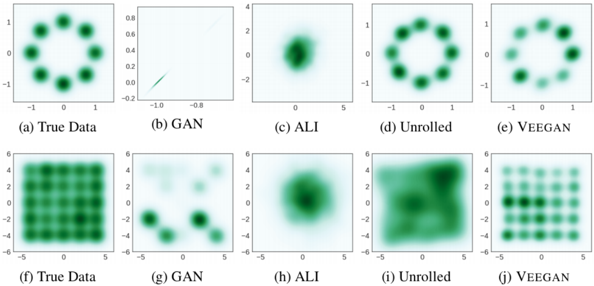
(Credit: Stanford CS231N)

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples

(Credit: [Goodfellow et al., 2014])

mode collapse: samples only generated from a subset of original support

Figure 2: Density plots of the true data and generator distributions from different GAN methods trained on mixtures of Gaussians arranged in a ring (top) or a grid (bottom).



(Credit: [Srivastava et al., 2017])

What really happens in the competition?

Lemma ([Goodfellow et al., 2014])

For any fixed G, the optimal discriminator D is $D_G^*(\boldsymbol{x}) = \frac{p_{train}(\boldsymbol{x})}{p_{train}(\boldsymbol{x})+p_{g(\boldsymbol{x})}}$.

Then

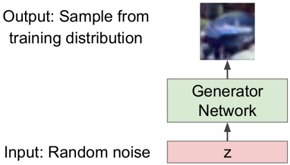
$$
\begin{aligned}
C(G) &= \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{train}}} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z}\sim p(Z)} \log[1 - D(G(\boldsymbol{z}))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{train}}} \log D_G^*(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{x}\sim p_g} \log[1 - D_G^*(\boldsymbol{x})] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{train}}} \log \frac{p_{\text{train}}(\boldsymbol{x})}{p_{\text{train}}(\boldsymbol{x}) + p_g(\boldsymbol{x})} + \mathbb{E}_{\boldsymbol{x}\sim p_g} \log \frac{p_g(\boldsymbol{x})}{p_{\text{train}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}
\end{aligned}
$$

Theorem ([Goodfellow et al., 2014])

The global minimizer of $C(G)$ is at $p_g = p_{train}$.

Also, $C(G) = D_{\text{KL}}\left(P_{\text{train}} \| \frac{P_{\text{train}}+P_g}{2}\right) + D_{\text{KL}}\left(P_g \| \frac{P_{\text{train}}+P_g}{2}\right) - \log 4$, where D_{KL} denotes the KL-divergence KL $(P\|Q) = \int p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\, d\boldsymbol{x}$

Wasserstein GAN



(Credit: Stanford CS231N)

- **task**: given $\{x_i\}$, generate new samples from the distribution
- **idea**: map samples from a **simple** distribution to samples from the training distribution
- **how to**: measure the difference between **mapped/training** distributions

Jensen-Shannon (JS) divergence:

$$D_{\text{JS}}(P\|Q) \doteq \frac{1}{2} D_{\text{KL}}\left(P \parallel \frac{P+Q}{2}\right) + \frac{1}{2} D_{\text{KL}}\left(Q \parallel \frac{P+Q}{2}\right)$$

GAN tries to minimize $C(G) = 2D_{\text{JS}}(P_{\text{train}}\|P_g) + \text{const}$

Are there better measures for the difference?

Are all metrics created equal?

measure difference/distance between **distributions**

- **Total variation (TV) distance**

$$\mathsf{TV}\left(P \| Q\right) = \sup_{A \in \mathcal{F}} \left|\mathbb{P}(A) - \mathbb{Q}(A)\right| \quad (\mathcal{F} : \text{sigma-algebra})$$

  i.e., largest discrepancy of probabilities over all events A's

- **Kullback-Leibler (KL) divergence** — asymmetric

$$\mathsf{KL}\left(P \| Q\right) = \int p\left(\boldsymbol{x}\right) \log \frac{p\left(\boldsymbol{x}\right)}{q\left(\boldsymbol{x}\right)} \, d\boldsymbol{x}$$

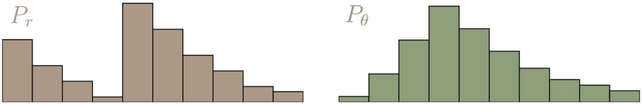
- **Jensen-Shannon (JS) divergence** — symmetric

$$D_{\mathsf{JS}}\left(P \| Q\right) = \frac{1}{2} D_{\mathsf{KL}}\left(P \parallel \frac{P+Q}{2}\right) + \frac{1}{2} D_{\mathsf{KL}}\left(Q \parallel \frac{P+Q}{2}\right)$$

- **Earth-Mover (EM) or Wasserstein-1 distance (or Kantorovich–Rubinstein metric)**

$$W_1\left(P, Q\right) = \inf_{\gamma \in \Gamma(P,Q)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}) \sim \gamma} \left\|\boldsymbol{x} - \boldsymbol{y}\right\|$$

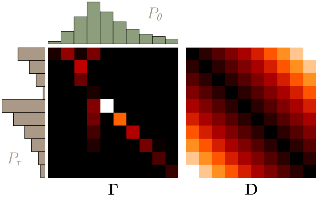
Earth mover distance/W-distance



(Credit: https://vincentherrmann.github.io/blog/wasserstein/)

problem: move around stacks of earth P_r to match the shape/distribution of P_θ

goal: minimize the total movement effort, measured in **distance** $\times$ **volume**



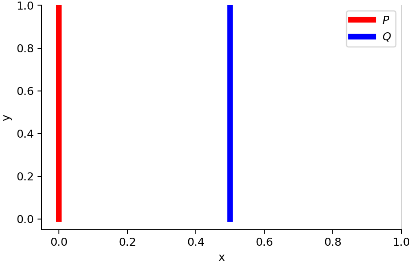
(Credit: https://vincentherrmann.github.io)

restrict P_θ and P_r to distributions

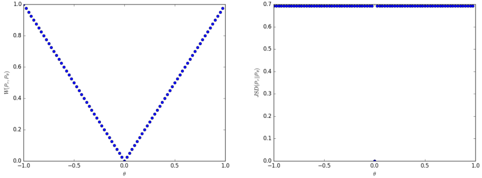
- let γ be a **transport plan**, Γ set of all plans.
- $\sum_x \gamma(x, y) = P_r(y) \; \forall \; y$: total move-in mass matches the target
- $\sum_y \gamma(x, y) = P_\theta(x) \; \forall \; x$: total move-out mass matches the original
- EMD $(P_r, P_\theta) = \inf_{\gamma \in \Gamma} \sum_{x,y} \|x - y\| \gamma(x, y) = \inf_{\gamma \in \Gamma} \mathbb{E}_{(x,y)\sim\gamma} \|x - y\|$

Why Wasserstein distance?

$$P : (0, W), W \sim \text{Uniform}(0,1)$$
$$Q : (\theta, V), V \sim \text{Uniform}(0,1)$$



- $\text{TV}(P \| Q) = \begin{cases} 1 & \theta \neq 0 \\ 0 & \theta = 0 \end{cases}$
- $D_{\text{KL}} = \begin{cases} \infty & \theta \neq 0 \\ 0 & \theta = 0 \end{cases}$
- $D_{\text{JS}} = \begin{cases} \log 2 & \theta \neq 0 \\ 0 & \theta = 0 \end{cases}$
- $W_1(P, Q) = |\theta|$



(Credit: [Arjovsky et al., 2017])

W-distance vs. JS-divergence

Wasserstein GAN (WGAN)

$$W_1(P,Q) = \inf_{\gamma \in \Gamma(P,Q)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\gamma} \|\boldsymbol{x}-\boldsymbol{y}\|$$

hard to compute due to the minimization with the Γ constraint

Lemma (Kantorovich-Rubinstein duality)

$W_1(P,Q) = \sup_{f:\|f\|_L \le 1} \left(\mathbb{E}_{\boldsymbol{x}\sim P} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}\sim Q} f(\boldsymbol{x})\right)$, where $\|f\|_L \le 1$ denotes all functions with Lipschitz constant no greater than 1. (A function f is Lipschitz with a constant C if $\|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \le C\|\boldsymbol{x}-\boldsymbol{y}\|$ for all $\boldsymbol{x},\boldsymbol{y}$.)

WGAN:

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \mathbb{E}_{\boldsymbol{x}\sim p_{\text{train}}} D_{\theta_d}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p(Z)} D_{\boldsymbol{\theta}_d}\left(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})\right) \quad \text{s. t. } \|D_{\boldsymbol{\theta}_d}\|_L \le 1.$$

GAN:

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \mathbb{E}_{\boldsymbol{x}\sim p_{\text{train}}} \log D_{\boldsymbol{\theta}_d}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p(Z)} - \log[1 - D_{\theta_d}\left(G_{\theta_g}(\boldsymbol{z})\right)]$$

Isn't it just a simple modification?

WGAN-GP

To train

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \mathbb{E}_{\boldsymbol{x} \sim p_{\text{train}}} D_{\theta_d}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z} \sim p(Z)} D_{\boldsymbol{\theta}_d}\left(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})\right) \quad \text{s. t. } \|D_{\boldsymbol{\theta}_d}\|_L \leq 1$$

the challenge is to enforce the Lipschitz constraint. Two heuristic:

- **weight clipping**: clip the discriminator $\boldsymbol{\theta}_d$ into a predefined range $(-c, c)$. But performance sensitive to choice of c
- **gradient penalty (GP)**: $\|D_{\theta_d}\|_L \leq 1 \Longrightarrow \|\nabla_{\boldsymbol{x}} D\| \leq 1$, and max achieved when equality holds $\Longrightarrow$ encourage $\|\nabla_{\boldsymbol{x}} D\| = 1$

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \Big[\mathbb{E}_{\boldsymbol{x} \sim p_{\text{train}}} D_{\theta_d}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z} \sim p(Z)} D_{\boldsymbol{\theta}_d}\left(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})\right) - \lambda \mathbb{E}_{\widetilde{\boldsymbol{x}} \sim \widetilde{p}}\left(\|\nabla_{\widetilde{\boldsymbol{x}}} D_{\boldsymbol{\theta}_d}(\widetilde{\boldsymbol{x}})\| - 1\right)^2\Big],$$

where $\widetilde{\boldsymbol{x}} = t\boldsymbol{x} + (1-t) G_{\boldsymbol{\theta}_g}(\boldsymbol{z})$ with $t \sim$ uniform $(0, 1)$

WGAN-GP training

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \left[\mathbb{E}_{\boldsymbol{x} \sim p_{\text{train}}} D_{\theta_d}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z} \sim p(Z)} D_{\boldsymbol{\theta}_d}\left(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})\right) - \lambda \mathbb{E}_{\widetilde{\boldsymbol{x}} \sim \widetilde{p}} \left(\left\|\nabla_{\widetilde{\boldsymbol{x}}} D_{\boldsymbol{\theta}_d}(\widetilde{\boldsymbol{x}})\right\| - 1\right)^2\right],$$

where $\widetilde{\boldsymbol{x}} = t\boldsymbol{x} + (1-t)\, G_{\boldsymbol{\theta}_g}(\boldsymbol{z})$ with $t \sim$ uniform $(0, 1)$

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

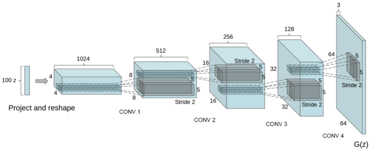
```
while θ has not converged do
    for t = 1, ..., n_critic do
        for i = 1, ..., m do
            Sample real data x ~ P_r, latent variable z ~ p(z), a random number ε ~ U[0, 1].
            x̃ ← G_θ(z)
            x̂ ← εx + (1 − ε)x̃
            L^(i) ← D_w(x̃) − D_w(x) + λ(‖∇_x̂ D_w(x̂)‖_2 − 1)^2
        end for
        w ← Adam(∇_w (1/m) Σ_{i=1}^m L^(i), w, α, β1, β2)
    end for
    Sample a batch of latent variables {z^(i)}_{i=1}^m ~ p(z).
    θ ← Adam(∇_θ (1/m) Σ_{i=1}^m −D_w(G_θ(z)), θ, α, β1, β2)
end while
```

(Credit: [Gulrajani et al., 2017])

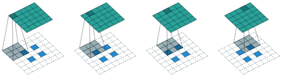
Deep convolutional GAN (DC-GAN)

both the generator $G_{\boldsymbol{\theta}_g}$ and discriminator $D_{\boldsymbol{\theta}_d}$ are deep convolutional networks [Radford et al., 2015]

the generator based on transposed (fractionally strided) convolutions



zero-padding, zero-dilution, then convolution



forward stride = 2

https://github.com/vdumoulin/conv_arithmetic

PyTorch implementation:

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

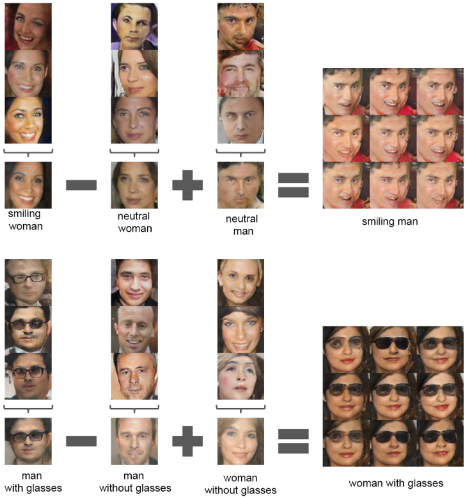
DC-GAN results

image generation

z interpolation

DC-GAN results

image arithmetic



Conditional GAN

augment additional info to both D and G inputs [Mirza and Osindero, 2014]

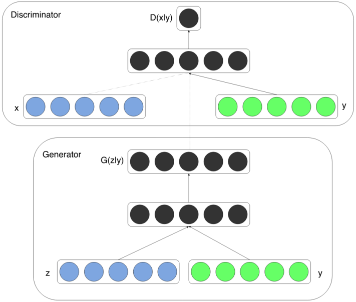
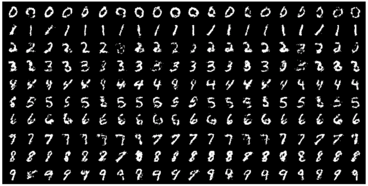


image + label (cond.)

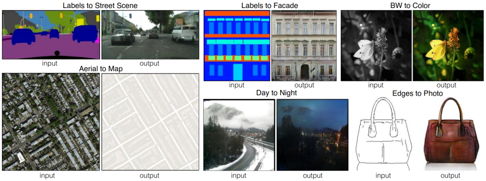


text generation + image (cond.)

	User tags + annotations	Generated tags
	montanha, trem, inverno, frio, people, male, plant life, tree, structures, transport, car	taxi, passenger, line, transportation, railway station, passengers, railways, signals, rail, rails
	food, raspberry, delicious, homemade	chicken, fattening, cooked, peanut, cream, cookie, house made, bread, biscuit, bakes
	water, river	creek, lake, along, near, river, rocky, treeline, valley, woods, waters
	people, portrait, female, baby, indoor	love, people, posing, girl, young, strangers, pretty, women, happy, life

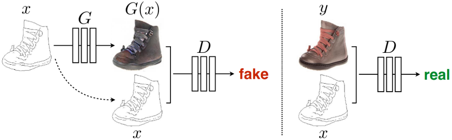
Conditional GAN

image-to-image translation [Isola et al., 2016]



training data

idea: conditional GAN + regression loss



$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

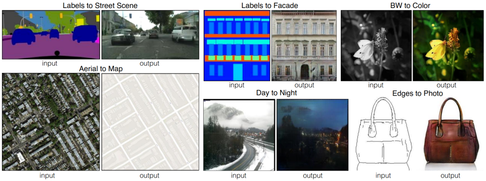
where

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))]$$

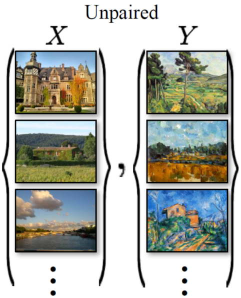
$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1]$$

CycleGAN

image-to-image translation **without** paired data [Zhu et al., 2017]

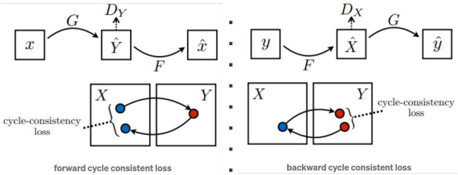


training data



idea: match distributions with cycle consistency

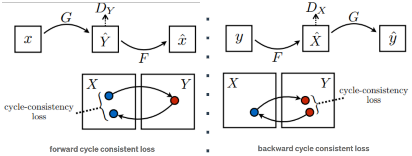
G: forward generator $\quad F$: backward generator



$$x \to G(x) \to F(G(x)) \approx x$$

$$y \to F(y) \to G(F(y)) \approx y$$

More on CycleGAN



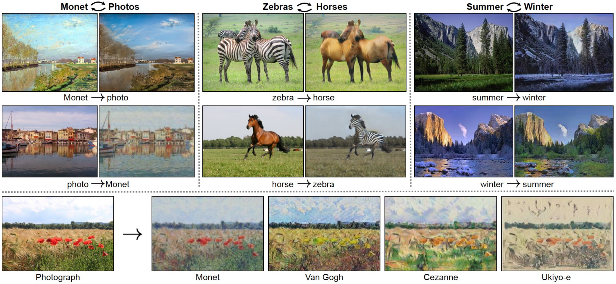
$$x \to G(x) \to F(G(x)) \approx x$$
$$y \to F(y) \to G(F(y)) \approx y$$

two generators $\Longrightarrow$ two GANs

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) =& \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ &+ \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ &+ \lambda \mathcal{L}_{\text{cyc}}(G, F),\end{aligned}$$

where
$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) =& \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] \\ &+ \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].\end{aligned}$$

$\mathcal{L}_{cyc}$ looks familiar? autoencoder!



check out more https://junyanz.github.io/CycleGAN/

Progressive GAN

Toward **high-resolution generation**: progressively grow the generator output (and discriminator input) resolution as training goes on [Karras et al., 2017]

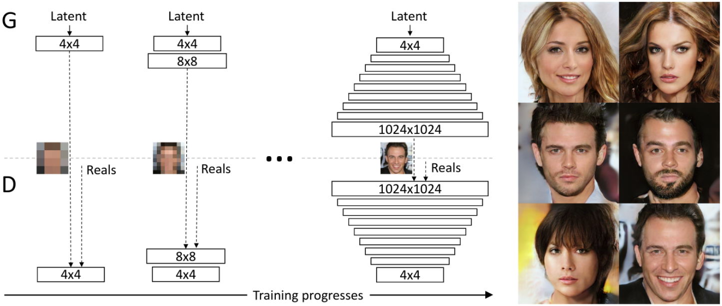


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

BigGAN

Toward **high-resolution generation** [Brock et al., 2018]

LARGE SCALE GAN TRAINING FOR
HIGH FIDELITY NATURAL IMAGE SYNTHESIS

Andrew Brock[*†]
Heriot-Watt University
ajb5@hw.ac.uk

Jeff Donahue[†]
DeepMind
jeffdonahue@google.com

Karen Simonyan[†]
DeepMind
simonyan@google.com

ABSTRACT

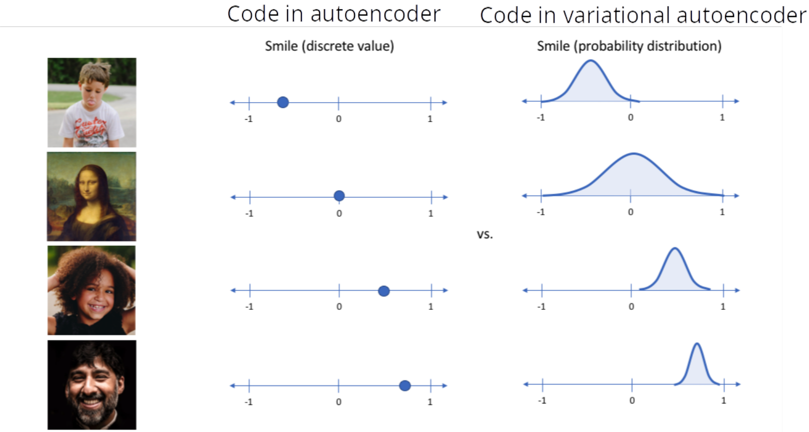
Despite recent progress in generative image modeling, successfully generating high-resolution, diverse samples from complex datasets such as ImageNet remains an elusive goal. To this end, we train Generative Adversarial Networks at the largest scale yet attempted, and study the instabilities specific to such scale. We find that applying orthogonal regularization to the generator renders it amenable to a simple "truncation trick," allowing fine control over the trade-off between sample fidelity and variety by reducing the variance of the Generator's input. Our modifications lead to models which set the new state of the art in class-conditional image synthesis. When trained on ImageNet at 128×128 resolution, our models (BigGANs) achieve an Inception Score (IS) of 166.5 and Fréchet Inception Distance (FID) of 7.4, improving over the previous best IS of 52.52 and FID of 18.65.

1 INTRODUCTION

Figure 1: Class-conditional samples generated by our model.

Outline

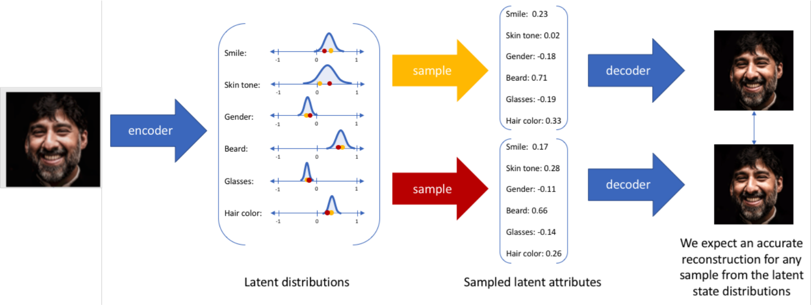
Autoencoder vs. Variational autoencoder



(Credit: https://www.jeremyjordan.me/variational-autoencoders/)

- Autoencoder maps each input to a **deterministic vector**
- Variational Autoencoder maps each input to a (parameterized) **distribution**

Variational autoencoder



(Credit: https://www.jeremyjordan.me/variational-autoencoders/)

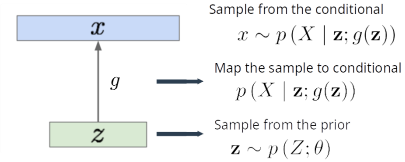
- **smoothness**: nearby codes tend to produce very similar reconstructions

Bayesian generation: to draw an $\boldsymbol{x}$ from $p(X, Z)$, where $p(X, Z) = p(X|Z)p(Z)$

- Step 1: draw a $\boldsymbol{z} \sim p(Z)$
- Step 2: draw an $\boldsymbol{x} \sim p(X|\boldsymbol{z})$

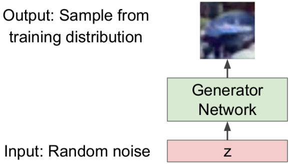
Variational AE (VAE)

Model the generation via an **explicit density**



- $p(Z; \boldsymbol{\theta})$, $p(X|Z; \boldsymbol{\theta})$ are easy to sample from, e.g., $\mathrm{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- the nonlinear mapping $g(\boldsymbol{z})$ allows expressive form of $p(X \mid \boldsymbol{z}; g(\boldsymbol{z}))$

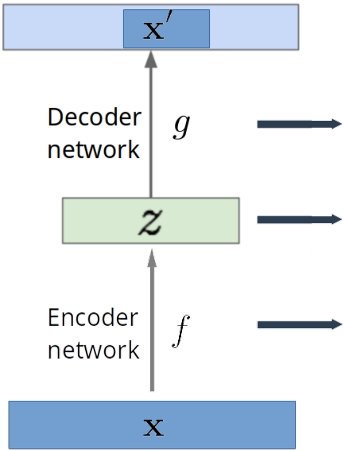
GANs



(Credit: Stanford CS231N)

- **task**: given $\{\boldsymbol{x}_i\}$, generate new samples from the distribution
- **idea**: map samples from a **simple** distribution to samples from the training distribution
- **how to**: measure the difference between **mapped/training** distributions (JS, W-dist, etc)

Augmenting the encoder



Sample from the conditional

$$\mathbf{x}' \sim p\left(X \mid \mathbf{z}; g(\mathbf{z})\right)$$

Map the sample to conditional

$$p\left(X \mid \mathbf{z}; g(\mathbf{z})\right)$$

Sample from the approx. posterior

$$\mathbf{z} \sim q\left(Z \mid \mathbf{x}; f(\mathbf{x})\right)$$

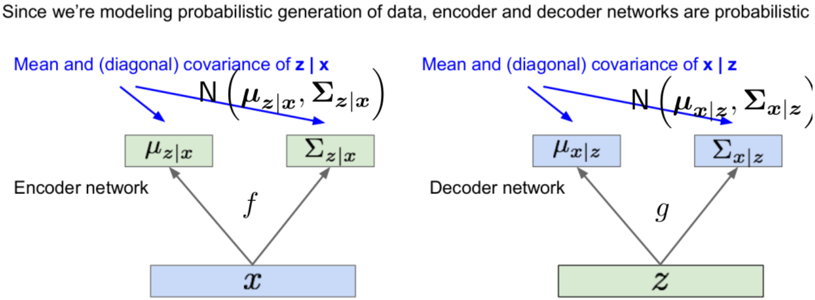
Map the sample to approx. posterior

$$q\left(Z \mid \mathbf{x}; f(\mathbf{x})\right) \approx p\left(Z \mid \mathbf{x}\right)$$

$$\mathbf{x} \approx \mathbf{x}'$$

A VAE example

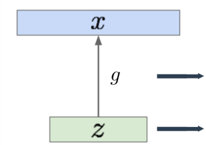
assume both $p(X \mid \boldsymbol{z})$ and $q\left(Z \mid \boldsymbol{x}\right)$ are multivariate Gaussian, i.e., $\mathsf{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$



(Credit: Stanford CS231N)

How to train it?

Model the generation via an **explicit density**



Sample from the conditional
$x \sim p\left(X \mid \mathbf{z}; g(\mathbf{z})\right)$

Map the sample to conditional
$p\left(X \mid \mathbf{z}; g(\mathbf{z})\right)$

Sample from the prior
$\mathbf{z} \sim p\left(Z; \theta\right)$

- $p(Z; \boldsymbol{\theta})$, $p(X|Z; \boldsymbol{\theta})$ are easy to sample from, e.g., $\mathrm{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$
- the nonlinear mapping $g\left(\boldsymbol{z}\right)$ allows expressive form of $p\left(X \mid \boldsymbol{z}; g\left(\boldsymbol{z}\right)\right)$

Assume m training samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$

$$\max_{\boldsymbol{\theta}, g} \prod_{i=1}^{m} p\left(\boldsymbol{x}_i; \boldsymbol{\theta}, g\right) \Longleftrightarrow \max_{\boldsymbol{\theta}, g} \sum_{i=1}^{m} \log p\left(\boldsymbol{x}_i; \boldsymbol{\theta}, g\right)$$

$$\Longleftrightarrow \max_{\boldsymbol{\theta}, g} \sum_{i=1}^{m} \log \int_{\boldsymbol{z}} p\left(\boldsymbol{x}_i \mid \boldsymbol{z}; g\left(\boldsymbol{z}\right)\right) p\left(\boldsymbol{z}; \boldsymbol{\theta}\right) \, d\boldsymbol{z}$$

(likely) intractable due to the integral

Monte Carlo sampling approx.? sample $\boldsymbol{z}_j$'s iid from $p\left(\boldsymbol{z}; \boldsymbol{\theta}\right)$

$\int_{\boldsymbol{z}} p\left(\boldsymbol{x} \mid \boldsymbol{z}; g\left(\boldsymbol{z}\right)\right) p\left(\boldsymbol{z}; \boldsymbol{\theta}\right) \, d\boldsymbol{z} \approx \frac{1}{K} \sum_{j=k}^{K} p\left(\boldsymbol{x} \mid \boldsymbol{z}_j; g\left(\boldsymbol{z}_j\right)\right)$—expensive and hard to converge

Get around

$\boldsymbol{x}$—observation, $\boldsymbol{z}$—latent variable

- $p(\boldsymbol{z})$—prior
- $p(\boldsymbol{x} \mid \boldsymbol{z})$—likelihood/conditional
- $p(\boldsymbol{z} \mid \boldsymbol{x})$—posterior

want:

$$\max_{\boldsymbol{\theta}, g} \sum_{i=1}^{m} \log p\left(\boldsymbol{x}_i; \boldsymbol{\theta}, g\right)$$

Definition (Evidence lower bound (ELBO))

$\log p\left(\boldsymbol{x}\right) \geq \mathcal{L}\left(\boldsymbol{x}; q\right) \doteq \log p\left(\boldsymbol{x}\right) - D_{\mathsf{KL}}\left(q\left(Z \mid \boldsymbol{x}\right) \| p\left(Z \mid \boldsymbol{x}\right)\right)$ for any probability distribution q over Z (remember $D_{\mathsf{KL}}(\cdot|\cdot) \geq 0$)

variational inference: find q so that the lower bound is tight as possible (i.e., $q\left(Z \mid \boldsymbol{x}\right) \approx p\left(Z \mid \boldsymbol{x}\right)$ for all $\boldsymbol{x}$) but remains **tractable**

idea: restrict to a parameterized family $q\left(Z \mid \boldsymbol{x}; f\left(\boldsymbol{x}\right)\right)$

(lots of other ideas in Bayesian inference, e.g., mean-field approximation. See, e.g., Chapter 19 of [Goodfellow et al., 2017])

How to train it?

need another identity

$$\log p(\boldsymbol{x}) - \mathsf{KL}\left(q(Z \mid \boldsymbol{x}) \| p(Z \mid \boldsymbol{x})\right) = \mathbb{E}_{\boldsymbol{z}\sim q} \log p(\boldsymbol{x} \mid \boldsymbol{z}) - D_{\mathsf{KL}}\left(q(Z \mid \boldsymbol{x}) \| p(Z)\right)$$

maximize the right side instead

- maximizing $-D_{\mathsf{KL}}\left(q(Z \mid \boldsymbol{x}) \| p(Z)\right)$ ensures $q(Z \mid \boldsymbol{x})$ close to the prior $p(Z)$
- maximizing $\mathbb{E}_{\boldsymbol{z}\sim q} \log p(\boldsymbol{x} \mid \boldsymbol{z})$ maximizes the likelihood of reproducing $\boldsymbol{x}$—minimizing reconstruction error
- overall, maximizing a lower bound to maximize the original

overall objective:

$$\boxed{\max_{g,f} \sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{z}\sim q(Z \mid \boldsymbol{x}_i; f(\boldsymbol{x}_i))} \log p\left(\boldsymbol{x}_i \mid \boldsymbol{z}; g(\boldsymbol{x}_i)\right) - D_{\mathsf{KL}}\left(q\left(Z \mid \boldsymbol{x}_i; f(\boldsymbol{x}_i)\right) \| p(Z)\right)}$$

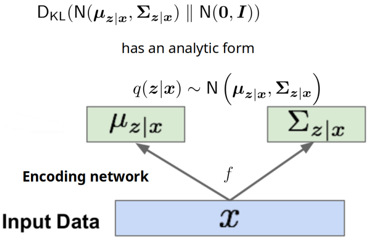
How to train it?

overall objective:

$$\max_{g,f} \sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{z}\sim q(Z \mid \boldsymbol{x}_i; f(\boldsymbol{x}_i))} \log p\left(\boldsymbol{x}_i \mid \boldsymbol{z}; g\left(\boldsymbol{x}_i\right)\right) - D_{\mathsf{KL}}\left(q\left(Z \mid \boldsymbol{x}_i; f\left(\boldsymbol{x}_i\right)\right) \| p\left(Z\right)\right)$$

set $p\left(Z\right) \sim \mathsf{N}\left(\mathbf{0}, \boldsymbol{I}\right)$

build the computational graph for a single sample:



(Credit: adapted from Stanford CS231N)

- although choice of N is simplistic, the nonlinear mapping is powerful
- all operations are differentiable so far

How to train it?

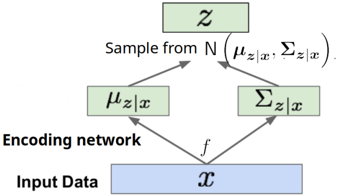
overall objective:

$$\max_{g,f} \sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{z} \sim q(Z \mid \boldsymbol{x}_i; f(\boldsymbol{x}_i))} \log p\left(\boldsymbol{x}_i \mid \boldsymbol{z}; g\left(\boldsymbol{x}_i\right)\right) - D_{\mathrm{KL}}\left(q\left(Z \mid \boldsymbol{x}_i; f\left(\boldsymbol{x}_i\right)\right) \| p\left(Z\right)\right)$$

set $p(Z) \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})$

build the computational graph for a single sample:

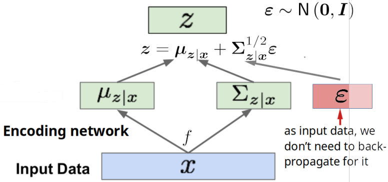
$(\boldsymbol{\mu}_{\boldsymbol{z}|\boldsymbol{x}}, \boldsymbol{\Sigma}_{\boldsymbol{z}|\boldsymbol{x}}) \mapsto \boldsymbol{z}$ is a random function and hence not differentiable



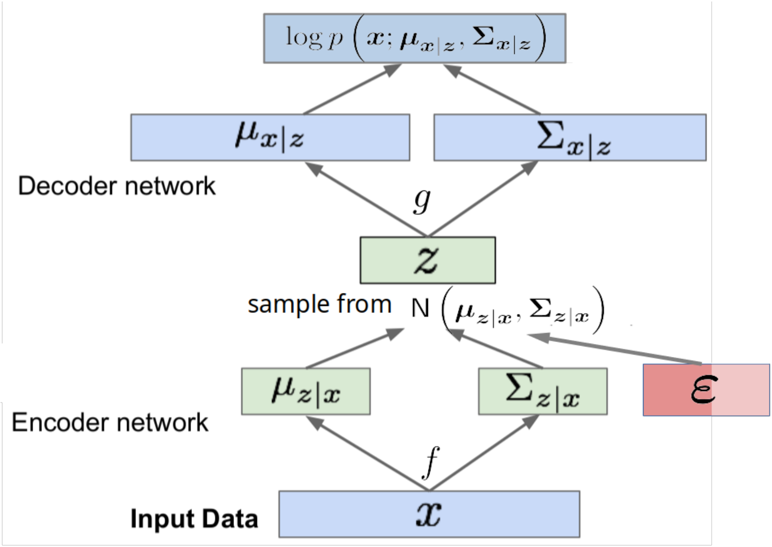
(Credit: adapted from Stanford CS231N)

Reparametrization trick

$(\boldsymbol{\mu}_{\boldsymbol{z}|\boldsymbol{x}}, \boldsymbol{\Sigma}_{\boldsymbol{z}|\boldsymbol{x}}) \mapsto \boldsymbol{z}$ is a deterministic and differentiable function for any ε

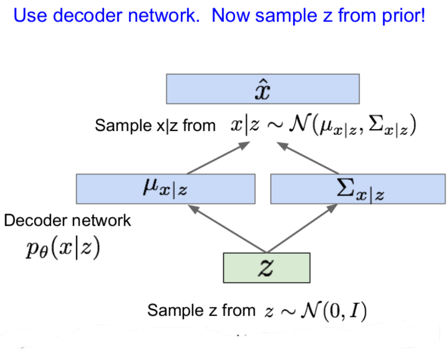


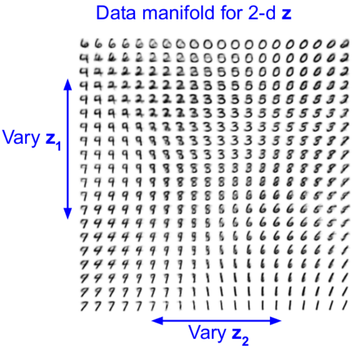
(Credit: adapted from Stanford CS231N)



(Credit: adapted from Stanford CS231N)

Generate new samples





(Credit: Stanford CS231N)

the coordinates of the codes potentially correspond to different physical properties (due to the diagonal covariance prior)

VAE objective:

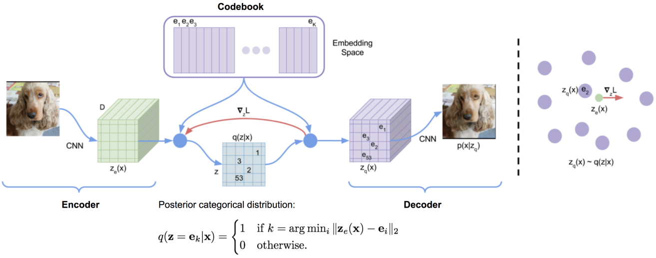
$$\max_{g,f} \sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{z} \sim q(Z \mid \boldsymbol{x}_i; f(\boldsymbol{x}_i))} \log p\left(\boldsymbol{x}_i \mid \boldsymbol{z}; g\left(\boldsymbol{x}_i\right)\right) - D_{\mathsf{KL}}\left(q\left(Z \mid \boldsymbol{x}_i; f\left(\boldsymbol{x}_i\right)\right) \| p\left(Z\right)\right)$$

β**-VAE objective** [Higgins et al., 2017]:

$$\max_{g,f} \sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{z} \sim q(Z \mid \boldsymbol{x}_i; f(\boldsymbol{x}_i))} \log p\left(\boldsymbol{x}_i \mid \boldsymbol{z}; g\left(\boldsymbol{x}_i\right)\right) - \beta D_{\mathsf{KL}}\left(q\left(Z \mid \boldsymbol{x}_i; f\left(\boldsymbol{x}_i\right)\right) \| p\left(Z\right)\right)$$

$\beta > 1$ to put more emphasis on the similarity of $q\left(Z \mid \boldsymbol{x}_i; f\left(\boldsymbol{x}_i\right)\right)$ and $p\left(Z\right) \Longrightarrow$ diagonal covariance of $p(Z)$ encourages decorrelation of coordinates in Z—disentangled representation

Vector quantization (VQ)-VAE



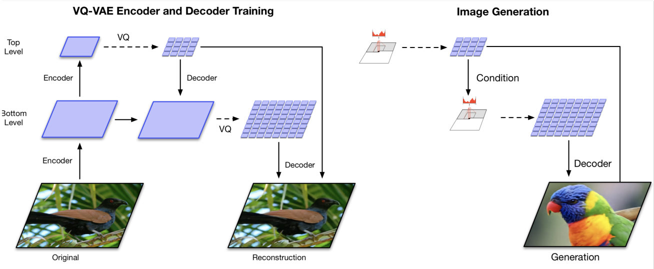
(Credit: [Van Den Oord et al., 2017])

- Finitely many latent codes, and hence discrete distribution
- Code assignment via nearest neighbor search
- Training via minimizing

$$L = \underbrace{\|\boldsymbol{x} - D(\boldsymbol{e}_k)\|_2^2}_{\text{reconstruction loss}} + \underbrace{\|\mathrm{sg}[E(\boldsymbol{x})] - \boldsymbol{e}_k\|_2^2}_{\text{VQ loss}} + \underbrace{\beta\|E(\boldsymbol{x}) - \mathrm{sg}[\boldsymbol{e}_k]\|_2^2}_{\text{commitment loss}}$$

  where sg is the `stop_gradient` operator
- Plus an auto-regressive prior

VQ-VAE-2



(Credit: [Razavi et al., 2019])

– Hierarchical VQ-VAE
– A prior learned on the discrete codebook
– In combination with self-attention enhanced autoregressive prior

Proof of the key equality

we'll omit the density probability for simplicity

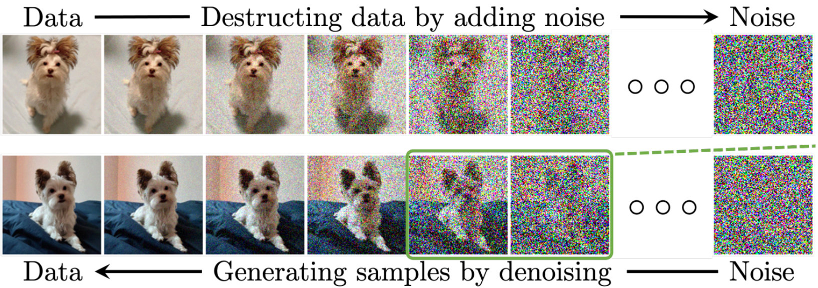
$$
\begin{aligned}
& D_{\mathsf{KL}}(q(Z|\boldsymbol{x})\|p(Z|\boldsymbol{x})) \\
&= \int q(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&= \int q(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{z}, \boldsymbol{x})} d\boldsymbol{z} \qquad \text{Because } p(z|x){=}p(z,x)/p(x) \\
&= \int q(\boldsymbol{z}|\boldsymbol{x}) \left(\log p(\boldsymbol{x}) + \log \frac{q(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}, \boldsymbol{x})} \right) d\boldsymbol{z} \\
&= \log p(\boldsymbol{x}) + \int q(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}, \boldsymbol{x})} d\boldsymbol{z} \qquad \text{Because } \int q(z|x)dz{=}1 \\
&= \log p(\boldsymbol{x}) + \int q(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})} d\boldsymbol{z} \qquad \text{Because } p(z,x){=}p(x|z)p(z) \\
&= \log p(\boldsymbol{x}) + D_{\mathsf{KL}}(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})) - \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \log p(\boldsymbol{x}|\boldsymbol{z}).
\end{aligned}
$$

So

$$
\log p(\boldsymbol{x}) - D_{\mathsf{KL}}(q(Z|\boldsymbol{x})\|p(Z|\boldsymbol{x})) = \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \log p(\boldsymbol{x}|\boldsymbol{z}) - D_{\mathsf{KL}}(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})).
$$

Outline

Image to noise, and noise back to image



(Credit: [Yang et al., 2022])

Forward diffusion: image $\longrightarrow$ noise
Reverse diffusion: noise $\longrightarrow$ image

Forward diffusion

Start from an image $\boldsymbol{x}_0$ and $\alpha_i \in (0,1)$ for all i

- Step 1: $\boldsymbol{x}_1 = \sqrt{\alpha_1}\boldsymbol{x}_0 + \sqrt{1-\alpha_1}\boldsymbol{\varepsilon}_1$ where $\varepsilon_1 \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})$
- Step 2: $\boldsymbol{x}_2 = \sqrt{\alpha_2}\boldsymbol{x}_1 + \sqrt{1-\alpha_2}\boldsymbol{\varepsilon}_2$ where $\varepsilon_2 \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})$
- $\cdots$
- Step T: $\boldsymbol{x}_T = \sqrt{\alpha_T}\boldsymbol{x}_{T-1} + \sqrt{1-\alpha_T}\boldsymbol{\varepsilon}_T$ where $\varepsilon_T \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})$

Now we have

$$\begin{aligned}
\boldsymbol{x}_T &= \sqrt{\alpha_T}\boldsymbol{x}_{T-1} + \sqrt{1-\alpha_T}\boldsymbol{\varepsilon}_T \\
&= \sqrt{\alpha_T}\left(\sqrt{\alpha_{T-1}}\boldsymbol{x}_{T-2} + \sqrt{1-\alpha_{T-1}}\boldsymbol{\varepsilon}_{T-1}\right) + \sqrt{1-\alpha_T}\boldsymbol{\varepsilon}_T \\
&= \sqrt{\alpha_T\alpha_{T-1}}\boldsymbol{x}_{T-2} + \sqrt{1-\alpha_T\alpha_{T-1}}\boldsymbol{\varepsilon} \text{ where } \boldsymbol{\varepsilon} \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})
\end{aligned}$$

Keep the induction, we obtain

$$\boxed{\boldsymbol{x}_T = \sqrt{\prod_{t=1}^{T}\alpha_t}\ \boldsymbol{x}_0 + \sqrt{1-\prod_{t=1}^{T}\alpha_t}\ \boldsymbol{\varepsilon} \quad \text{where } \boldsymbol{\varepsilon} \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})}$$

typically $\alpha_1 > \alpha_2 > \cdots > \alpha_T$, and so $\boldsymbol{x}_T \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})$ as $T \to \infty$

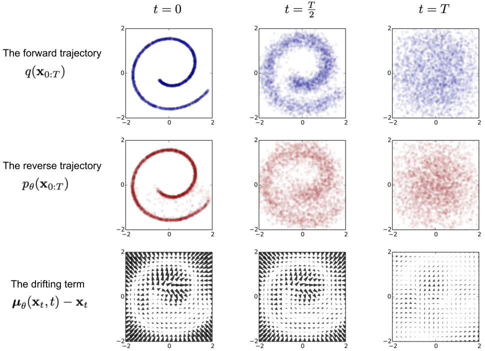
Reverse diffusion

Assume the true prob. density in the forward diffusion $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$. If we also know $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, we can reverse the process

Solution via simplification:

$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ is approximated by $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \doteq \mathsf{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t))$, where $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ are learnable function parameterized by DNNs

- Recall VAE?
- Training: minimize $\mathrm{dist}(q(\boldsymbol{x}_0), p_{\boldsymbol{\theta}}(\boldsymbol{x}_0))$



(Credit: [Sohl-Dickstein et al., 2015])

Training

minimize $\mathrm{dist}(q(\boldsymbol{x}_0), p_{\boldsymbol{\theta}}(\boldsymbol{x}_0))$. If we can cross-entropy loss, then

$$\begin{aligned}
L_{\text{CE}} &= -\mathbb{E}_{q(\boldsymbol{x}_0)} \log p_\theta(\boldsymbol{x}_0) \\
&= -\mathbb{E}_{q(\boldsymbol{x}_0)} \log \Big(\int p_\theta(\boldsymbol{x}_{0:T}) d\boldsymbol{x}_{1:T} \Big) \\
&= -\mathbb{E}_{q(\boldsymbol{x}_0)} \log \Big(\int q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0) \frac{p_\theta(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} d\boldsymbol{x}_{1:T} \Big) \\
&= -\mathbb{E}_{q(\boldsymbol{x}_0)} \log \Big(\mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \frac{p_\theta(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \Big) \\
&\leq -\mathbb{E}_{q(\boldsymbol{x}_{0:T})} \log \frac{p_\theta(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \\
&= \mathbb{E}_{q(\boldsymbol{x}_{0:T})} \Big[\log \frac{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}{p_\theta(\boldsymbol{x}_{0:T})} \Big] = L_{\text{VLB}}
\end{aligned}$$

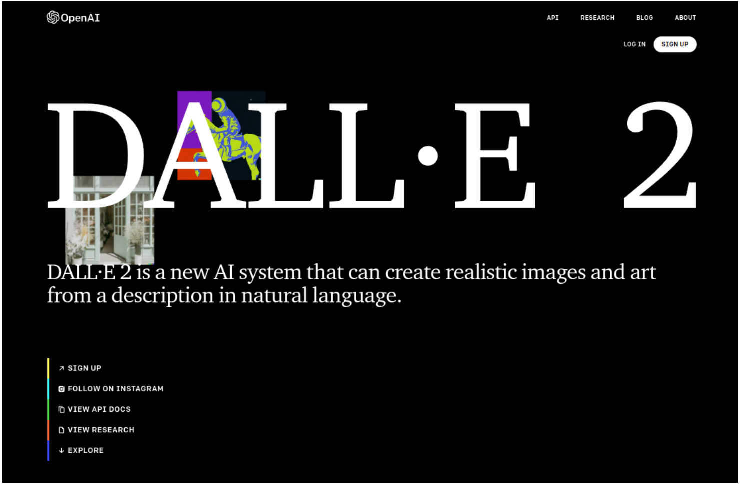
where VLB means evidence lower bound

Training

After further rearrangement,

$$
\begin{aligned}
L_{\mathsf{VLB}} &= \mathbb{E}_{q(\boldsymbol{x}_{0:T})}\Big[\log \frac{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}{p_\theta(\boldsymbol{x}_{0:T})}\Big] \\
&= \mathbb{E}_q[\underbrace{D_{\mathsf{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_0) \parallel p_\theta(\boldsymbol{x}_T))}_{L_T\text{:constant}} + \\
&\qquad \sum_{t=2}^{T} \underbrace{D_{\mathsf{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \parallel p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))}_{L_{t-1}\text{: KL between Gaussians}} \underbrace{- \log p_\theta(\boldsymbol{x}_0|\boldsymbol{x}_1)}_{L_0}]
\end{aligned}
$$

Diffusion models (and other models in action)



https://openai.com/dall-e-2/

Diffusion models (and other models in action)

vibrant portrait painting of Salvador Dalí with a robotic half face

a shiba inu wearing a beret and black turtleneck

a close up of a handpalm with leaves growing from it

an espresso machine that makes coffee from human souls, artstation

panda mad scientist mixing sparkling chemicals, artstation

a corgi's head depicted as an explosion of a nebula

https://openai.com/dall-e-2/

Diffusion models (and other models in action)

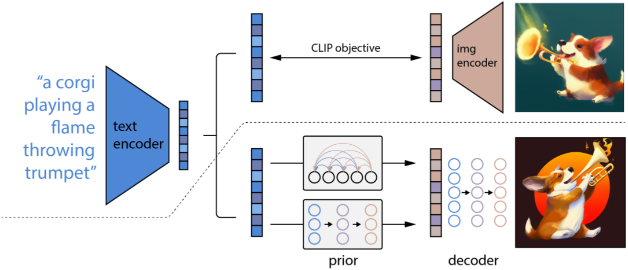
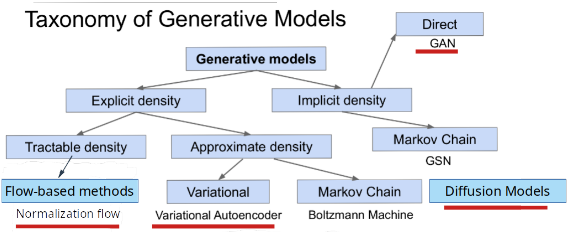


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

https://openai.com/dall-e-2/

Outline

Modeling density directly



(Credit: adapted from Stanford CS231N slides)

Classical ideas:

- (Gaussian) Mixture models
- non-parametric methods, e.g., kernel density estimation

Change of variable theorem in probability

Consider a scalar random variable Z and its density $\pi(z)$, what's the density of $x = f(z)$, suppose f is invertible?

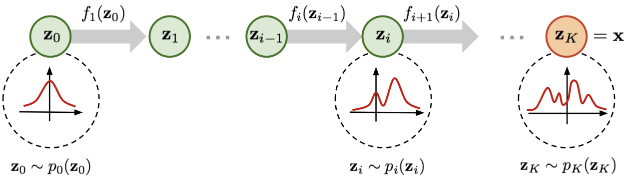
$$p(x) = \pi(z)\left|\frac{dz}{dx}\right| = \pi(f^{-1}(x))\left|\frac{df^{-1}}{dx}\right|$$

The multivariate version

$$\boldsymbol{z} \sim \pi(\boldsymbol{z}), \boldsymbol{x} = f(\boldsymbol{z}), \boldsymbol{z} = f^{-1}(\boldsymbol{x})$$
$$p(\boldsymbol{x}) = \pi(\boldsymbol{z})\left|\det \frac{d\boldsymbol{z}}{d\boldsymbol{x}}\right| = \pi(f^{-1}(\boldsymbol{x}))\left|\det \frac{df^{-1}}{d\boldsymbol{x}}\right|$$

where $\frac{df^{-1}}{d\boldsymbol{x}}$ is the Jacobian of the inverse function f^{-1} wrt $\boldsymbol{x}$

Normalization flow: idea



(Credit: https://lilianweng.github.io/)

$\boldsymbol{z}_{i-1} \sim p_{i-1}(\boldsymbol{z}_{i-1}),\ \boldsymbol{z}_i = f_i(\boldsymbol{z}_{i-1})$, thus $\boldsymbol{z}_{i-1} = f_i^{-1}(\boldsymbol{z}_i)$

$$
\begin{aligned}
p_i(\boldsymbol{z}_i) &= p_{i-1}(f_i^{-1}(\boldsymbol{z}_i)) \left| \det \frac{df_i^{-1}}{d\boldsymbol{z}_i} \right| \\
&= p_{i-1}(\boldsymbol{z}_{i-1}) \left| \det \left(\frac{df_i}{d\boldsymbol{z}_{i-1}} \right)^{-1} \right| && \text{According to the inverse func theorem.} \\
&= p_{i-1}(\boldsymbol{z}_{i-1}) \left| \det \frac{df_i}{d\boldsymbol{z}_{i-1}} \right|^{-1} && \text{According to a property of Jacobians of invertible func.} \\
&= \ldots \\
&= p_0(\boldsymbol{z}_0) \prod_{j=1}^{i} \left| \det \frac{df_j}{d\boldsymbol{z}_{j-1}} \right|^{-1}
\end{aligned}
$$

Normalization flow: ideas

$$p_i(\boldsymbol{z}_i) = p_0(\boldsymbol{z}_0) \prod_{j=1}^{i} \left| \det \frac{df_j}{d\boldsymbol{z}_{j-1}} \right|^{-1} \Longrightarrow p(\boldsymbol{x}) = p_K(\boldsymbol{z}_K) = p_0(\boldsymbol{z}_0) \prod_{j=1}^{K} \left| \det \frac{df_j}{d\boldsymbol{z}_{j-1}} \right|^{-1}$$

$$\Longrightarrow \log p(\boldsymbol{x}) = \log p_0(\boldsymbol{z}_0) - \sum_{j=1}^{K} \log \left| \det \frac{df_j}{d\boldsymbol{z}_{j-1}} \right|$$

So we can do maximum likelihood inference directly:

$$\max_{\boldsymbol{\theta}} \ \frac{1}{N} \sum_{\ell=1}^{N} \log p(\boldsymbol{x}_\ell)$$

f's are parametrized by DNNs with shared weights

Key challenges:

- ensure that f is invertible so that $\det \dfrac{df_j}{d\boldsymbol{z}_{j-1}}$ does not vanish
- computational tractability due to $\det$

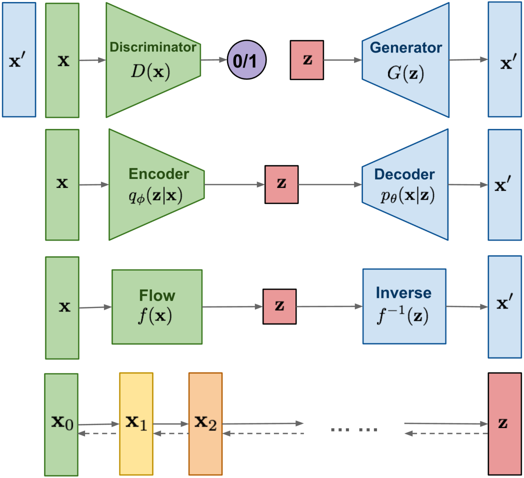
Summary of generative models

GAN: Adversarial training

VAE: maximize variational lower bound

Flow-based models: Invertible transform of distributions

Diffusion models: Gradually add Gaussian noise and then reverse



(Credit: https://lilianweng.github.io/)

Outline

Adversarial generative network (GAN)

Variational autoencoder (VAE)

Diffusion models

Normalization flow

Suggested reading

Suggested reading

- CVPR 2018 tutorial on GANs CVPR2018TutorialonGANs
- NIPS 2016 Tutorial:Generative Adversarial Networks https://arxiv.org/abs/1701.00160
- An Introduction to Variational Autoencoders [Kingma and Welling, 2019]
- Normalizing Flows for Probabilistic Modeling and Inference https://arxiv.org/abs/1912.02762
- From GAN to WGAN https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html
- From Autoencoder to Beta-VAE https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html
- Flow-based Deep Generative Models https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html#types-of-generative-models
- What are Diffusion Models? https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

References i

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). **Wasserstein gan.** *arXiv:1701.07875*.

[Brock et al., 2018] Brock, A., Donahue, J., and Simonyan, K. (2018). **Large scale gan training for high fidelity natural image synthesis.** *arXiv:1809.11096*.

[Goodfellow et al., 2017] Goodfellow, I., Bengio, Y., and Courville, A. (2017). **Deep Learning.** The MIT Press.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). **Generative adversarial networks.** *arXiv:1406.2661*.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). **Improved training of wasserstein gans.** *arXiv:1704.00028*.

[Higgins et al., 2017] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). **beta-vae: Learning basic visual concepts with a constrained variational framework.** In *International Conference on Learning Representations, ICLR 2017*.

References ii

[Isola et al., 2016] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2016). **Image-to-image translation with conditional adversarial networks.** *arXiv:1611.07004.*

[Karras et al., 2017] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). **Progressive growing of gans for improved quality, stability, and variation.** *arXiv:1710.10196.*

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). **Auto-encoding variational bayes.** *arXiv:1312.6114.*

[Kingma and Welling, 2019] Kingma, D. P. and Welling, M. (2019). **An introduction to variational autoencoders.** *arXiv:1906.02691.*

[Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). **Conditional generative adversarial nets.** *arXiv:1411.1784.*

[Papamakarios et al., 2019] Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2019). **Normalizing flows for probabilistic modeling and inference.** *Journal of Machine Learning Research, 22(57):1-64, 2021.*

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). **Unsupervised representation learning with deep convolutional generative adversarial networks.** *arXiv:1511.06434*.

[Razavi et al., 2019] Razavi, A., van den Oord, A., and Vinyals, O. (2019). **Generating diverse high-fidelity images with vq-vae-2.** *arXiv:1906.00446*.

[Razaviyayn et al., 2020] Razaviyayn, M., Huang, T., Lu, S., Nouiehed, M., Sanjabi, M., and Hong, M. (2020). **Non-convex min-max optimization: Applications, challenges, and recent theoretical advances.** *arXiv:2006.08141*.

[Rezende and Mohamed, 2015] Rezende, D. J. and Mohamed, S. (2015). **Variational inference with normalizing flows.** *arXiv:1505.05770*.

[Sohl-Dickstein et al., 2015] Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). **Deep unsupervised learning using nonequilibrium thermodynamics.** *arXiv:1503.03585*.

[Srivastava et al., 2017] Srivastava, A., Valkov, L., Russell, C., Gutmann, M. U., and Sutton, C. (2017). **Veegan: Reducing mode collapse in gans using implicit variational learning.** *arXiv:1705.07761*.

[Van Den Oord et al., 2017] Van Den Oord, A., Vinyals, O., et al. (2017). **Neural discrete representation learning.** *Advances in neural information processing systems*, 30.

[Yang et al., 2022] Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Shao, Y., Zhang, W., Cui, B., and Yang, M.-H. (2022). **Diffusion models: A comprehensive survey of methods and applications.** *arXiv:2209.00796*.

[Zhu et al., 2017] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). **Unpaired image-to-image translation using cycle-consistent adversarial networks.** *arXiv:1703.10593*.