Deep Learning for Computer Vision

Fall 2022

https://cool.ntu.edu.tw/courses/189345 (NTU COOL)

http://vllab.ee.ntu.edu.tw/dlcv.html (Public website)

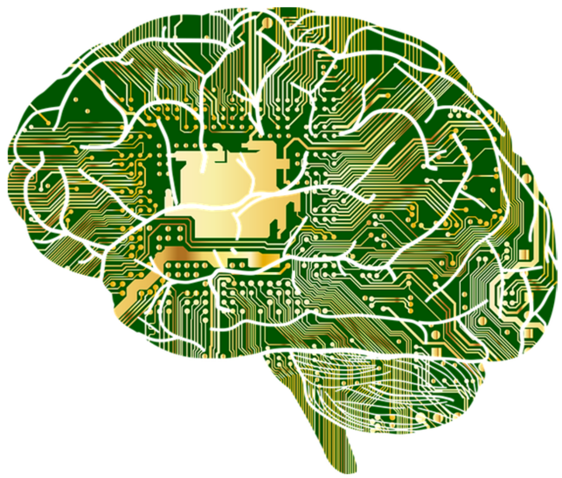
Yu-Chiang Frank Wang 王鈺強, Professor

Dept. Electrical Engineering, National Taiwan University

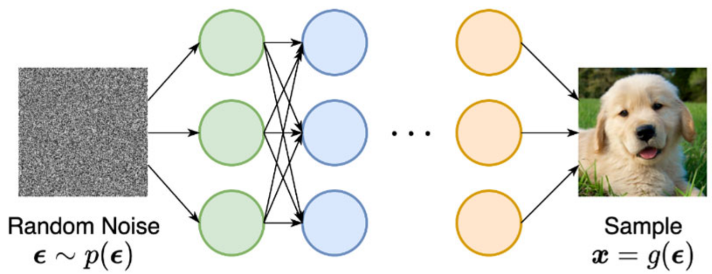
2022/10/4

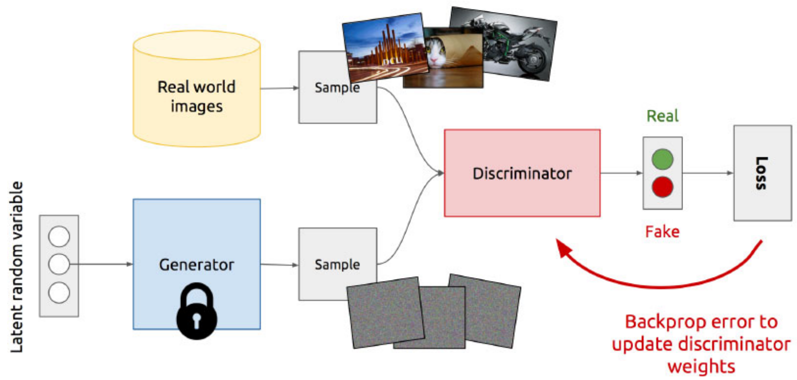
What's to Be Covered Today…

- Generative Models
  - Auto-Encoder vs. Variational Auto-Encoder
  - Generative Adversarial Network (GAN)
  - Diffusion Model



- Unfortunately, lots of equations today…
  I will try to make today's lecture as painless as possible!



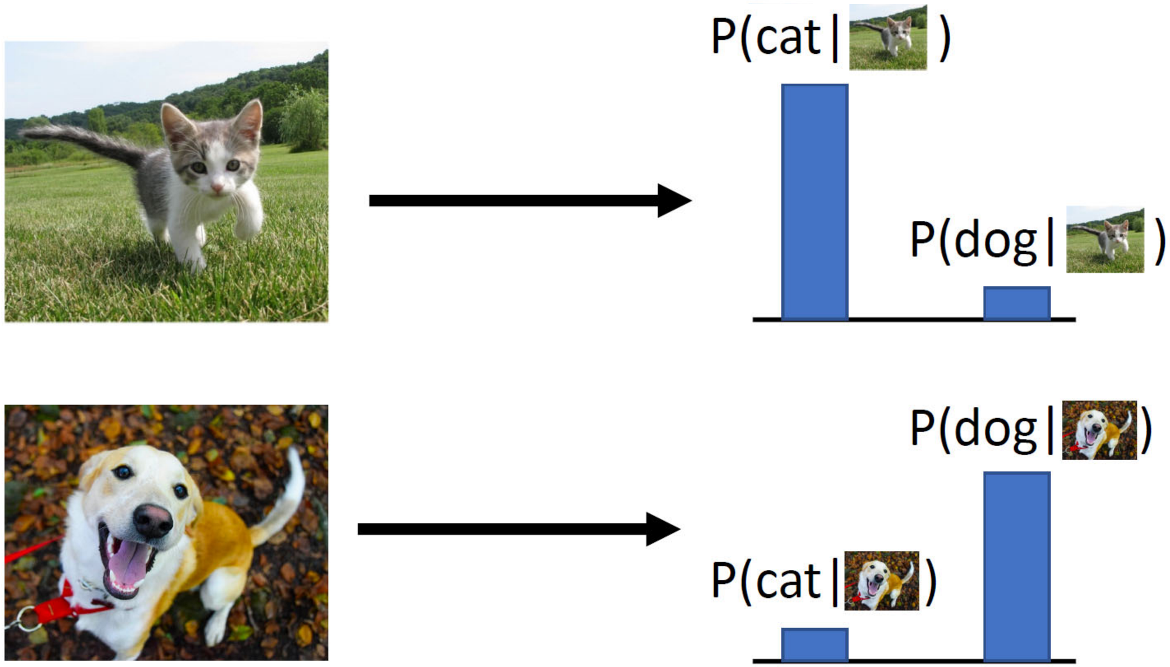


Discriminative vs. Generative Models

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

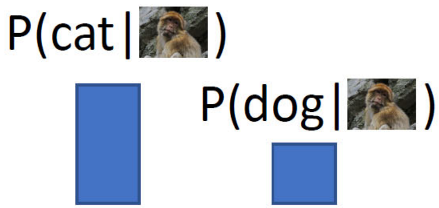
Conditional Generative Model: Learn p(x|y)



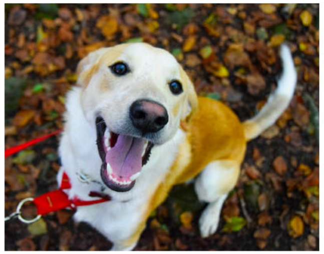
Discriminative model: the possible labels for each input ”compete” for probability mass. But no competition between **images**

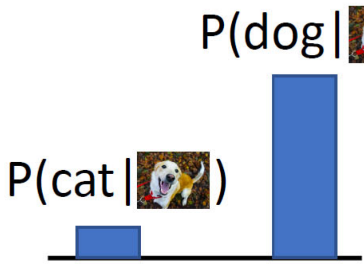
Discriminative vs. Generative Models (cont'd)

Discriminative Model: Learn a probability distribution p(y|x)



Generative Model: Learn a probability distribution p(x)





Conditional Generative Model: Learn p(x|y)

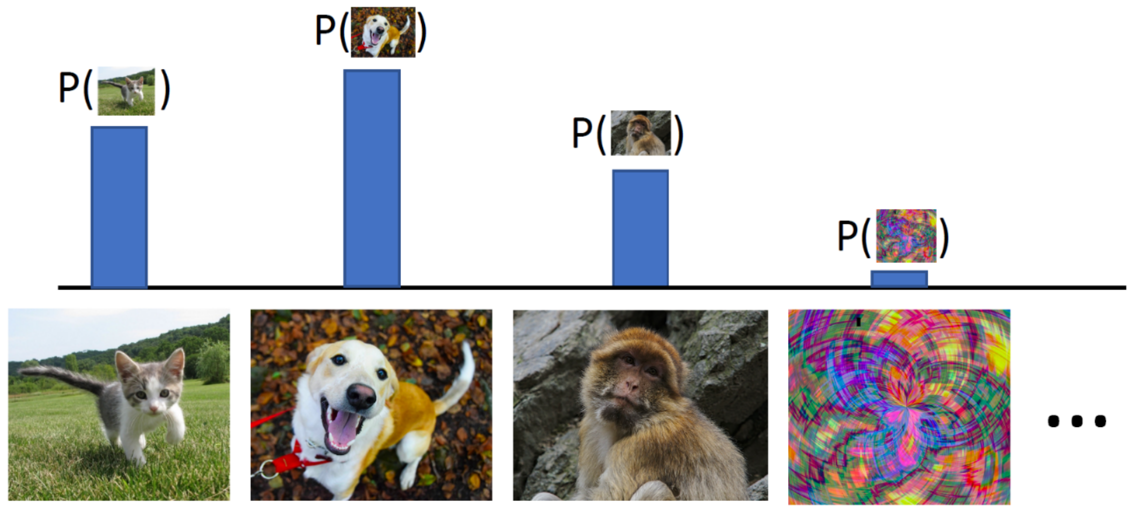
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs. Generative Models (cont'd)

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Generative model: All possible images compete with each other for probability mass

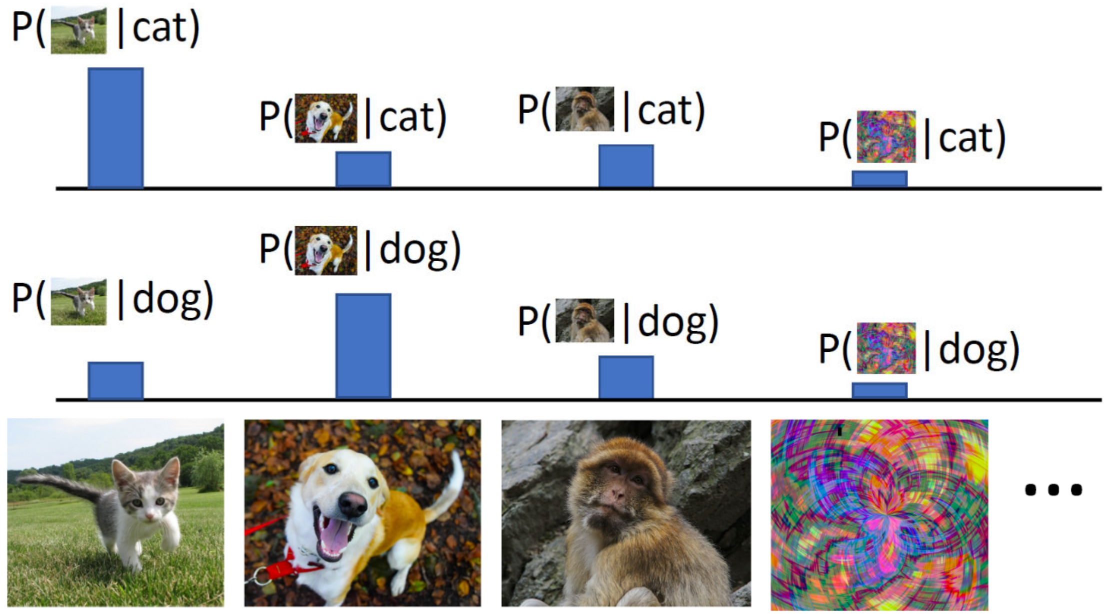
Model can "reject" unreasonable inputs by assigning them small values

Discriminative vs. Generative Models (cont'd)

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Conditional Generative Model: Each possible label induces a competition among all images

Discriminative vs. Generative Models (cont'd)

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

Discriminative Model

(Unconditional) Generative Model

$$\boxed{P(x \mid y)} = \frac{\boxed{P(y \mid x)}}{\boxed{P(y)}} \boxed{P(x)}$$

Conditional Generative Model

Prior over labels

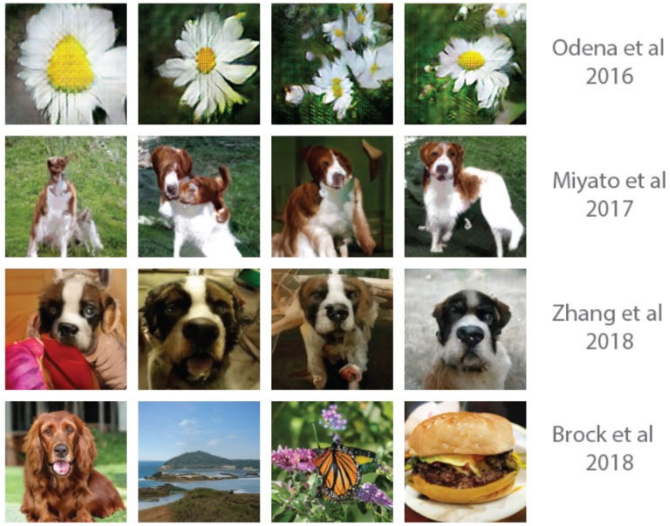
We can build a conditional generative model from other components!

Additional Remarks

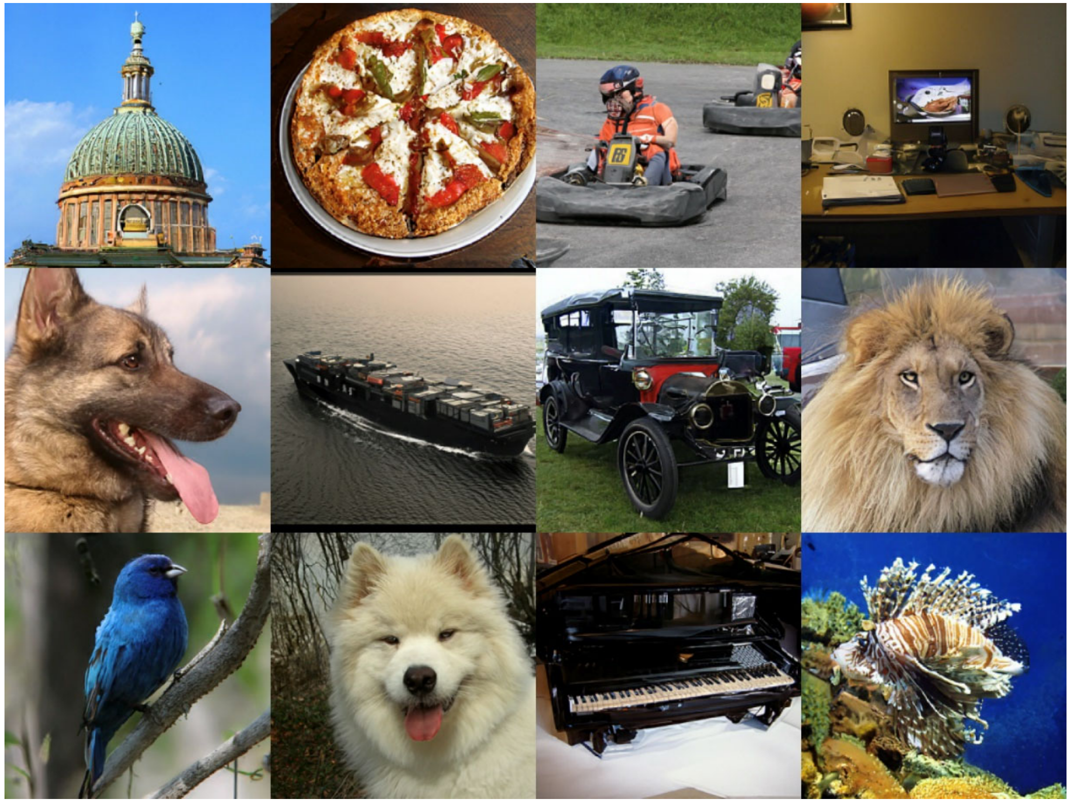
- Discriminative Models
  - Learn a (posterior)probability distribution p(y|x)
  - Assign labels to each instance x
  - Supervised learning
- Generative Models
  - Learn a probability distribution p(x)
  - Data representation, detect outliers, etc.
  - Unsupervised learning

What Have Been Done
Using Deep Generative Models?

- Progress on synthesizing images (ImageNet)



(Odena 2018)

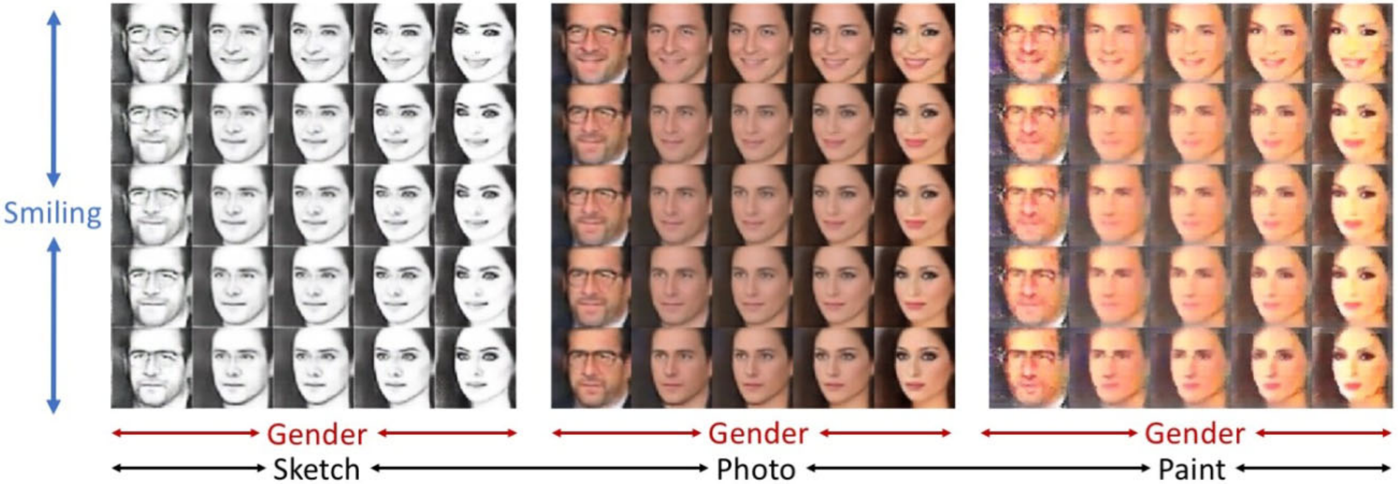


Super-Resolution via Repeated Refinements (SR3) by
Class Diffusion Models (Google, 2021)

Slide credit: I. Goodfellow & SR3

Why We Need Generative Models?

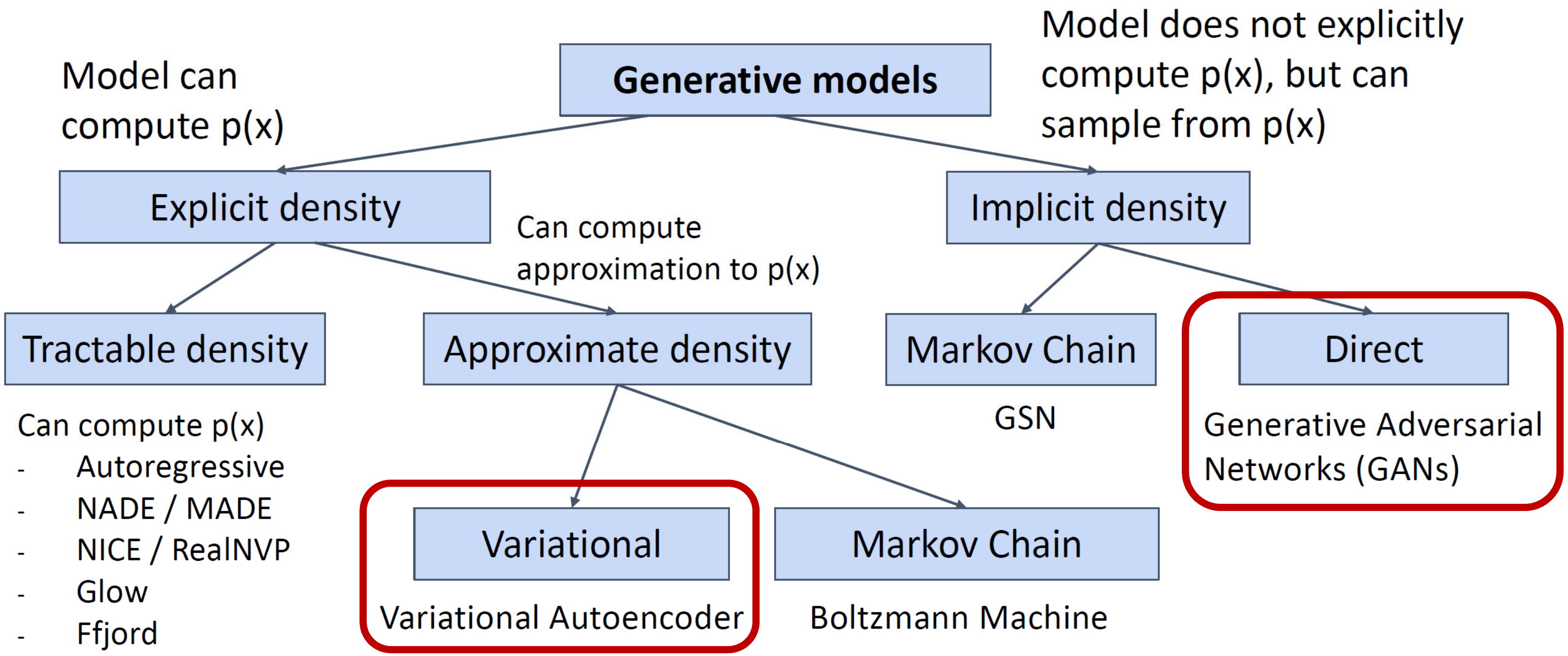
- Remarks
  - Able to process data information (e.g., priors like attribute, category, etc.) for synthesis, prediction, or recognition purposes
    - For example, with latent feature z derived from x, one may have P(z) may describe image variants.
    - Or, z in P(z) may annotate object categorical or attribute information.



  - We will talk about a variety of visual applications based on generative models later.

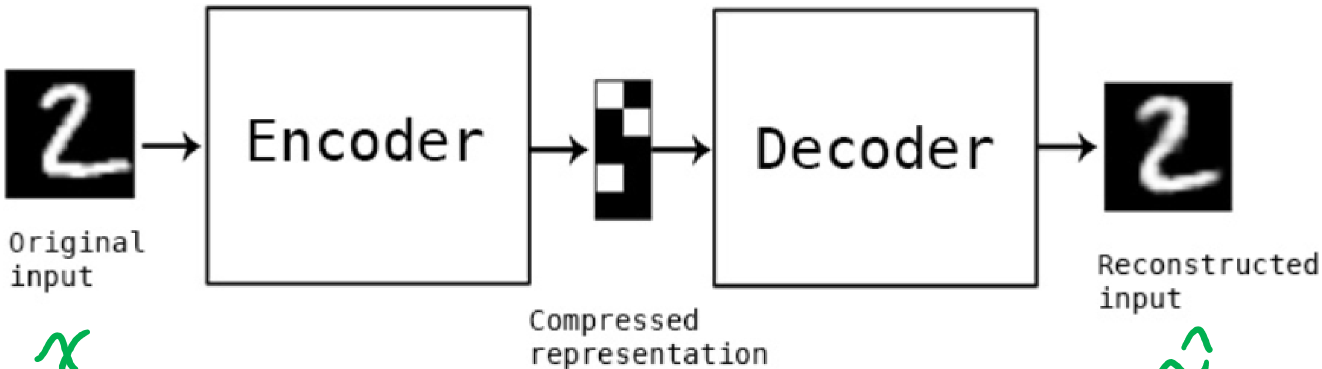
Liu et al., NeurIPS 2018

Taxonomy of Generative Models



Take a Deep Look to Discover Latent Variables/Representations

- Autoencoder
  - Autoencoding = encoding itself with recovery purposes
  - In other words, encode/decode data with reconstruction guarantees
  - Latent variables/features as deep representations
  - Example objective/loss function at output:
    - L2 norm between input and output, i.e., $\min \| \hat{\underline{x}} - \underline{x} \|_2$



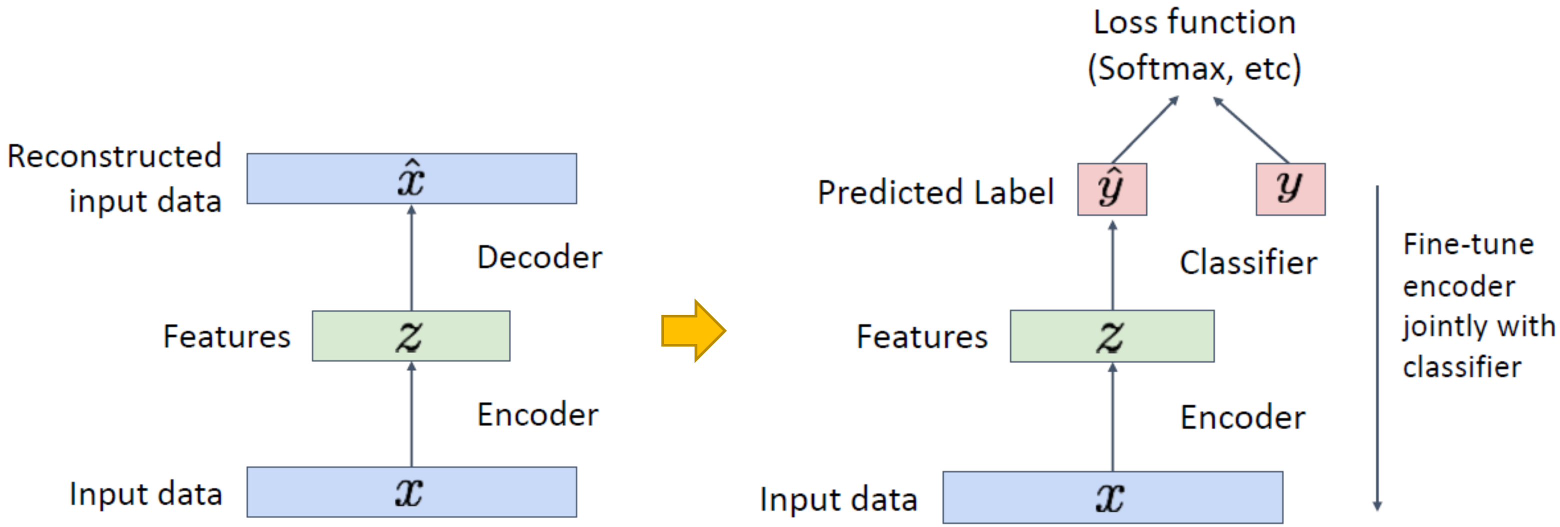
$\underline{x}$ $\hat{\underline{x}}$

most important information of input image ☞ latent space

Slide credit: W. Chiu

Take a Deep Look to Discover Latent Variables/Representations (cont'd)

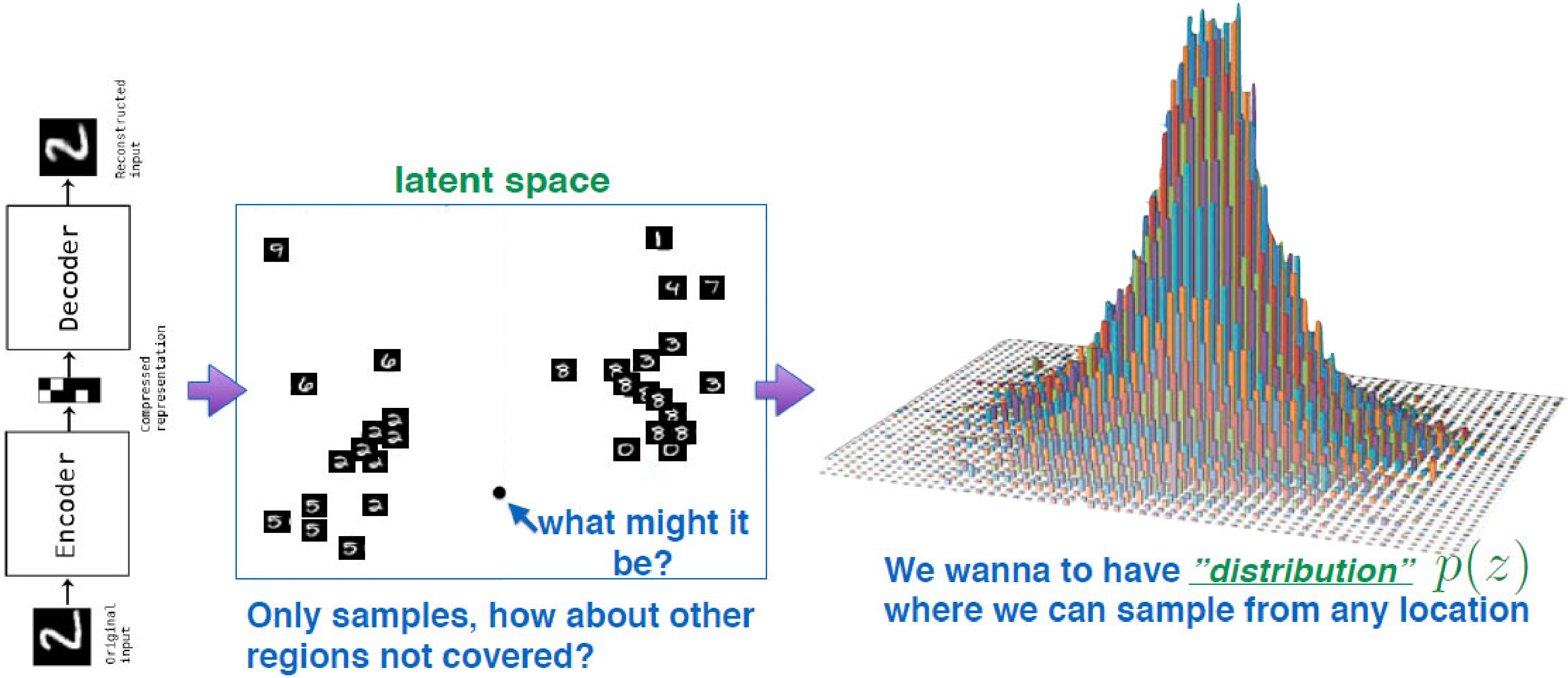
- Autoencoder (AE) for downstream tasks
  - Train AE with reconstruction guarantees
  - Keep encoder (and the derived features) for downstream tasks (e.g., classification)
  - Thus, a trained encoder can be applied to initialize a supervised model



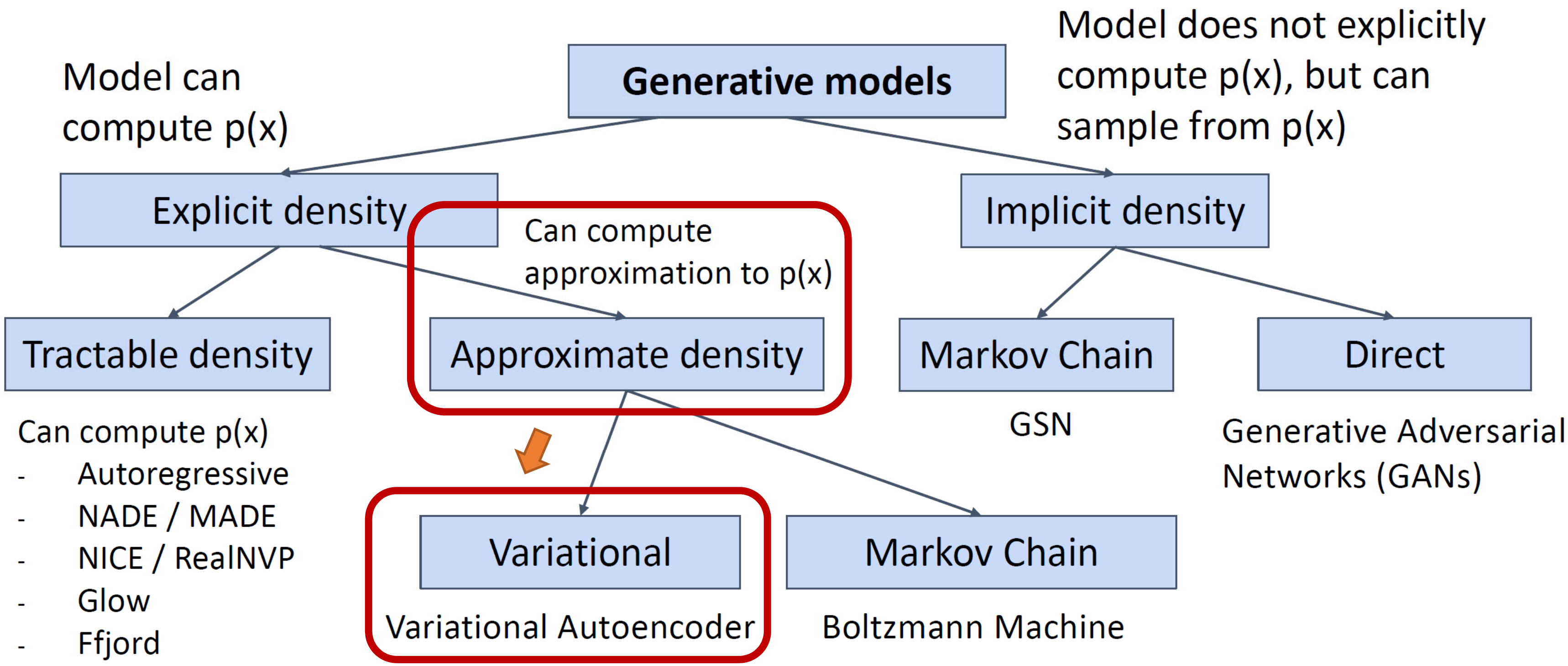
Slide credit: UMich EECS 498-007

Take a Deep Look to Discover Latent Variables/Representations (cont'd)

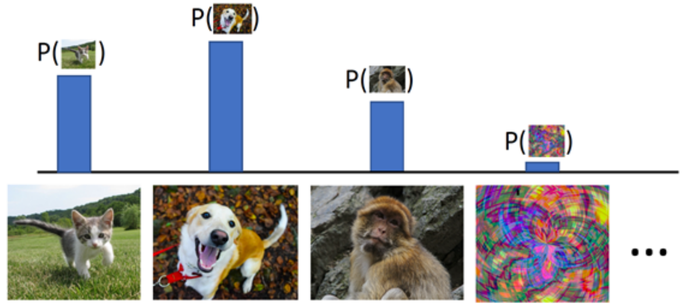
- What's the Limitation of Autoencoder?



Taxonomy of Generative Models



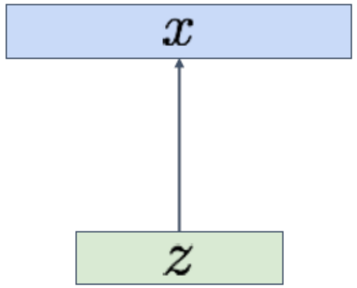
Variational Autoencoder



- Probabilistic Spin on AE
  - Learn latent feature z from raw data x
  - Sample from the latent space (via model) to generate data

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

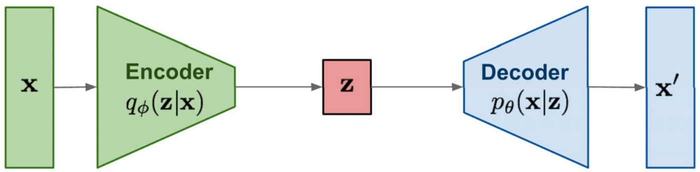
Sample z from prior $p_{\theta^*}(z)$

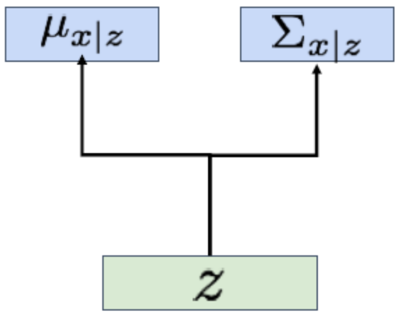


Assume simple prior p(z), e.g. Gaussian

Represent p(x|z) with a neural network (Similar to **decoder** from autencoder)

- p(x|z) is implemented via a (probabilistic) **decoder**



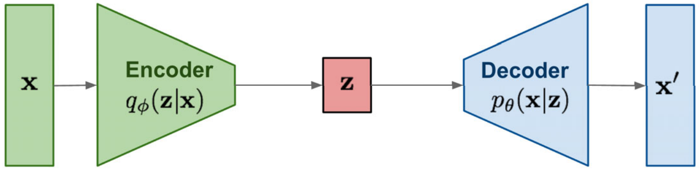


Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

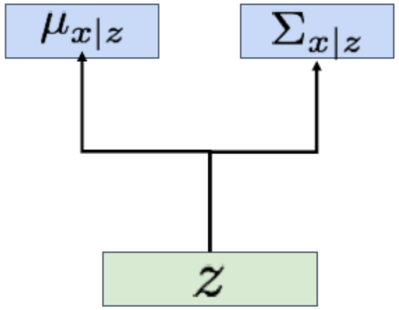
Slide credit: UMich EECS 498-007

Variational Autoencoder (cont'd)



- Remarks
  - Train VAE via maximum likelihood of data p(x)
  - Note that we don't observe z & need to marginalize it:

$$p_\theta(x) = \int p_\theta(x, z)dz = \int p_\theta(x|z)p_\theta(z)dz$$



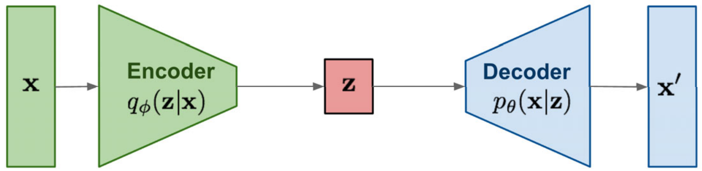
  - We can compute $p_\theta(x|z)$ with the decoder module, and we assume Gaussian prior for $p_\theta(z)$
  - However, can't integrate over all possible z!
  - What else can we do? Recall that we have Bayes' rule:

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

We can't compute $p_\theta(z \mid x)$, but we can train the encoder module to learn

$$q_\phi(z \mid x) \approx p_\theta(z \mid x)$$

Variational Autoencoder (cont'd)



- Again, we aim to maximize

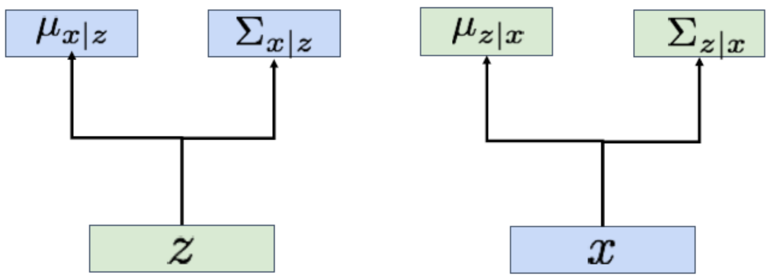
$$p_\theta(x) = \int p_\theta(x, z)dz = \int p_\theta(x|z)p_\theta(z)dz$$

we have…

Decoder network inputs latent code z, gives distribution over data x

Encoder network inputs data x, gives distribution over latent codes z

$$p_\theta(x \mid z) = N(\mu_{x|z}, \Sigma_{x|z}) \quad q_\phi(z \mid x) = N(\mu_{z|x}, \Sigma_{z|x})$$

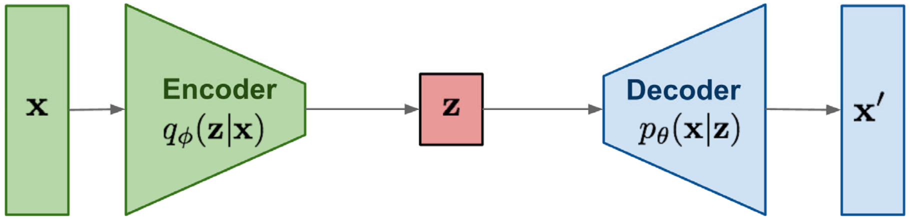


- If we ensure $q_\phi(z \mid x) \approx p_\theta(z \mid x)$

then we have

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)} \approx \frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}$$

Training VAE



$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{\boxed{p_\theta(x|z)}\boxed{p(z)}\boxed{q_\phi(z|x)}}{\boxed{p_\theta(z|x)}\boxed{q_\phi(z|x)}}$$

$$= E_z[\log \boxed{p_\theta(x|z)}] - E_z\left[\log \frac{\boxed{q_\phi(z|x)}}{\boxed{p(z)}}\right] + E_z\left[\log \frac{\boxed{q_\phi(z|x)}}{\boxed{p_\theta(z|x)}}\right]$$

$$= E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right) + D_{KL}(q_\phi(z|x), p_\theta(z|x))$$

Data reconstruction

KL divergence between sample distribution from the encoder and the prior

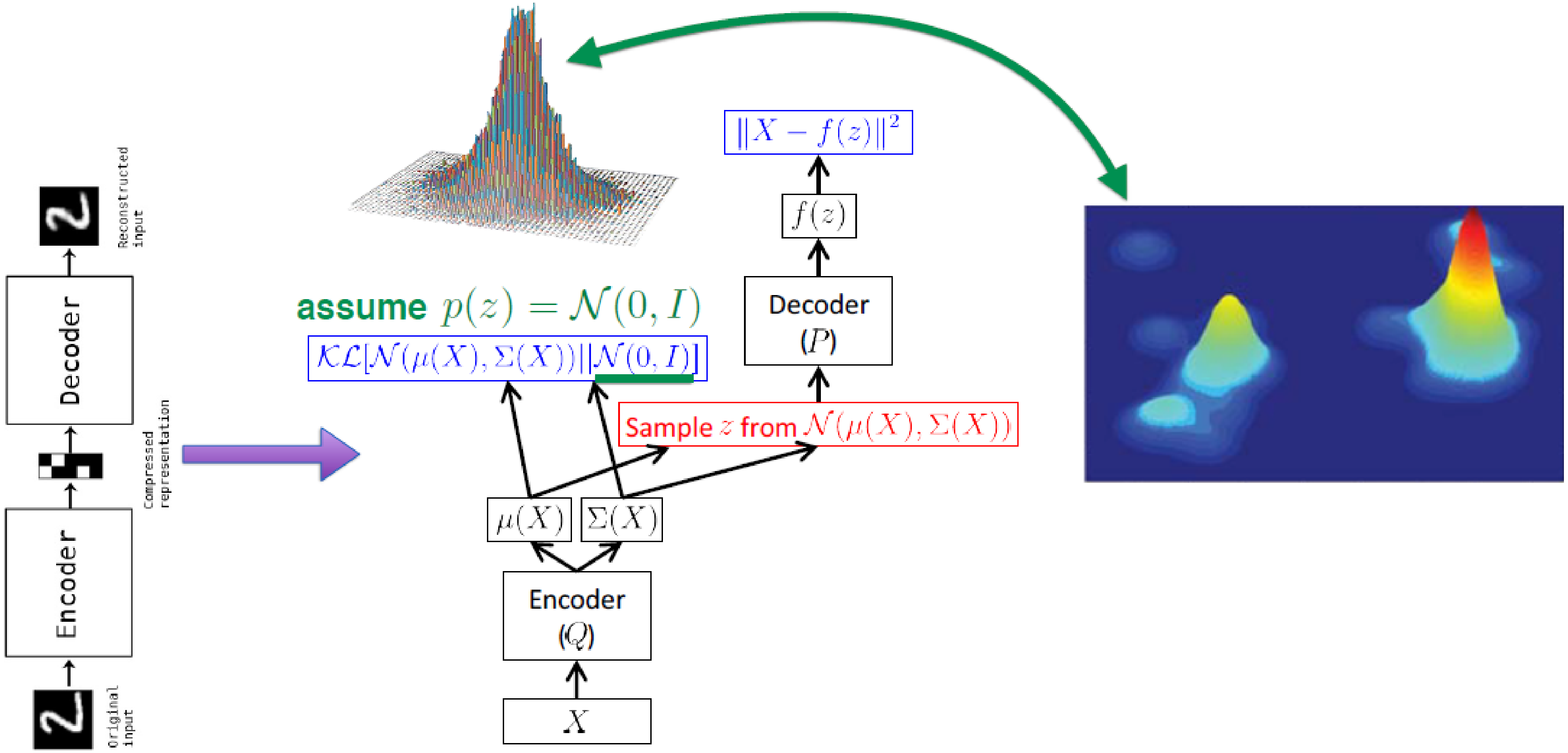
KL divergence between sample distribution from the encoder and the posterior of data

$$\log p_\theta(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

i.e., **variational lower bound** on the **data likelihood $p_\Theta(x)$**

Summary:
From Autoencoder to Variational Autoencoder

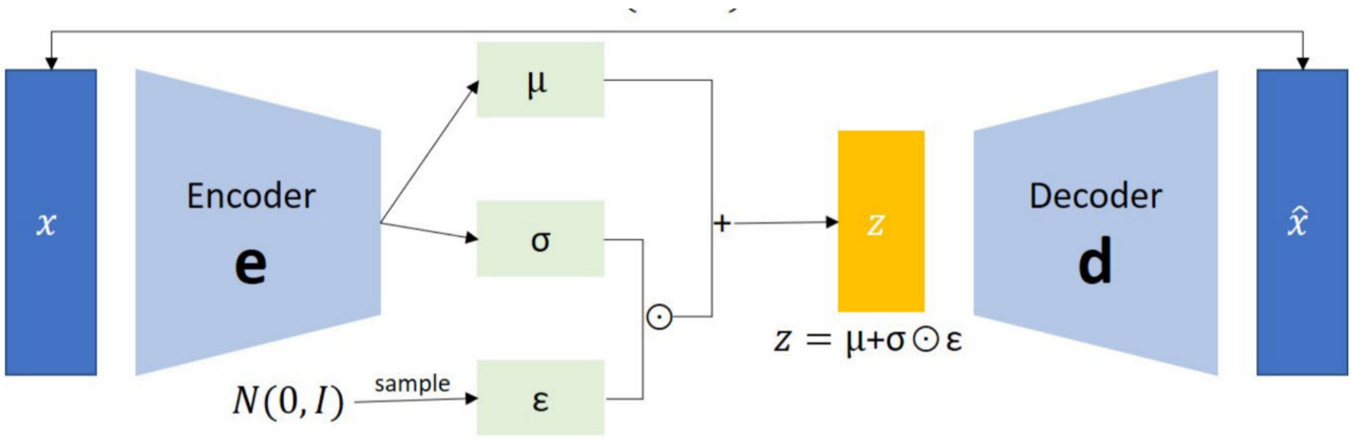
Now is a “*distribution*”, we can assume it to be
a distribution easy to sample from, e.g. Gaussian



Slide credit: W. Chiu

Reparameterization Trick in VAE

- Remarks
  - Given x, sample z from latent distribution (described by output parameters of encoder)
  - However, this creates a bottleneck since backpropagation cannot flow through
  - Alternatively, we apply $z = \mu + \sigma \odot \varepsilon$ (ε simply generated by **Normal distribution**).
  - This enables BP gradients in encoder through μ and σ,
    while maintaining stochasticity via ε (for generative model purposes).



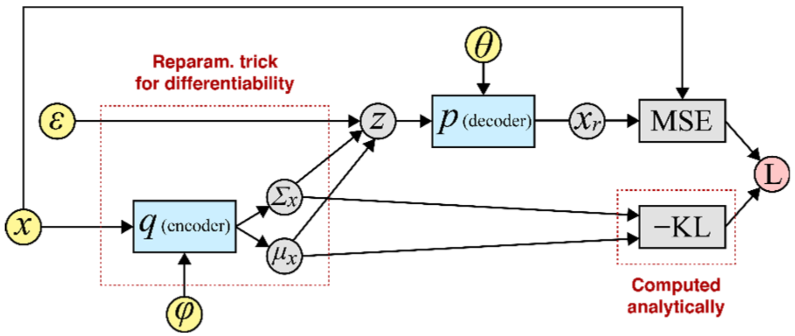
https://medium.com/geekculture/variational-autoencoder-vae-9b8ce5475f68

Implementation of VAE

$\|X - f(z)\|^2$

$\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x = M(\mathbf{x}), \Sigma(\mathbf{x})$	Push $\mathbf{x}$ through encoder
$\epsilon \sim \mathcal{N}(0,1)$	Sample noise
$\mathbf{z} = \epsilon\boldsymbol{\sigma}_x + \boldsymbol{\mu}_x$	Reparameterize
$\mathbf{x}_r = p_\theta(\mathbf{x} \mid \mathbf{z})$	Push $\mathbf{z}$ through decoder
recon. loss $= \text{MSE}(\mathbf{x}, \mathbf{x}_r)$	Compute reconstruction loss
var. loss $= -\text{KL}[\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x) \| \mathcal{N}(0, I)]$	Compute variational loss
L = recon. loss + var. loss	Combine losses

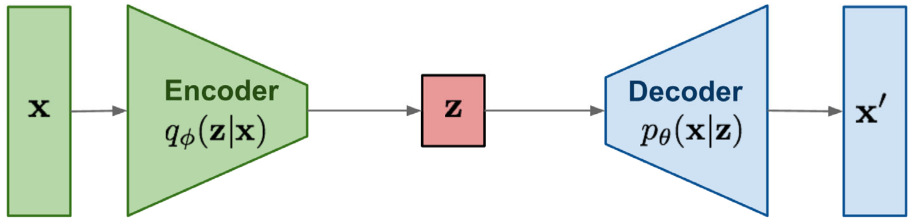
X



Initialize parameters of encoder and decoder
Repeat:
Get mini-batch of **X**
mu_X, **var_X** = encoder(**X**)
ϵ = sampling from Normal(**0**, **I**)
z = **mu_X** + ϵ***var_X**
X' = decoder(**z**)
recon_loss = ***MSE***(**X**,**X'**)
latent_loss =
KLD(Normal(**mu_X**,**var_X**)||Normal(**0**,**I**))
all_loss = recon_loss + latent_loss
all_loss.backward()
Until: parameters of encoder & decoder converge
Return parameters of encoder and decoder

First sample noise ϵ from Normal(0,I),
then reparameterize z by mu_X + ϵ*var_X,
(equivalently sampled Normal(mu_X, var_X)).
The model is now differentiable!

Before We Move On…



$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{\boxed{p_\theta(x|z)}\boxed{p(z)}\boxed{q_\phi(z|x)}}{\boxed{p_\theta(z|x)}\boxed{q_\phi(z|x)}}$$

$$= E_z[\log \boxed{p_\theta(x|z)}] - E_z\left[\log \frac{\boxed{q_\phi(z|x)}}{\boxed{p(z)}}\right] + E_z\left[\log \frac{\boxed{q_\phi(z|x)}}{\boxed{p_\theta(z|x)}}\right]$$

$$= E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right) + D_{KL}(q_\phi(z|x), p_\theta(z|x))$$

Data reconstruction

KL divergence between sample distribution from the encoder and the prior

KL divergence between sample distribution from the encoder and the posterior of data

$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

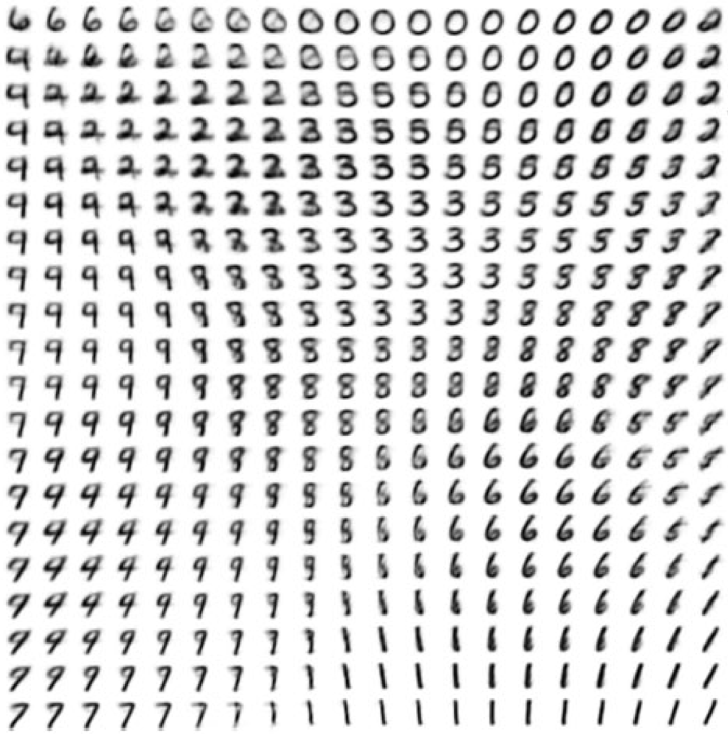
i.e., **variational lower bound** on the **data likelihood $p_\Theta(x)$**

Slide credit: UMich EECS 498-007

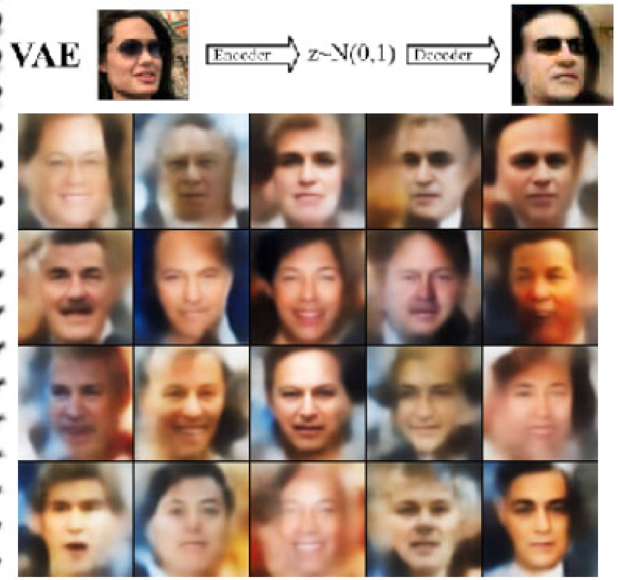
From Autoencoder to Variational Autoencoder (cont'd)

- Example Results

(a) Learned Frey Face manifold



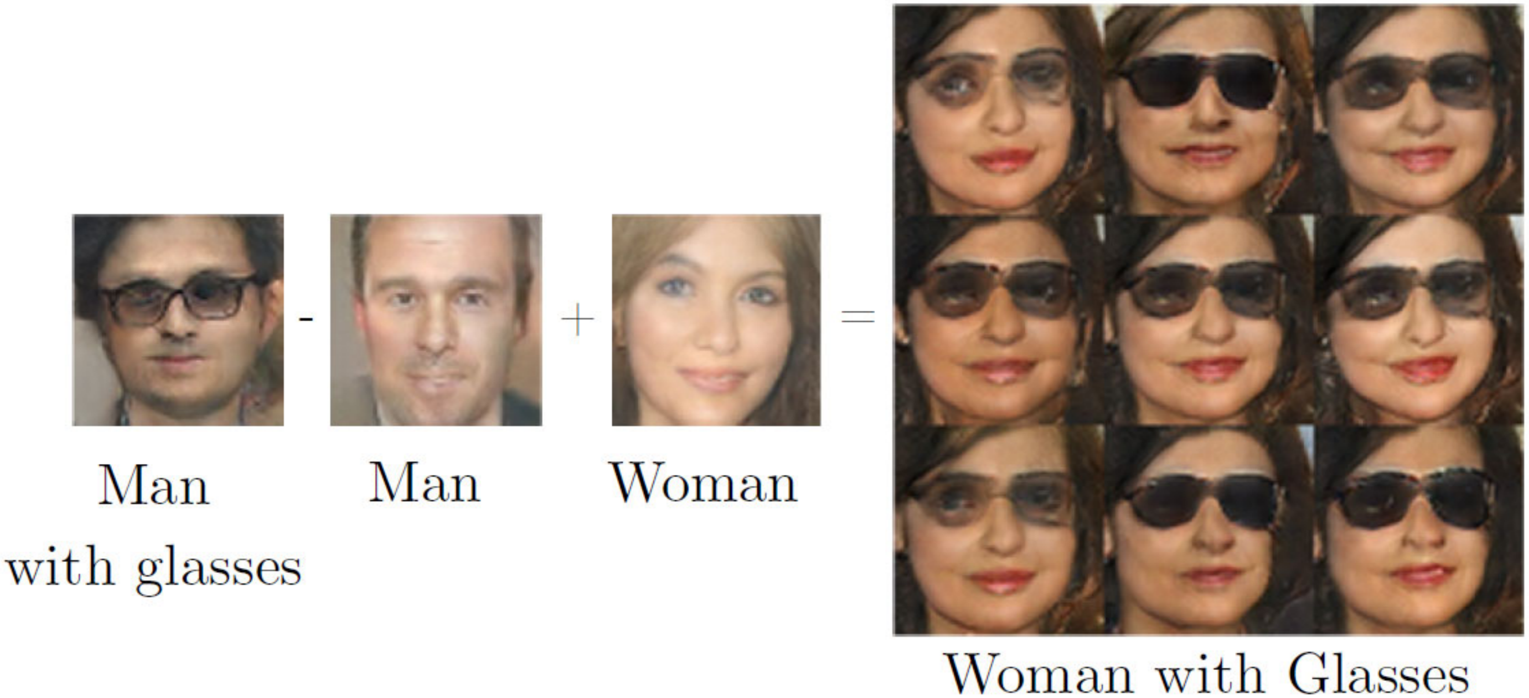
(b) Learned MNIST manifold



Kingma et al., 2013

From Autoencoder to Variational Autoencoder (cont'd)

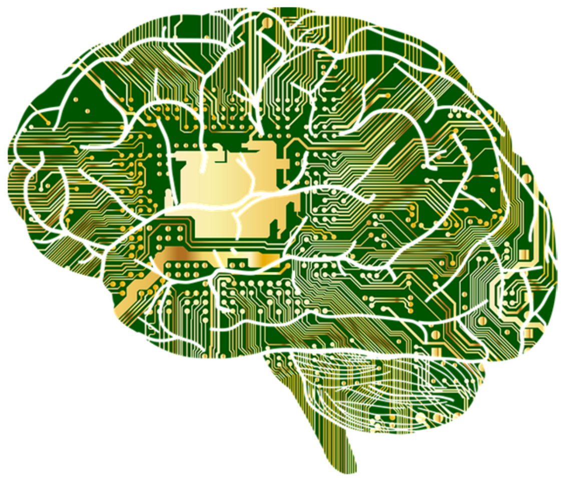
- Example Results
  - A' – A + B = B'



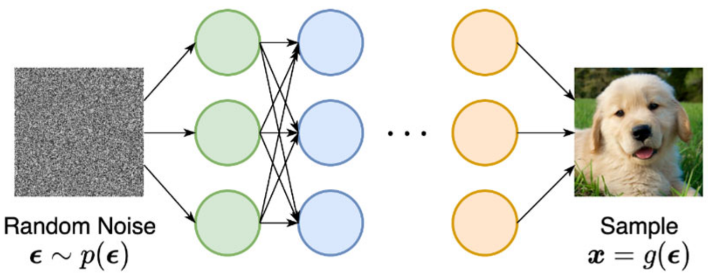
Radford et al., 2015

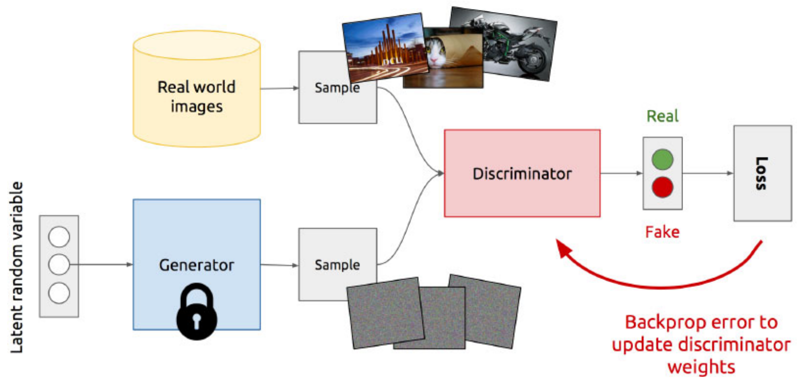
What’s to Be Covered Today…

- Generative Models
  - Auto-Encoder vs. Variational Auto-Encoder
  - Generative Adversarial Network (GAN)
  - Diffusion Model

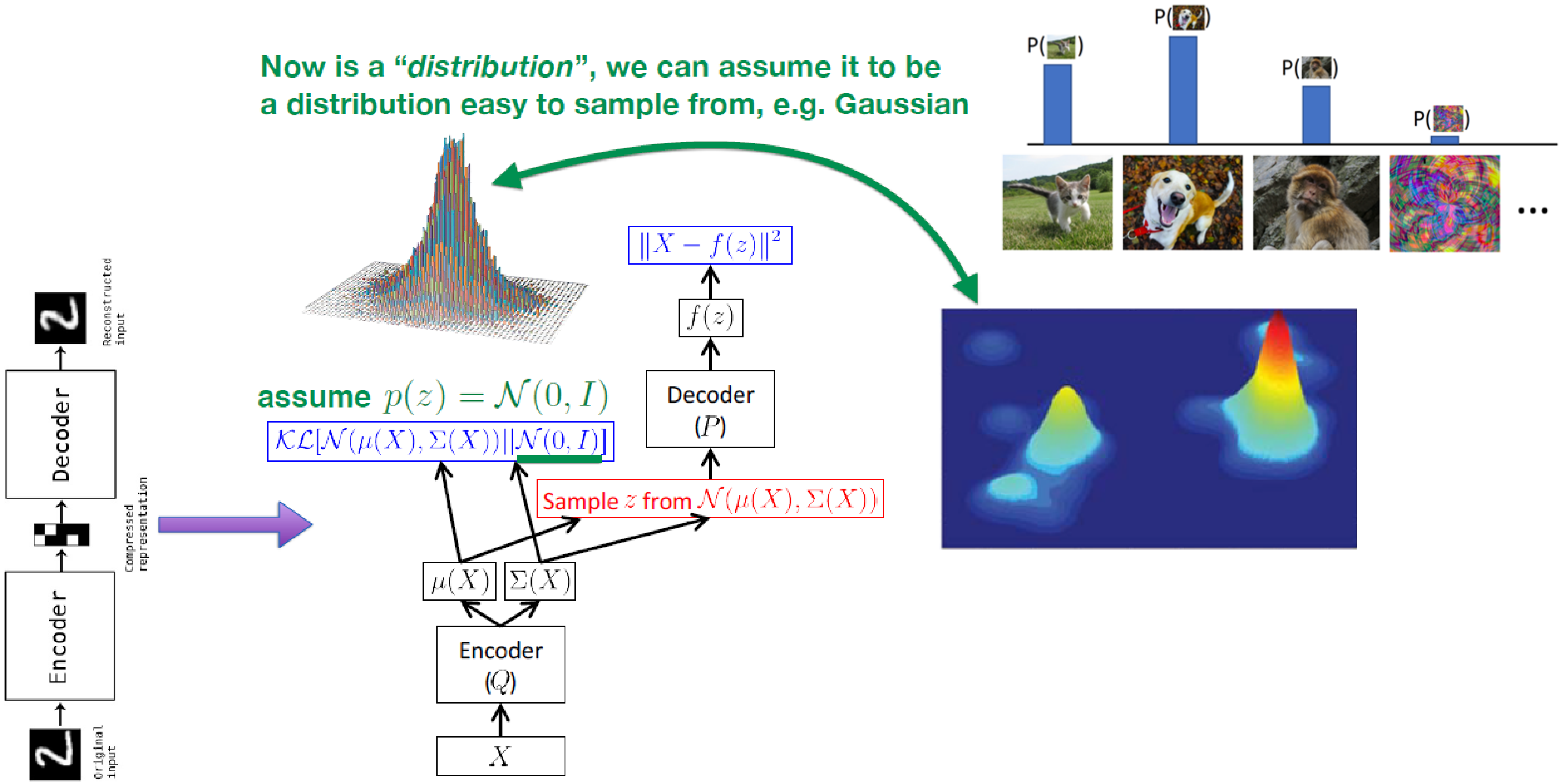


- HW #1 is due Oct. 10th Mon 23:59
- HW #2 will be out next week…

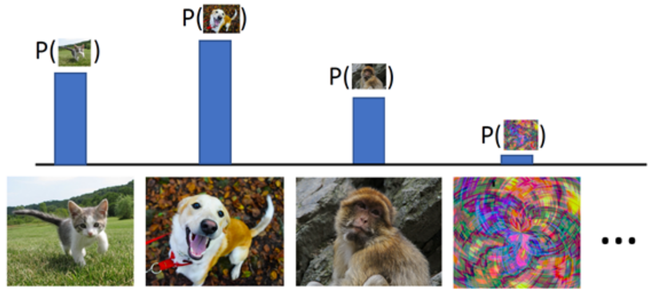




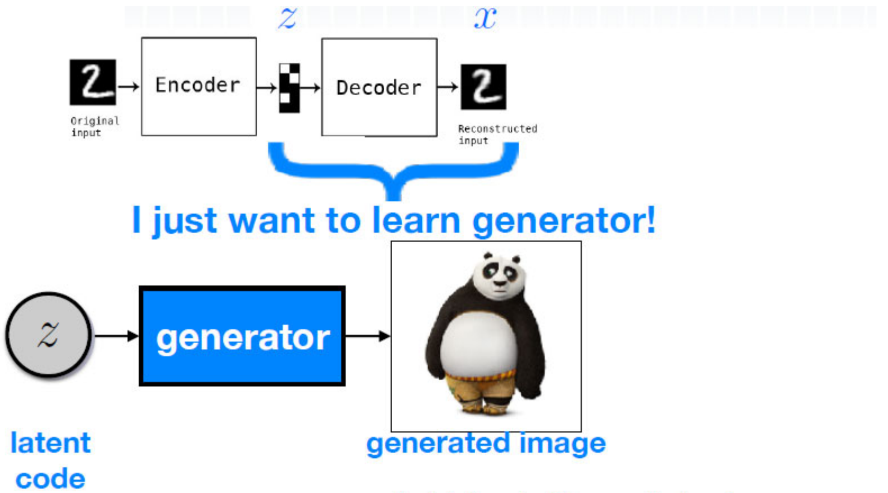
From VAE to *Generative Adversarial Networks (GAN)*



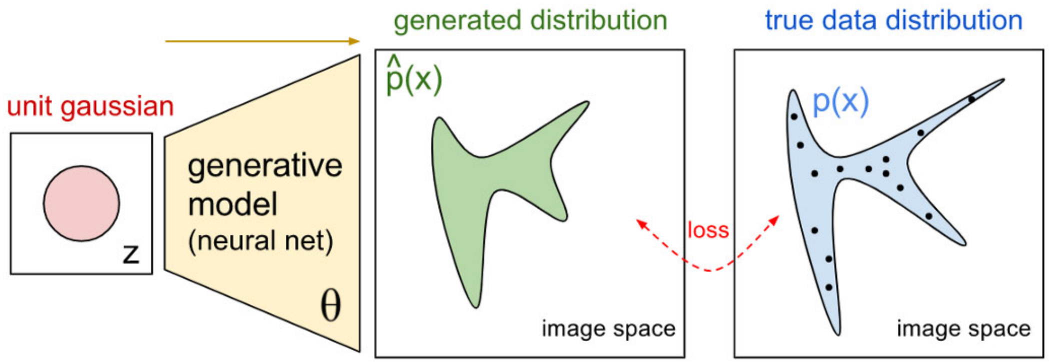
From VAE to GAN (cont'd)



- Remarks
  - What if we only need the decoder/generator in practice?
  - How do we know if the output images are sufficiently good?



might look like a fake image,
how to get it more realistic?



Slide credit: W. Chiu

Generative Adversarial Network

- Idea
  - **Generator** to convert a vector z (sampled from P_z) into fake data x (from P_G), while we need $P_G = P_{data}$
  - **Discriminator** classifies data as real or fake (1/0)
  - How? Impose an **adversarial loss** on the observed data distribution!

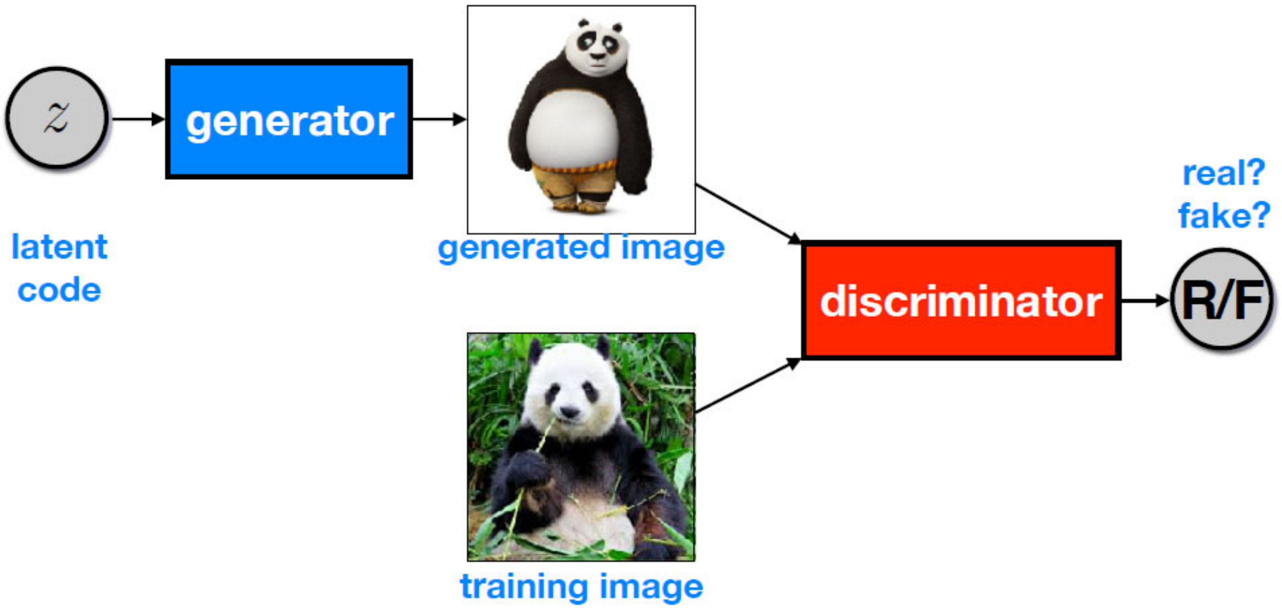


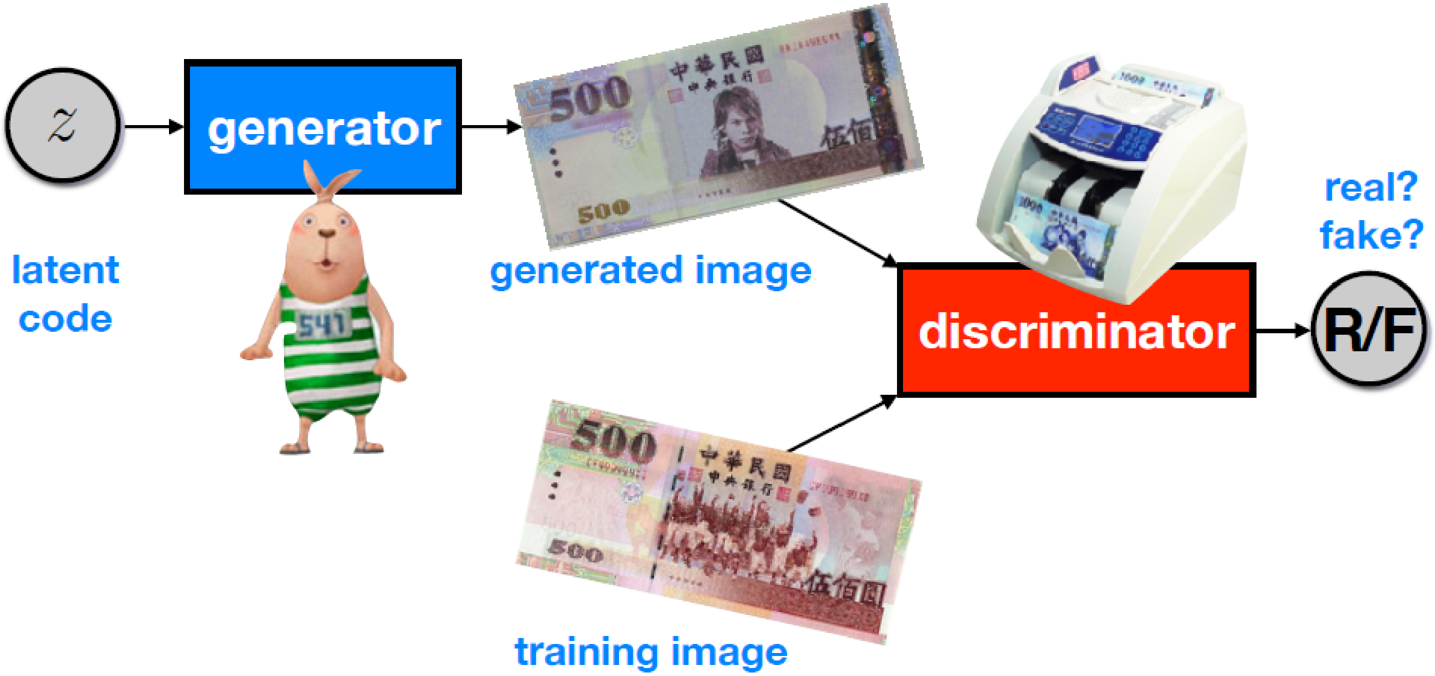
Image credit: W. Chiu

Generative Adversarial Network (cont'd)

- Idea
  - Impose adversarial loss on data distribution
  - Let's see a practical example…

generator: try to generate more realistic images to cheat discriminator

discriminator: try to distinguish whether the image is generated or real

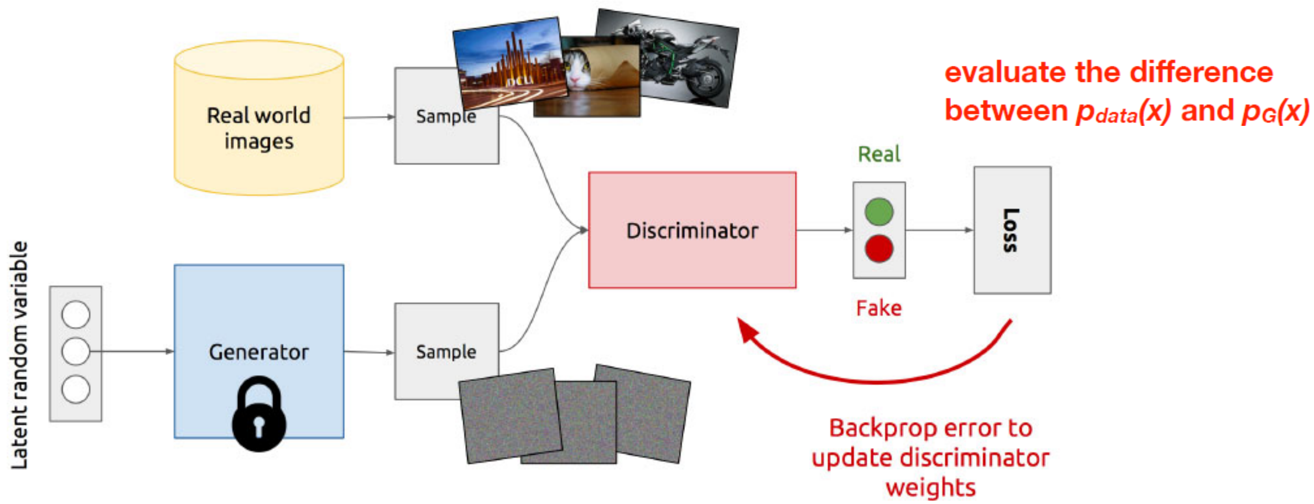


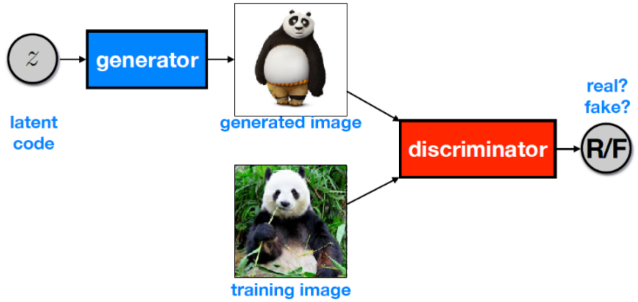
Slide credit: W. Chiu

GAN (cont'd)

- Remarks
  - A function maps **normal distribution** $\boldsymbol{N(0, I)}$ to $\boldsymbol{P_{data}}$
  - How good we are in mapping P_g to P_{data}?
    - Train & ask the discriminator!
  - Conduct a two-player min-max game

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

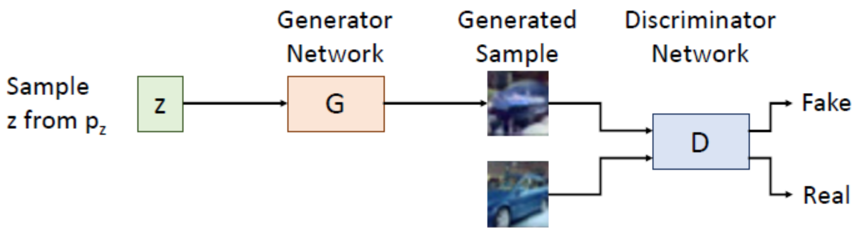




Training Objective of GAN

- Jointly train generator G and discriminator D with a min-max game

$$\min_{G} \max_{D} \left(\overbrace{E_{x\sim p_{data}}[\log D(x)]}^{\substack{\text{Discriminator wants} \\ D(x)\,=\,1 \text{ for real data}}} + E_{z\sim p(z)} \underbrace{\overbrace{\left[\log\left(1 - D\left(G(z)\right)\right)\right]}^{\substack{\text{Discriminator wants} \\ D(x)\,=\,0 \text{ for fake data}}}}_{\substack{\text{Generator wants} \\ D(x)\,=\,1 \text{ for fake data}}} \right)$$



- Train G & D with alternating gradient updates

$$\min_{G} \max_{D} V(G, D)$$

For t in 1, … T:

1. (Update D) $D = D + \alpha_D \frac{\partial V}{\partial D}$
2. (Update G) $G = G - \alpha_G \frac{\partial V}{\partial G}$

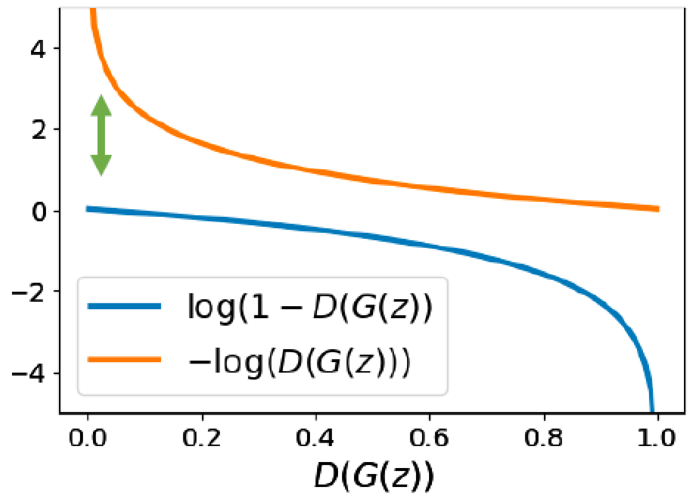
Slide credit: I. Goodfellow

Training Objective of GAN *(optional trick)*

- Potential Problem

$$\min_{G}\max_{D}\left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]\right)$$

  - At start of training, G is not OK yet (obviously);
    D easily tells apart real/fake data (i.e., D(G(z)) close to 0).
  - Solution:
    - Instead of training G to minimize log(1-D(z)) in the beginning,
      we train G to minimize -log(D(G(z)).
    - With strong gradients from G, we start the training of the above min-max game.



Slide credit: I. Goodfellow

Optimality of GAN

- Why the min-max game as objective a good idea?

$$\min_{G} \max_{D} \left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)} \left[\log\left(1 - D\left(G(z)\right)\right)\right]\right)$$

$$= \min_{G} \max_{D} \left(E_{x\sim p_{data}}[\log D(x)] + E_{x\sim p_G}[\log(1 - D(x))]\right)$$

$$= \min_{G} \int_X \max_{D} \left(p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x))\right) dx$$

$$\left.\begin{aligned} f(y) &= a \log y + b \log(1-y) \\ f'(y) &= \frac{a}{y} - \frac{b}{1-y} \end{aligned}\right\} f'(y) = 0 \Leftrightarrow y = \frac{a}{a+b} \text{ (local max)}$$

Optimal Discriminator: $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$

Optimality of GAN

- Why the min-max game as objective a good idea? (cont'd)

$$\min_{G} \max_{D} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D\left(G(z) \right) \right) \right] \right)$$

$$\Rightarrow \min_{G} \int_{X} \left(p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} + p_G(x) \log \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right) dx$$

$$= \min_{G} \left(E_{x \sim p_{data}} \left[\log \frac{2}{2} \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[\log \frac{2}{2} \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right] \right)$$

$$= \min_{G} \left(E_{x \sim p_{data}} \left[\log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[\log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

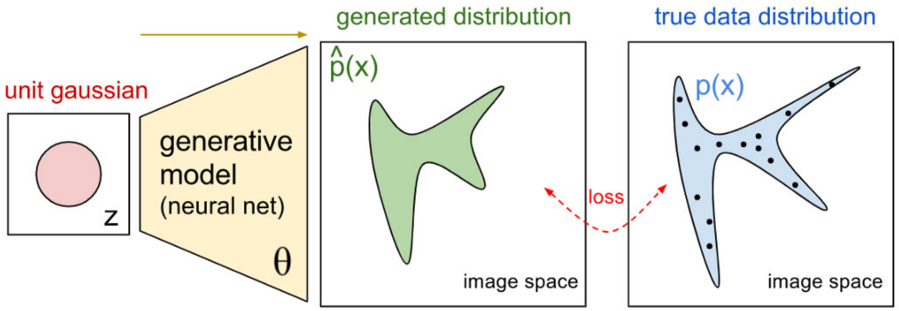
Optimality of GAN

- Why the min-max game as objective a good idea? (cont'd)

$$\min_G \max_D \left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]\right)$$

$$= \min_G \left(E_{x\sim p_{data}}\left[\log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)}\right] + E_{x\sim p_G}\left[\log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)}\right] - \log 4\right)$$

$$= \min_G \left(KL\left(p_{data}, \frac{p_{data} + p_G}{2}\right) + KL\left(p_G, \frac{p_{data} + p_G}{2}\right) - \log 4\right)$$



Kullback-Leibler Divergence:

$$KL(p, q) = E_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right]$$

Slide credit: UMich EECS 498-007

Optimality of GAN

- Why the min-max game as objective a good idea? (cont'd)

$$\min_G \max_D \left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]\right)$$

$$= \min_G \left(E_{x\sim p_{data}}\left[\log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)}\right] + E_{x\sim p_G}\left[\log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)}\right] - \log 4\right)$$

$$= \min_G \left(KL\left(p_{data}, \frac{p_{data} + p_G}{2}\right) + KL\left(p_G, \frac{p_{data} + p_G}{2}\right) - \log 4\right)$$

$$= \min_G (2 * JSD(p_{data}, p_G) - \log 4)$$

JSD is always nonnegative, and zero only when the two distributions are equal! Thus $p_{data} = p_G$ is the global min, QED

Jensen-Shannon Divergence:

$$JSD(p, q) = \frac{1}{2}KL\left(p, \frac{p+q}{2}\right) + \frac{1}{2}KL\left(q, \frac{p+q}{2}\right)$$

Slide credit: UMich EECS 498-007

Remarks on Optimality of GAN

$$\min_G \max_D \left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]\right)$$
$$= \min_G (2 * JSD(p_{data}, p_G) - \log 4)$$

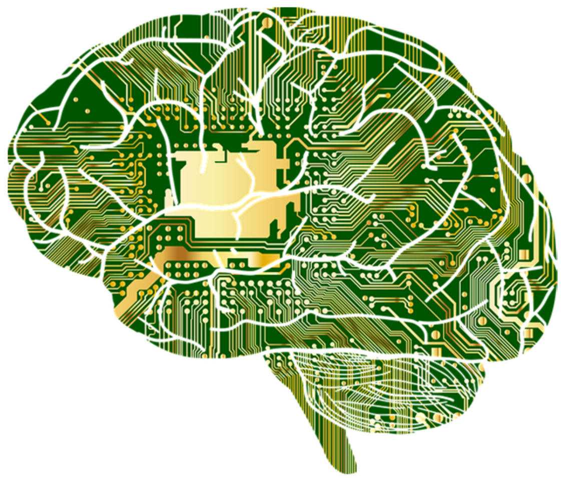
- Summary
  - The global min of the minmax game happens when

1. $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$ (Optimal discriminator for any G)
2. $p_G(x) = p_{data}(x)$ (Optimal generator for optimal D)

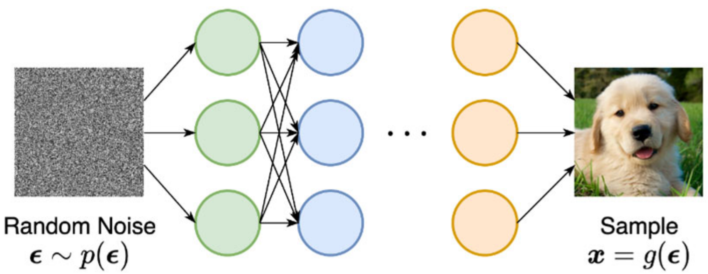
- **Caution!**
  - G and D are learned models (i.e., DNNs) with fixed architectures.
    We don’t know whether we can actually represent the optimal D & G.
  - Optimality of GAN does not tell anything about convergence to the optimal D/G.

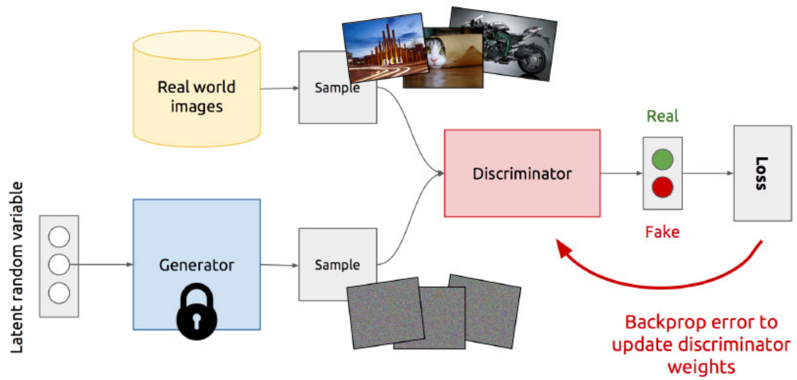
What's to Be Covered Today…

- Generative Models
  - Auto-Encoder vs. Variational Auto-Encoder
  - Generative Adversarial Network (GAN)
    - Challenges & Variants of GAN
  - Diffusion Model



- HW #1 is due Oct. 10th Mon 23:59
- HW #2 will be out next week…





Deep Convolutional GAN (DC-GAN)

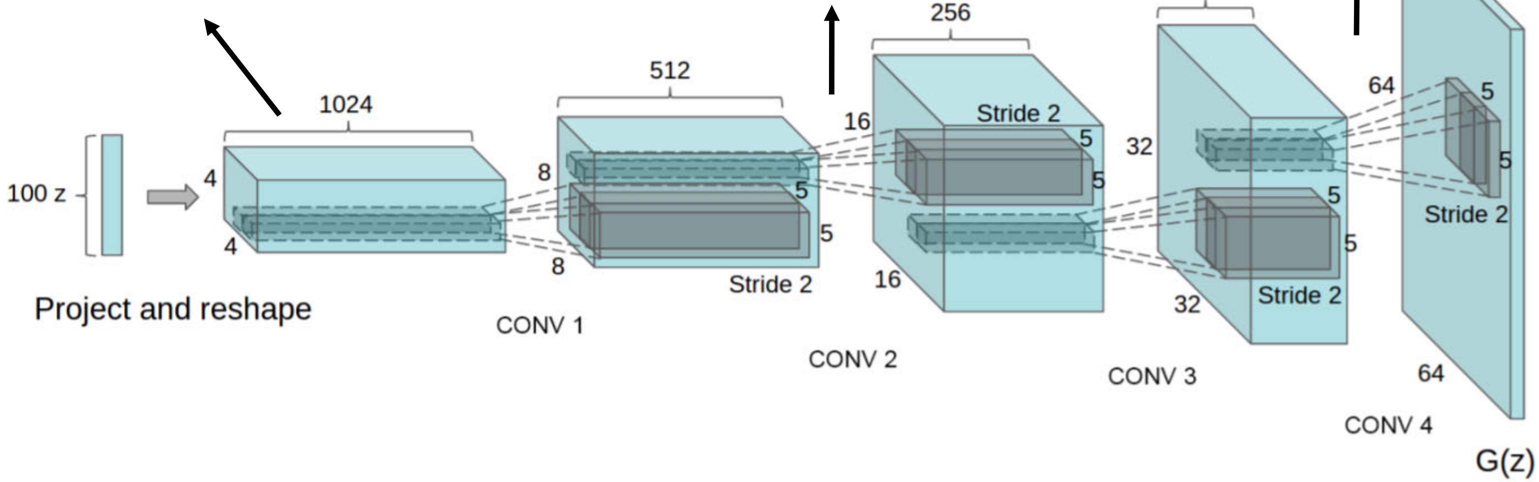
- Remarks
  - ICLR 2016
  - A CNN+GAN architecture
  - Empirically make training of GAN more stable

Generator

Batchnorm & ReLU activation

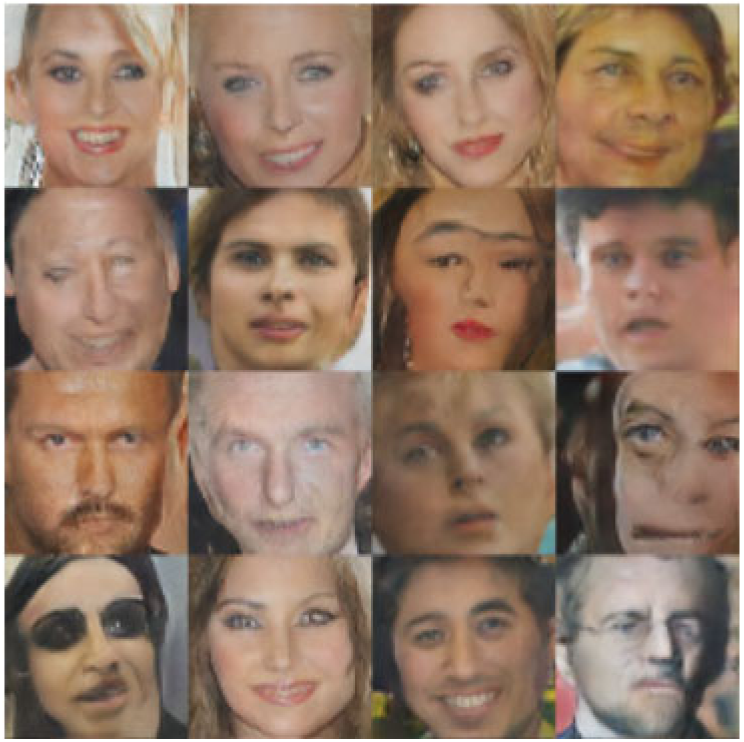
Fractional convolution

Remove fully connected layer

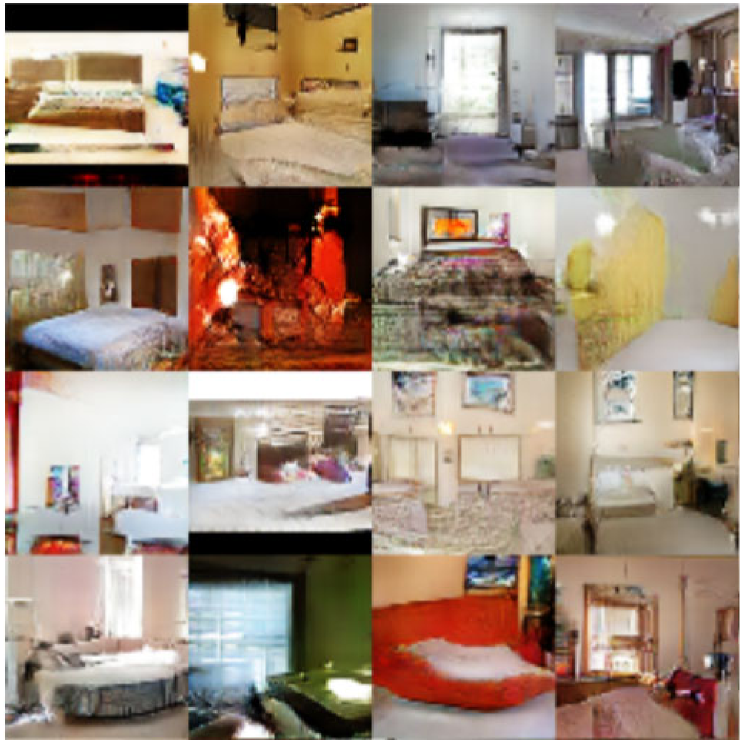


Deep Convolutional GAN (DC-GAN)

- Example Results



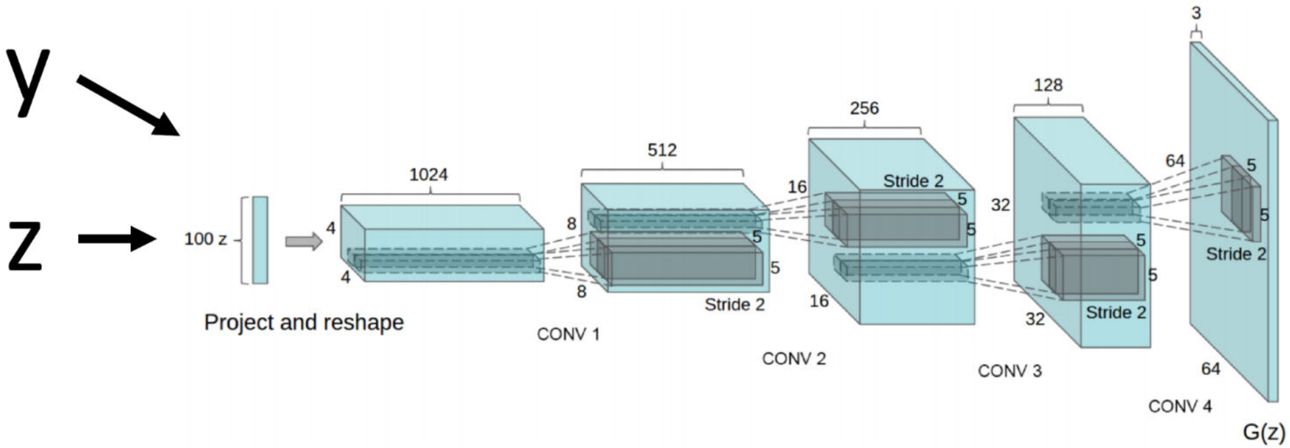
Collected face dataset



LSUN dataset

Conditional GANs

- Remarks
  - ICLR 2016
  - Conditional generative model p(x|y) instead of p(x)
  - Both G and D take the label y as an additional input…Why? Why not just use D as designed in the standard GAN?



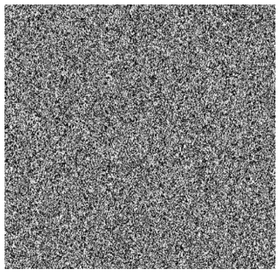
Conditional GANs

- Example Results

Miyato et al, “Spectral Normalization for Generative Adversarial Networks”, ICLR 2018

Problems in Training GANs: Vanishing Gradients

- What Might Go Wrong?
  - GAN training is often unstable.
  - In other words, training might not converge properly.
  - The discriminator which we prefer is…



0

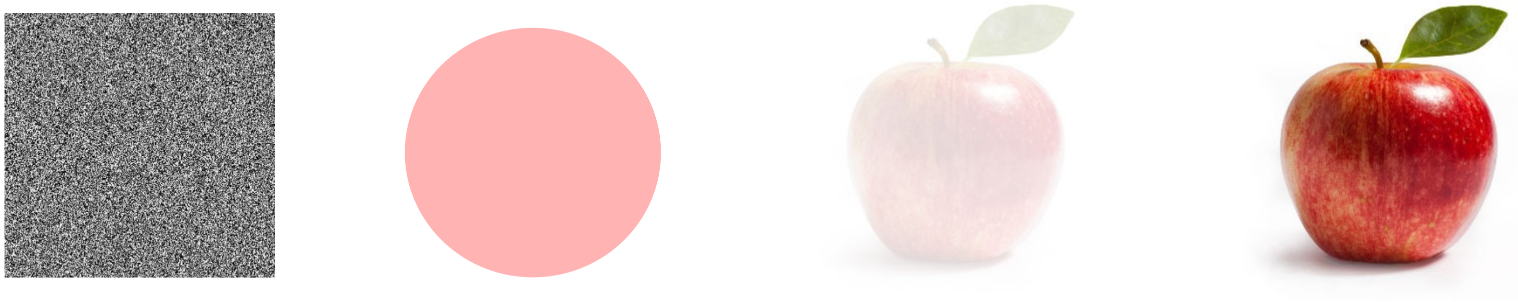
0.2

0.6

1

Problems in Training GANs: Vanishing Gradients (cont'd)

- What Might Go Wrong?
  - GAN training is often unstable.
  - In other words, training might not converge properly.
  - The discriminator we trained might be as follows.
    In other words, no gradient to guide the generator to output proper images.

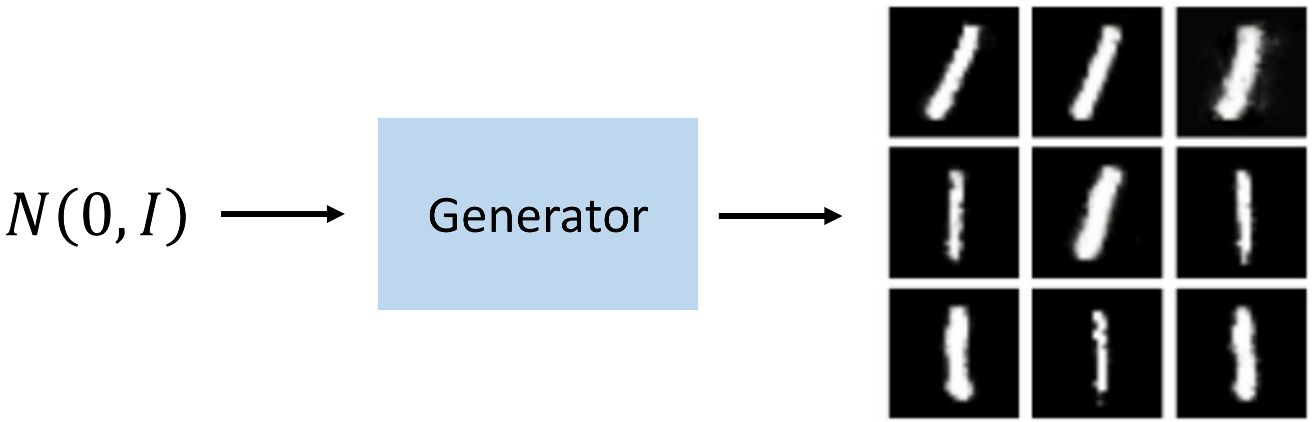


0 0 0 1

- This is known as the problem of *vanishing gradients*.

Problems in Training GANs: Mode Collapse

- Remarks
  - The generator only outputs a limited number of image variants regardless of the inputs.



Mode collapse!

Problems in Training GANs: Mode Collapse (cont'd)

- Remarks
  - The generator only outputs a limited number of image variants regardless of the inputs.

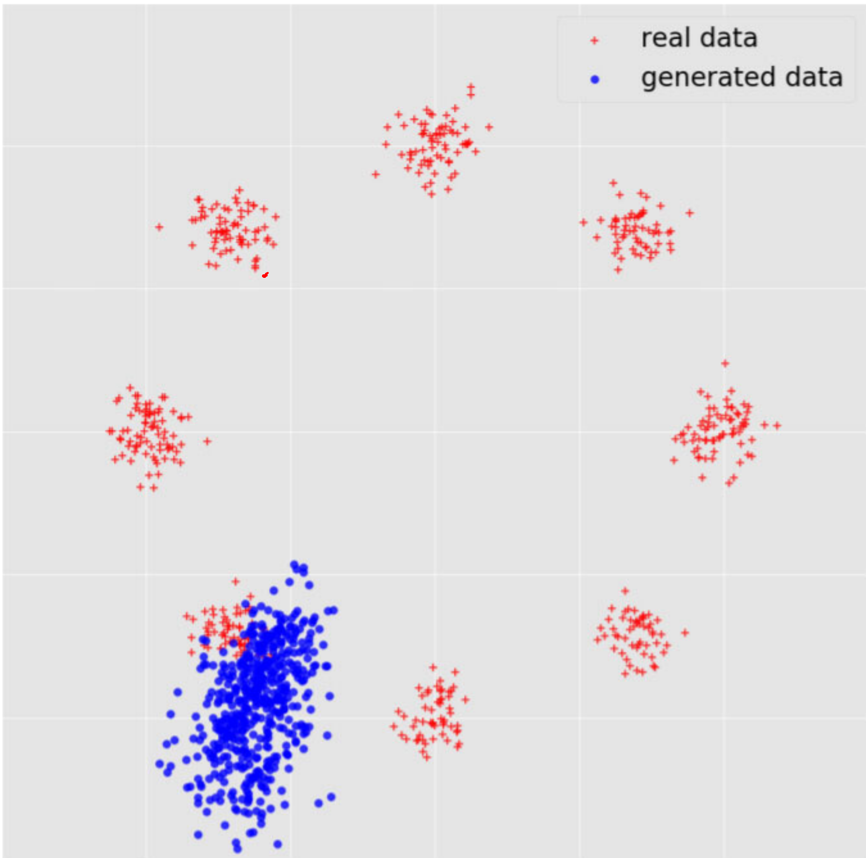
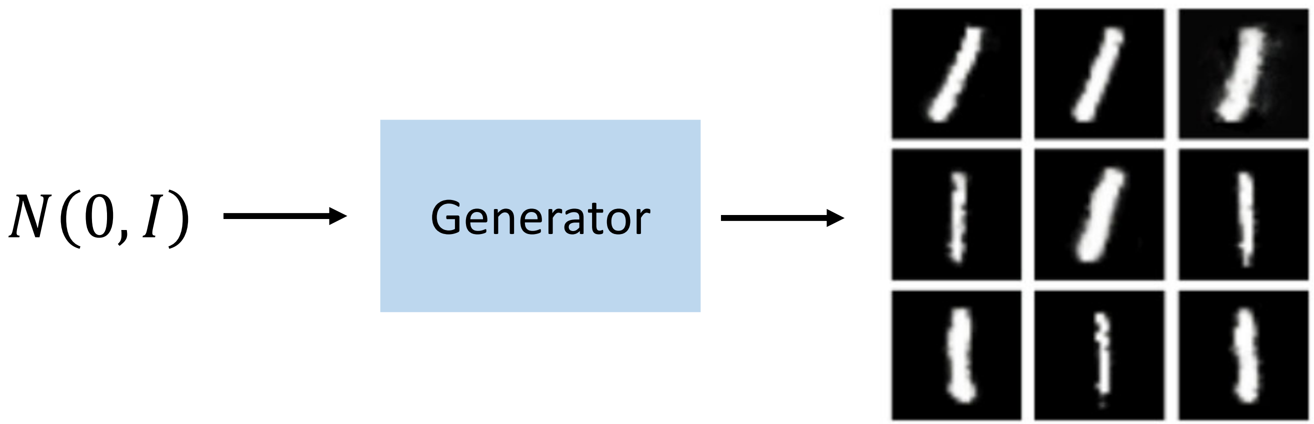


Photo credit:
https://openreview.net/pdf?id=rkmu5b0a-

Problems in Training GANs: Mode Collapse (cont'd)

- Why Mode Collapse Happens?
  - The objective of GANs assesses the image authenticity, not diversity.
  - Imbalance training between generator/discriminator (exploding/vanishing gradients)



Energy-Based GAN

- Energy Function
  - Converting input data into scalar outputs, viewed as energy values
  - Desired configuration is expected to output low energy values & vice versa.
- Energy Function as Discriminator
  - Use of autoencoder; can be pre-trained!
  - Reconstruction loss outputs a range of values instead of binary logistic loss.
  - Empirically better convergence

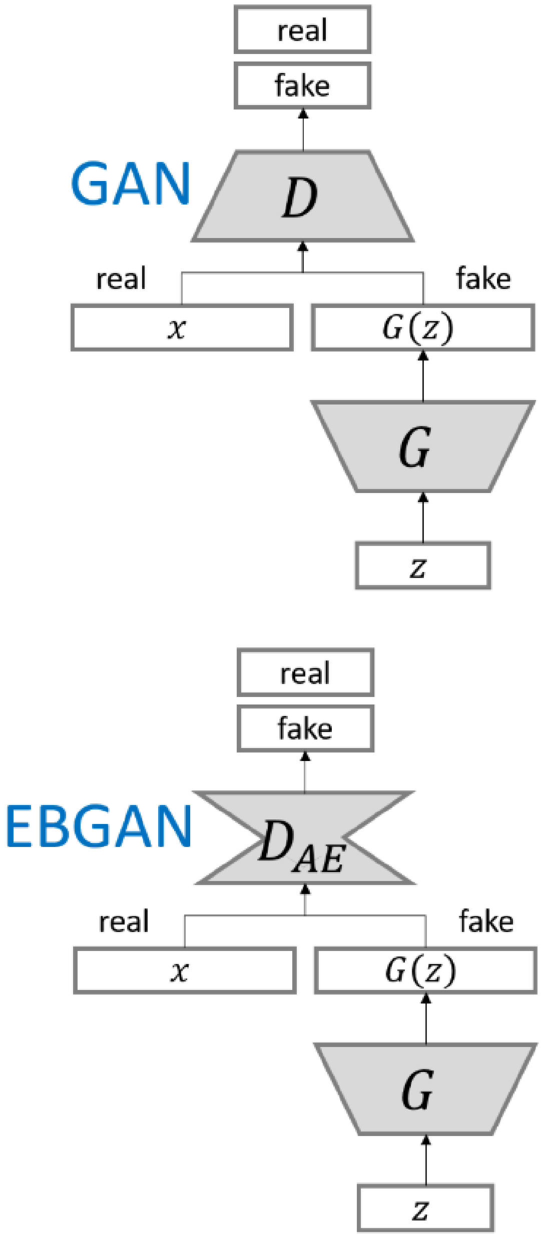
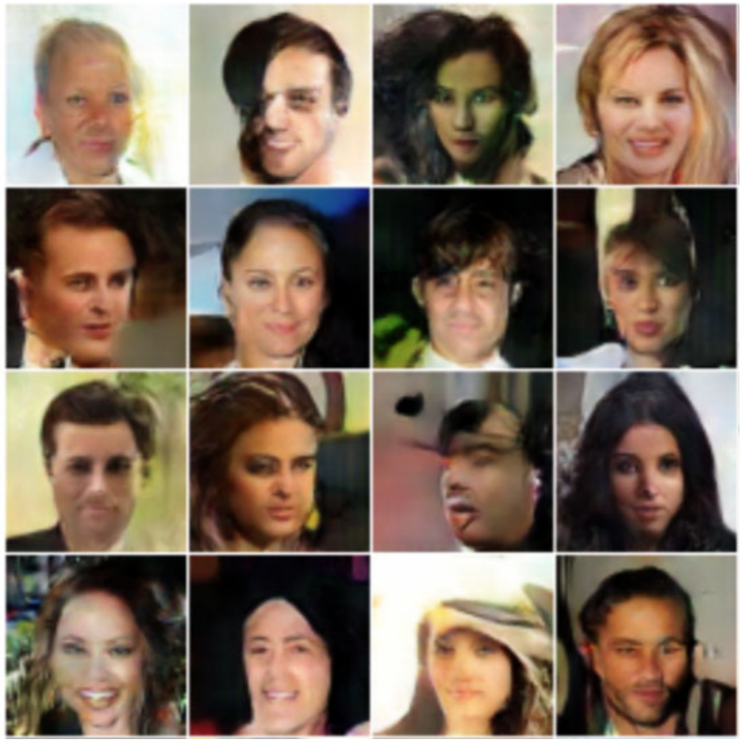


Photo credit: https://github.com/znxlwm/pytorch-generative-model-collections

EB-GAN

- Example Results

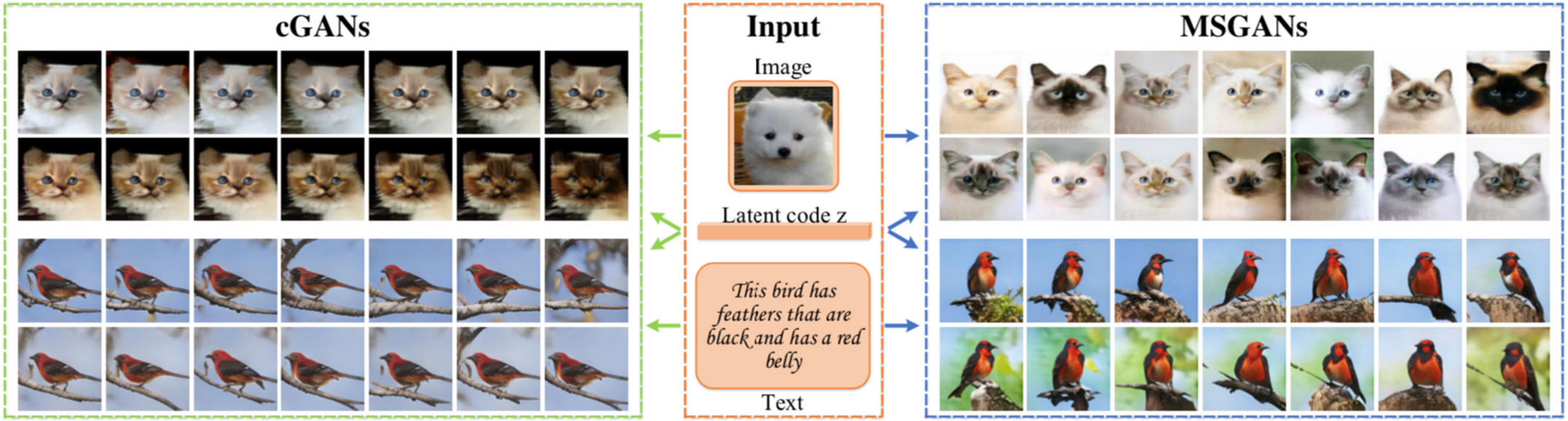
DCGAN



EBGAN

MSGAN

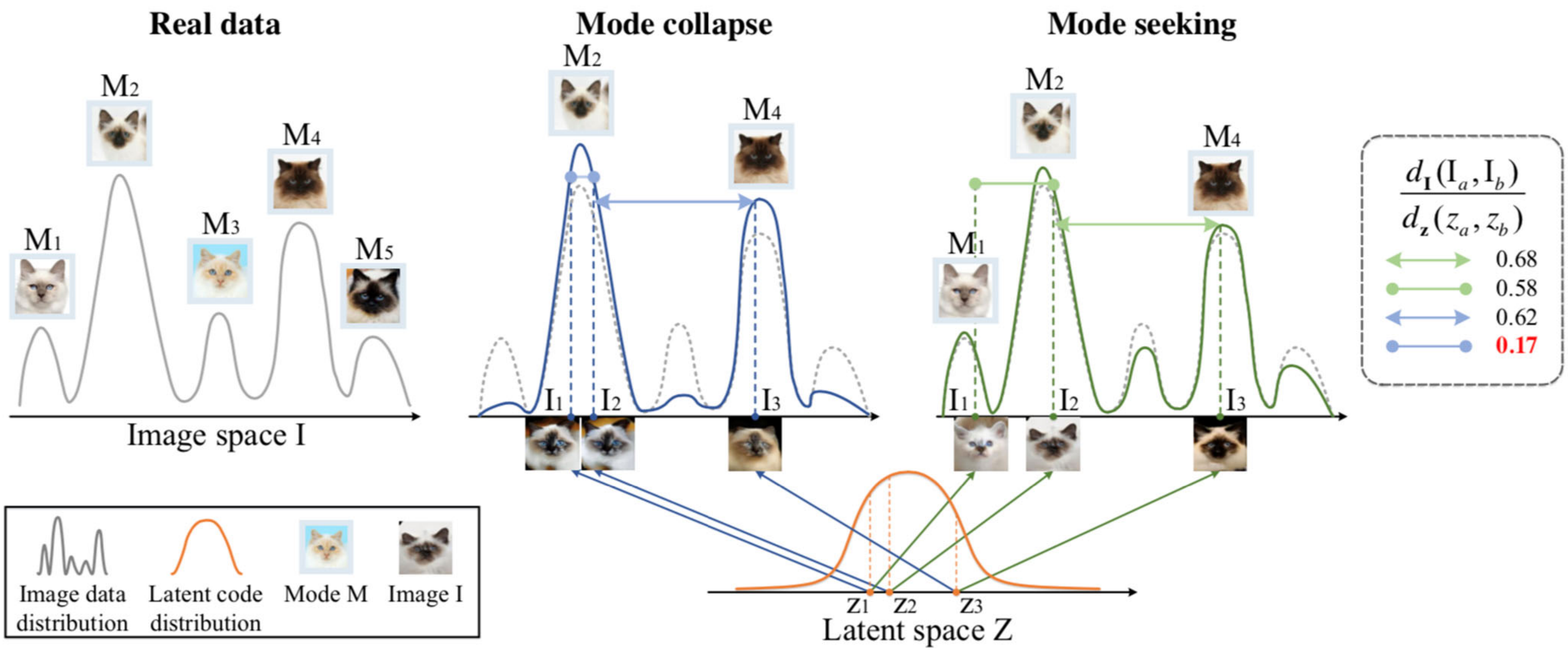
- Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis
- With the goal of producing **diverse** image outputs.
- To address the **mode collapse** issue by conditional GANs



Mao *et al.* " Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis. " *CVPR* 2019

MSGAN (cont'd)

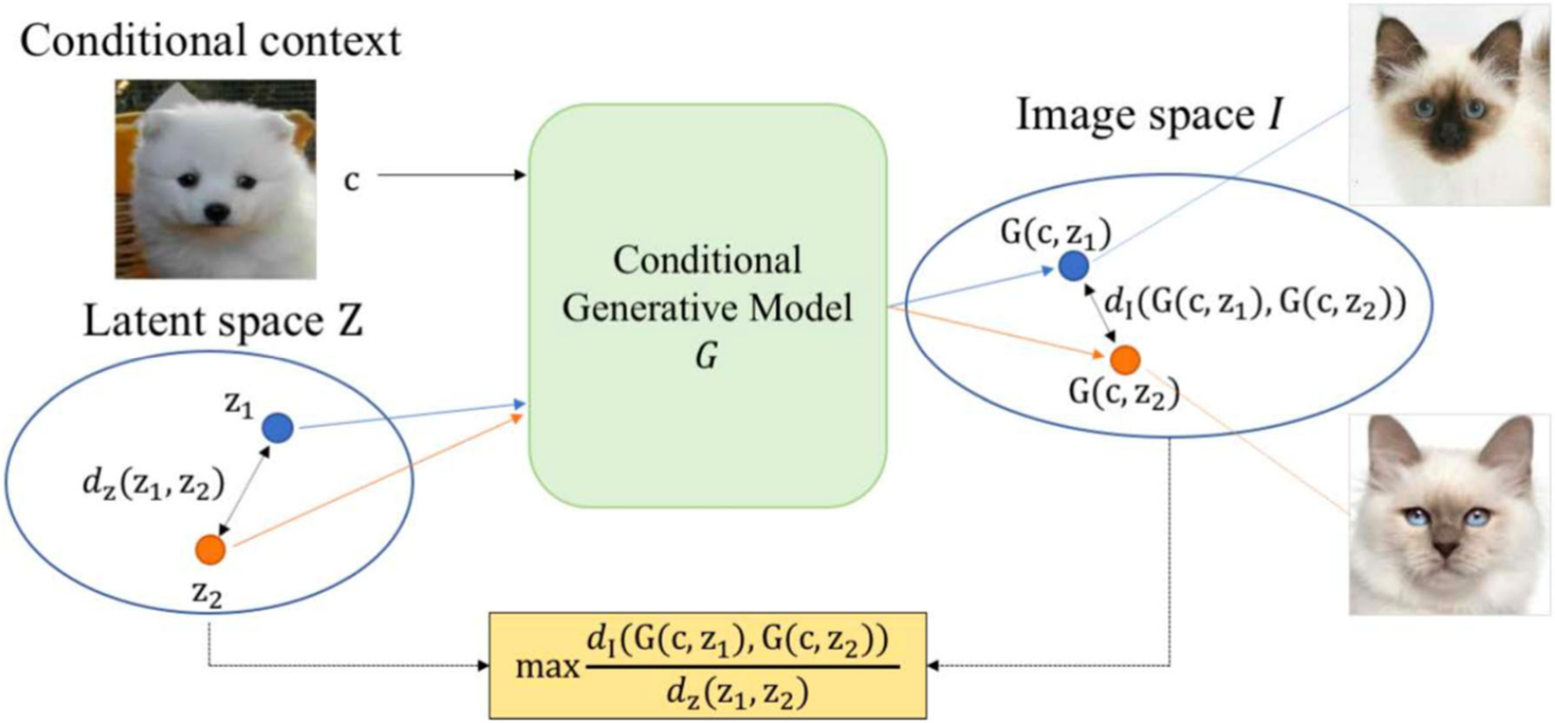
- Motivation (for unconditional GAN)



Mao *et al.* " Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis. " *CVPR* 2019

MSGAN (cont'd)

- Proposed Regularization (for conditional GAN)



Mao *et al.* " Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis. " *CVPR* 2019

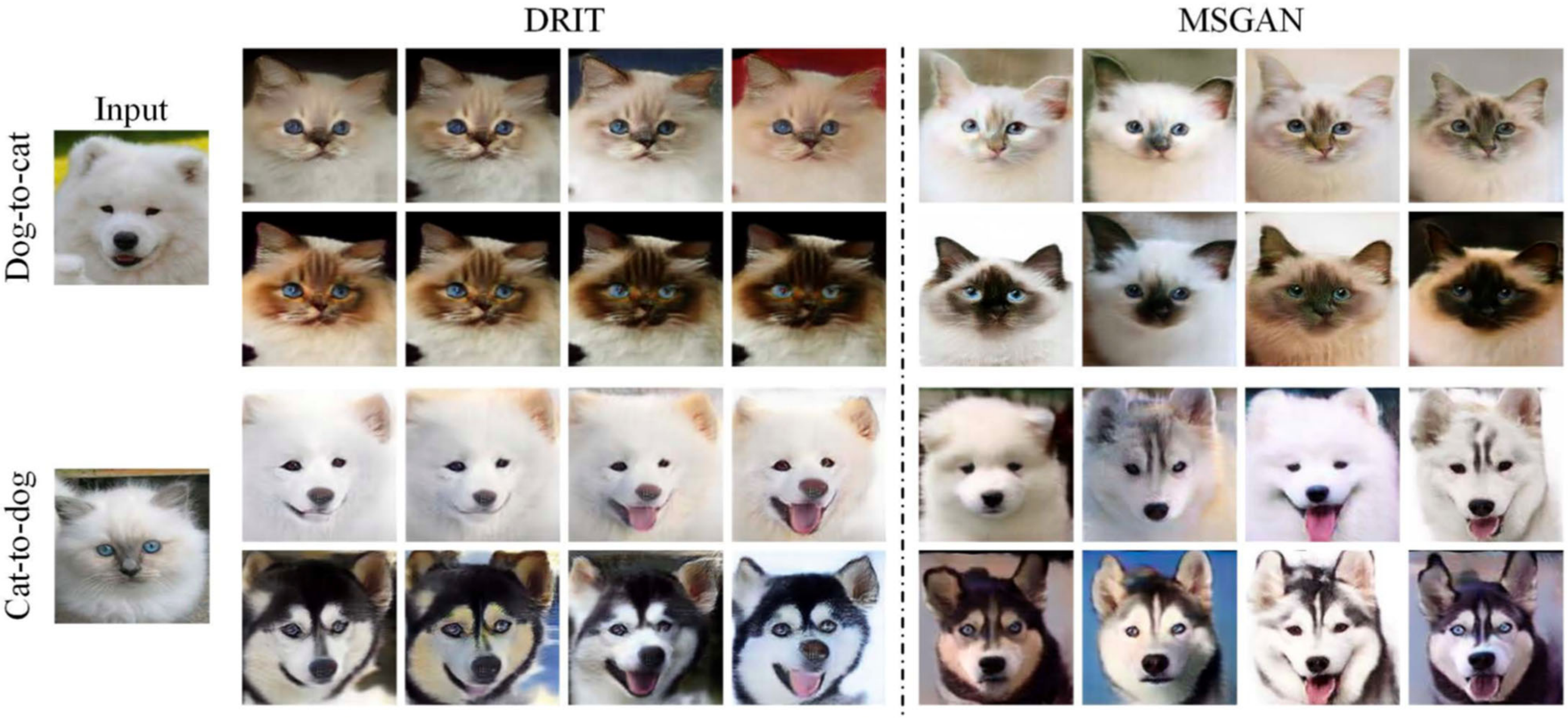
MSGAN (cont'd)

- Qualitative results
  - Conditioned on paired images

Mao *et al.* " Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis. " *CVPR* 2019

MSGAN (cont'd)

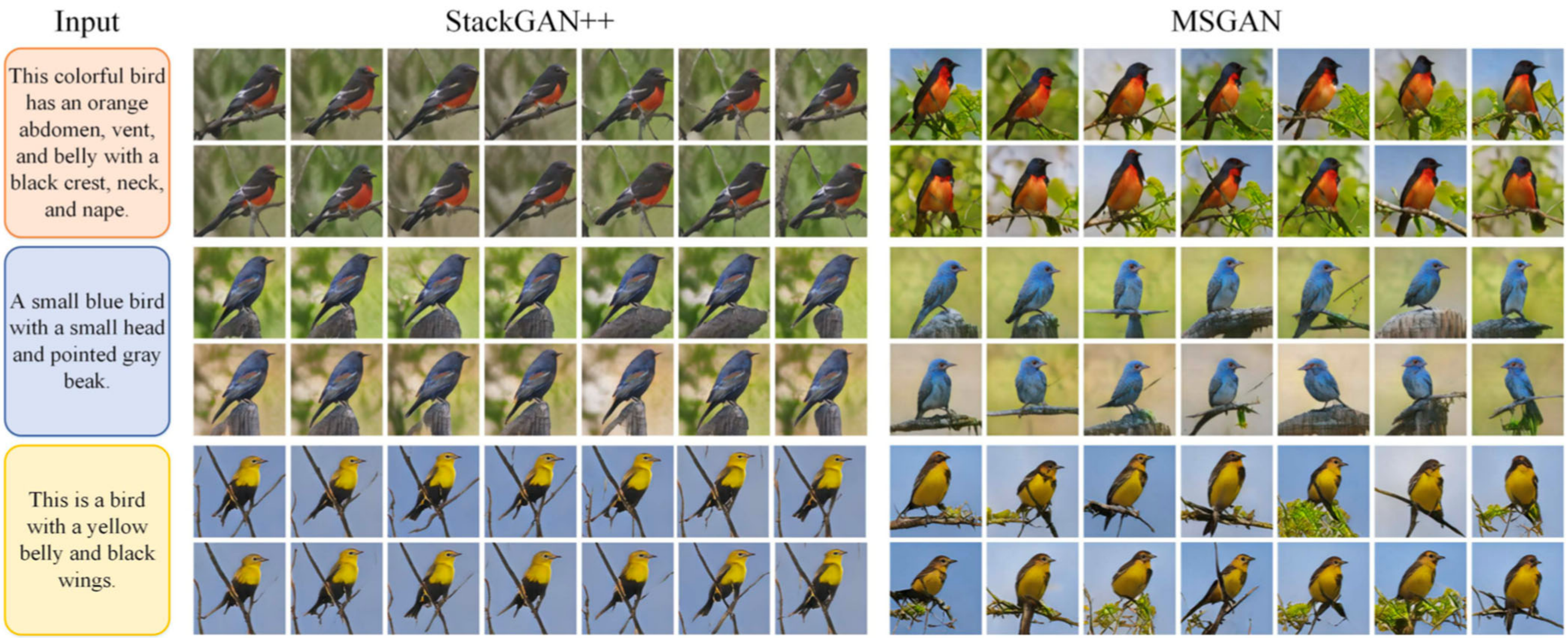
- Qualitative results
  - Conditioned on unpaired images



Mao *et al.* " Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis. " *CVPR* 2019

MSGAN (cont'd)

- Qualitative results
  - Conditioned on text (will talk about Vision & Language later this semester)



Mao *et al.* " Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis. " *CVPR* 2019

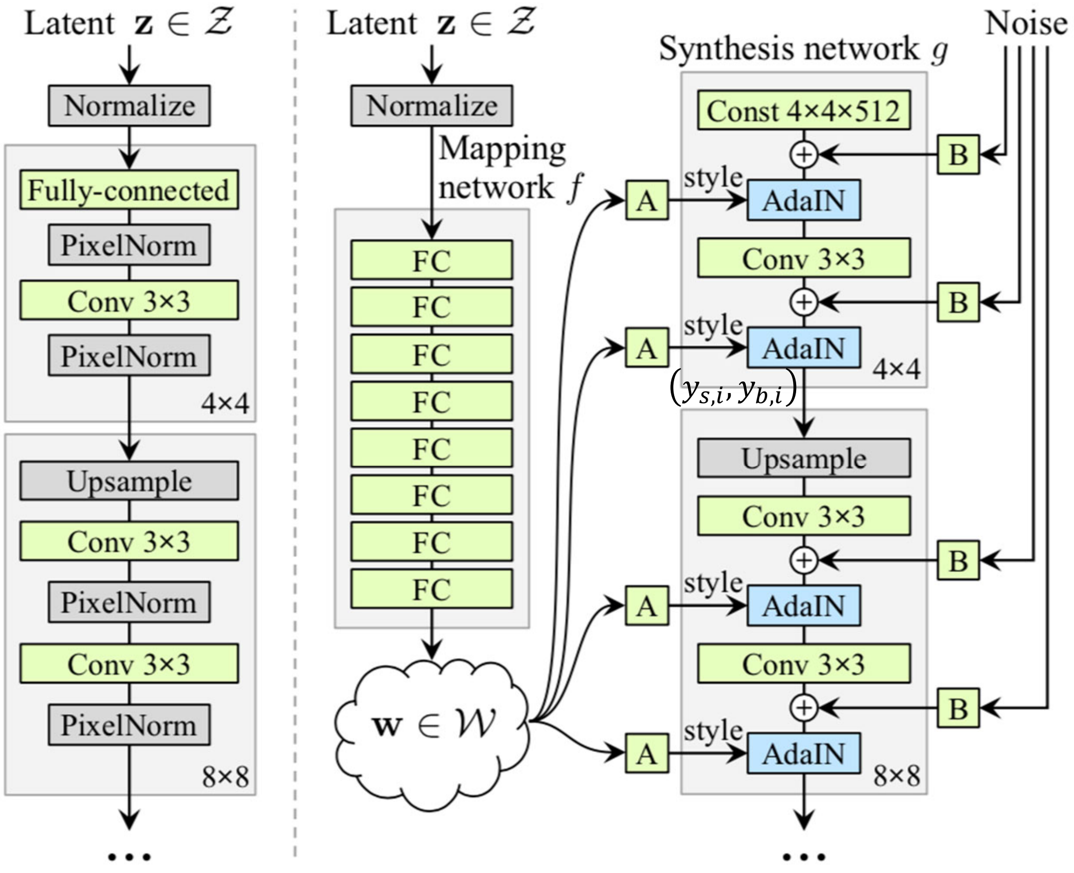
Style-based GAN (if time permits)

- A Style-Based Generator Architecture for Generative Adversarial Networks (CVPR'19)
- Design **style-based generator** to achieve **high-resolution** image synthesis
- No particular designs on loss functions, regularization, and hyper-parameters

Karras *et al.* " A Style-Based Generator Architecture for Generative Adversarial Networks. " *CVPR* 2019

Style-based GAN (cont'd)

- Style-based generator



(a) Traditional (b) Style-based generator

A : Affine transformation

B : Per-channel **scaling factors** to the noise input

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

where $(y_{s,i}, y_{b,i})$ are the outputs of *Affine transformation* A

<u>Mapping network & Affine transformations</u>
- a way to draw samples for each style from a learned distribution

<u>Synthesis network</u>
- a way to generate a novel image based on a collection of styles

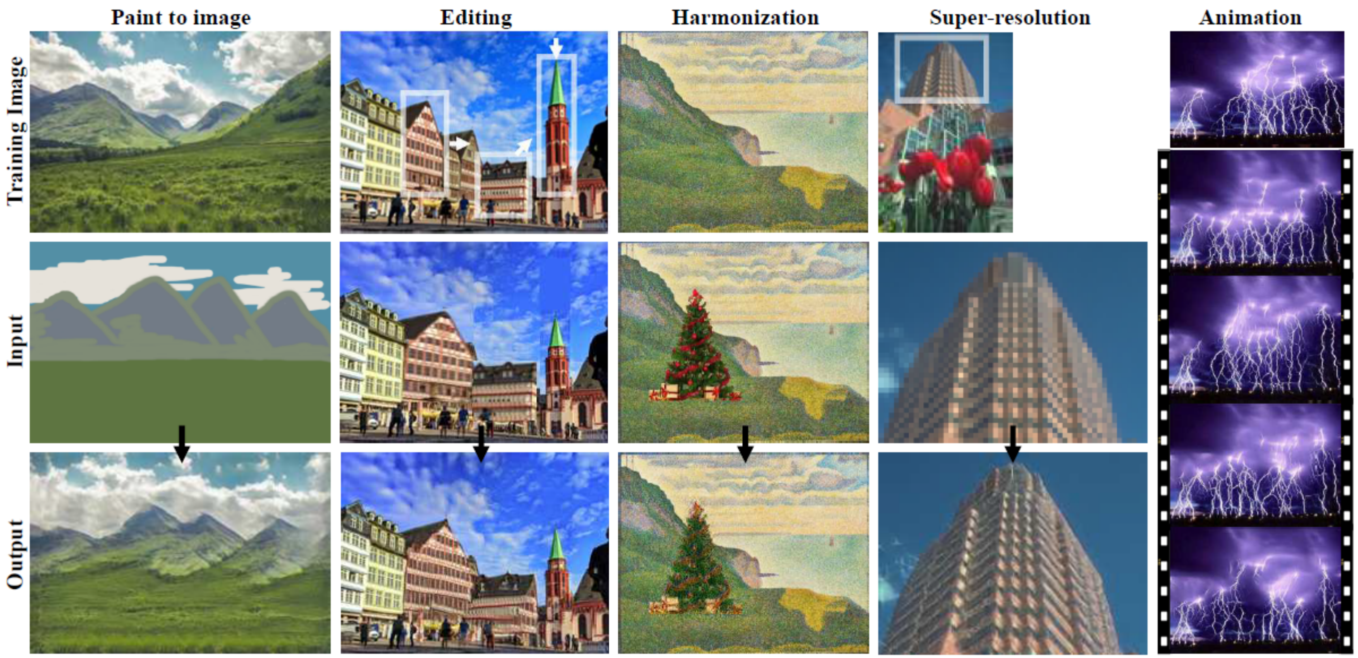
Karras *et al.* " A Style-Based Generator Architecture for Generative Adversarial Networks. " *CVPR* 2019

Style-based GAN (cont'd)

- Qualitative results

SinGAN:
Learning a Generative Model from a Single Natural Image

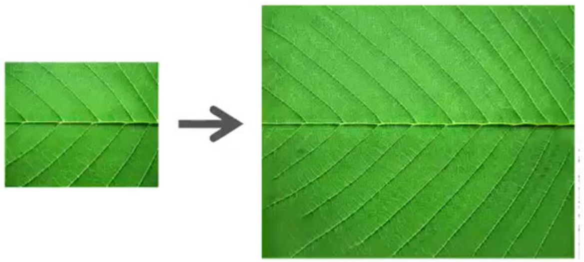
- ICCV 2019 Best Paper Award
- Remarks:
    - Learning from a **single image**
    - Handle **multiple image manipulation tasks**
        - Super-resolution, style conversion, harmonization, image editing, et.

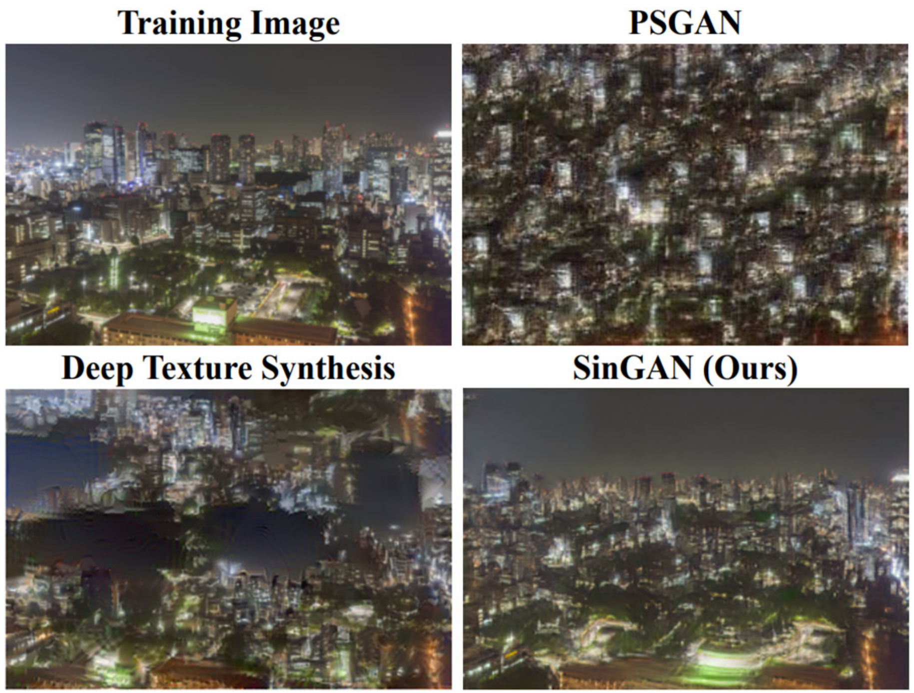


Shaham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV 2019

SinGAN:
Learning a Generative Model from a Single Natural Image

- Related Works
  - While single-image based learning models exist,
    most existing methods are designed to handle textural images but not natural ones.

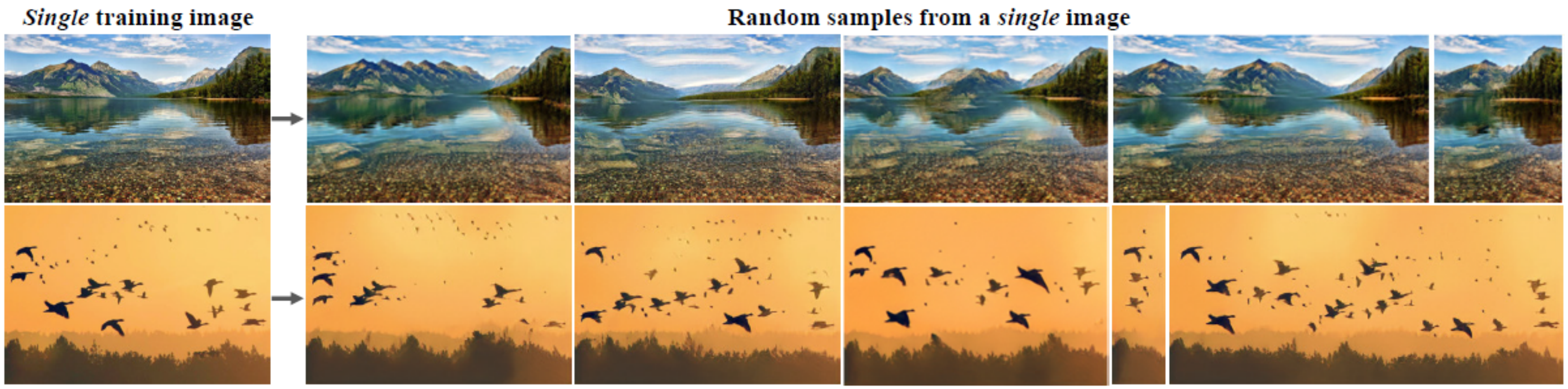




Shaham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV 2019

SinGAN:
Learning a Generative Model from a Single Natural Image

- Goal
  - Output images with arbitrary sizes and aspect ratios (via fully conv models) by changing dimensions of noise and the input size



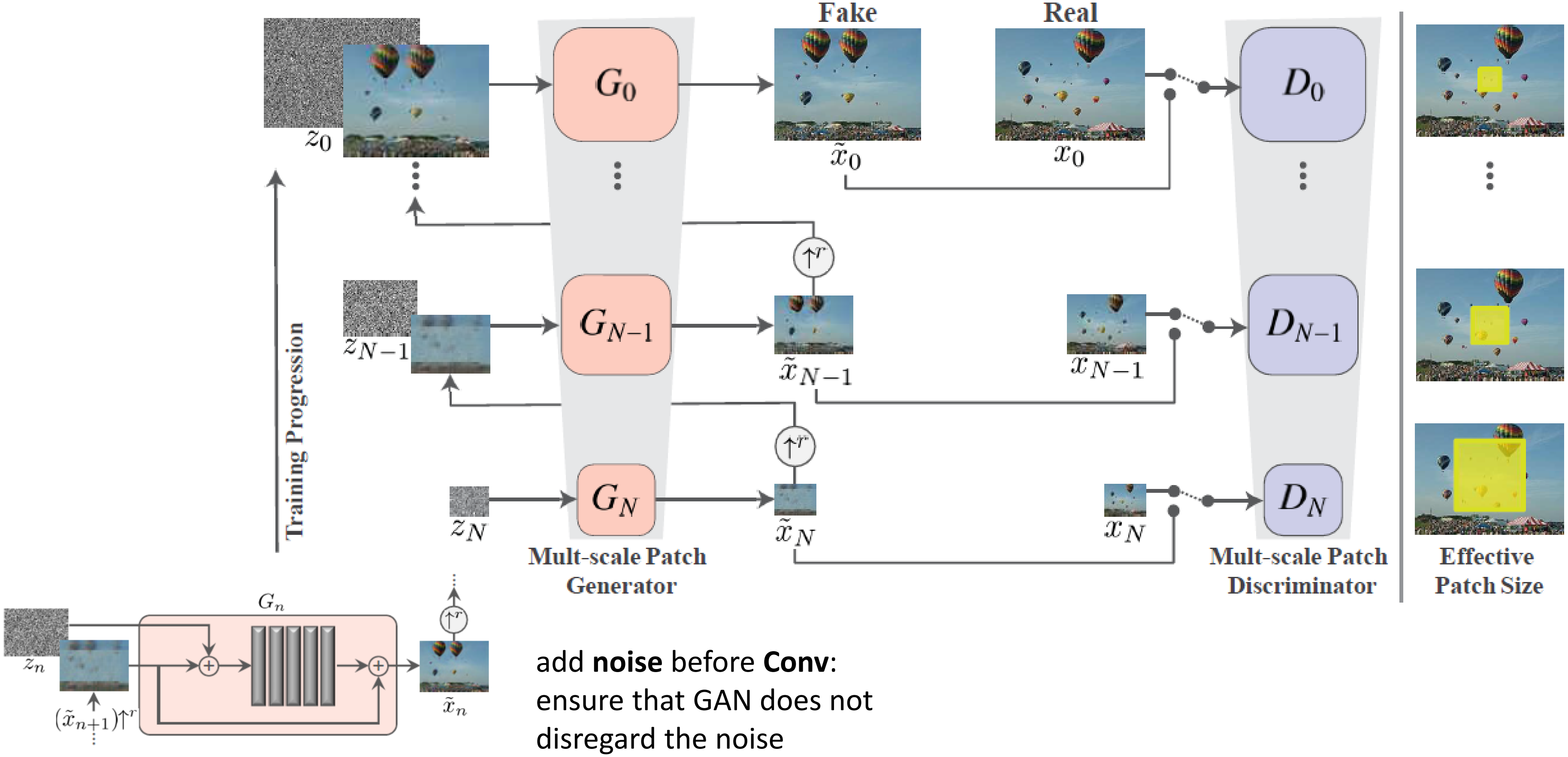
Shaham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV 2019

SinGAN:
Learning a Generative Model from a Single Natural Image

- Framework

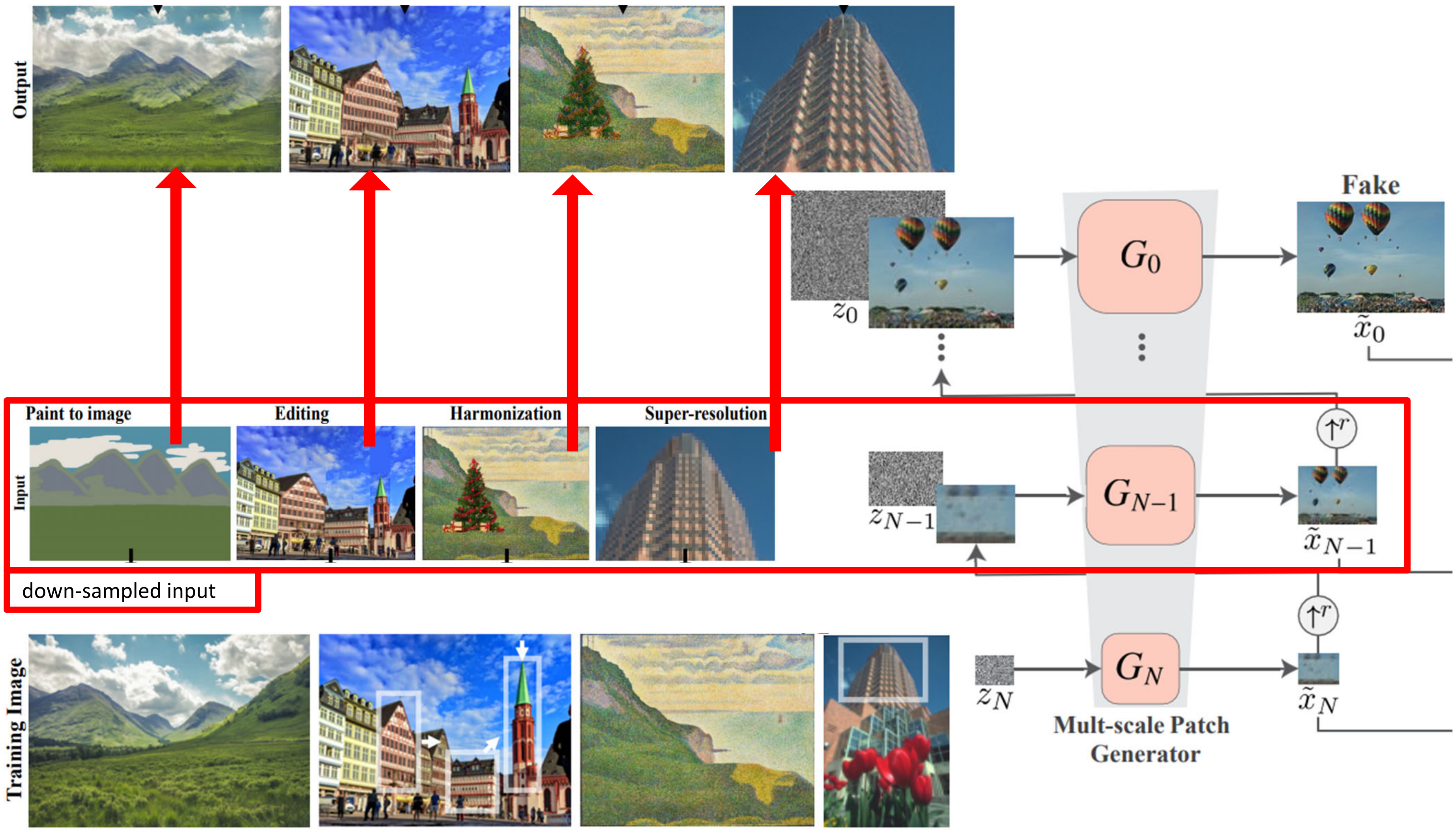
$$\min_{G_n} \max_{D_n} \mathcal{L}_{\text{adv}}(G_n, D_n) + \alpha \mathcal{L}_{\text{rec}}(G_n)$$

fix kernel (receptive field) size at each scale:
capture structures of decreasing size as we go up



add **noise** before **Conv**:
ensure that GAN does not
disregard the noise

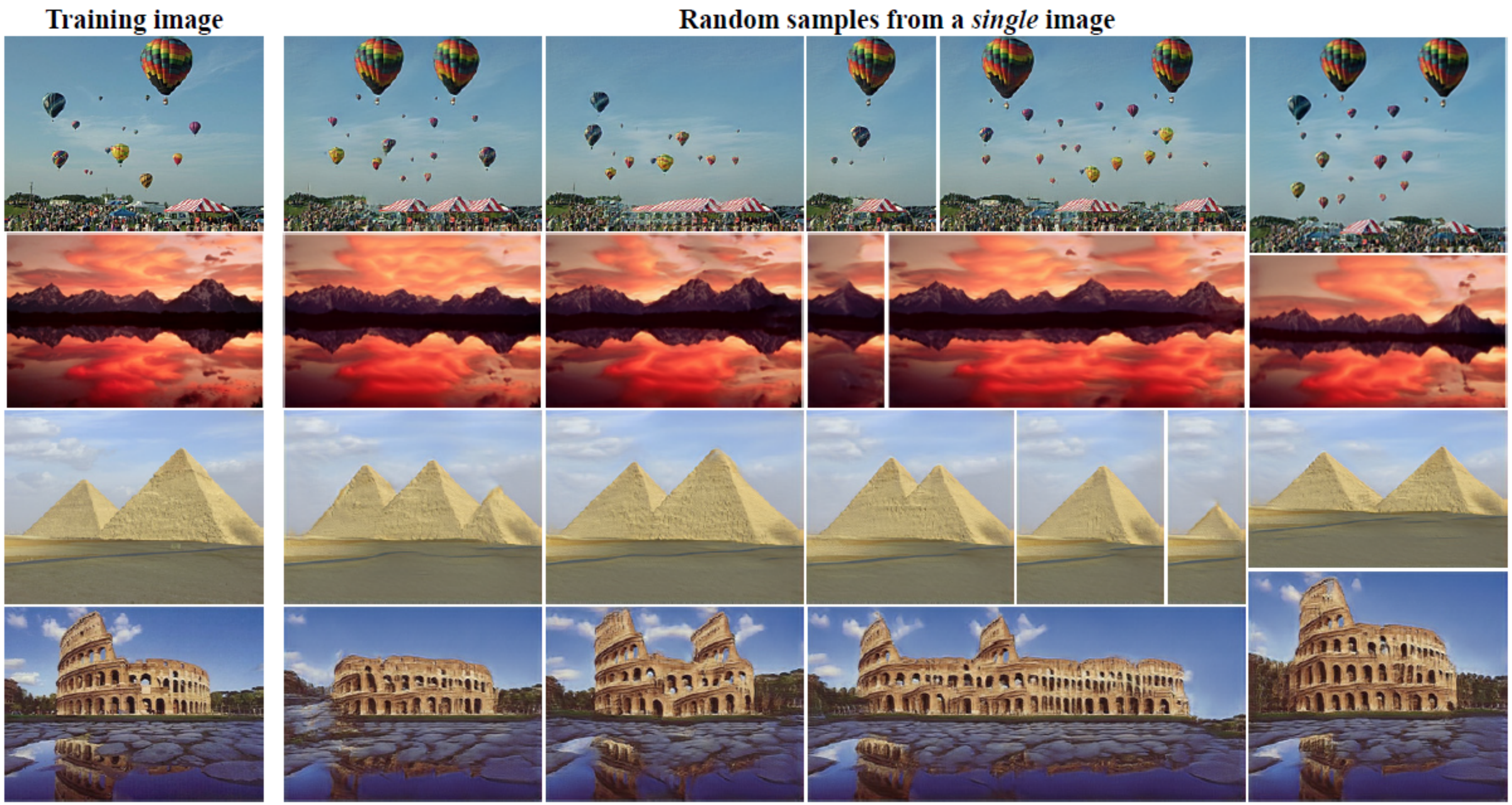
Inference Stage for SinGAN



Shaham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV 2019

SinGAN:
Learning a Generative Model from a Single Natural Image

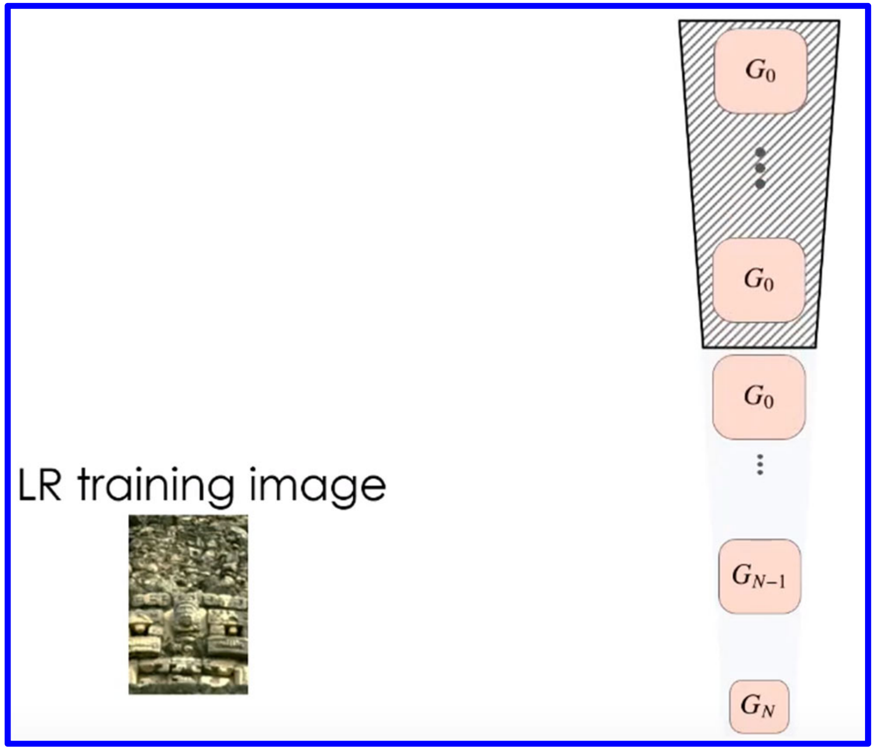
- Random image generation

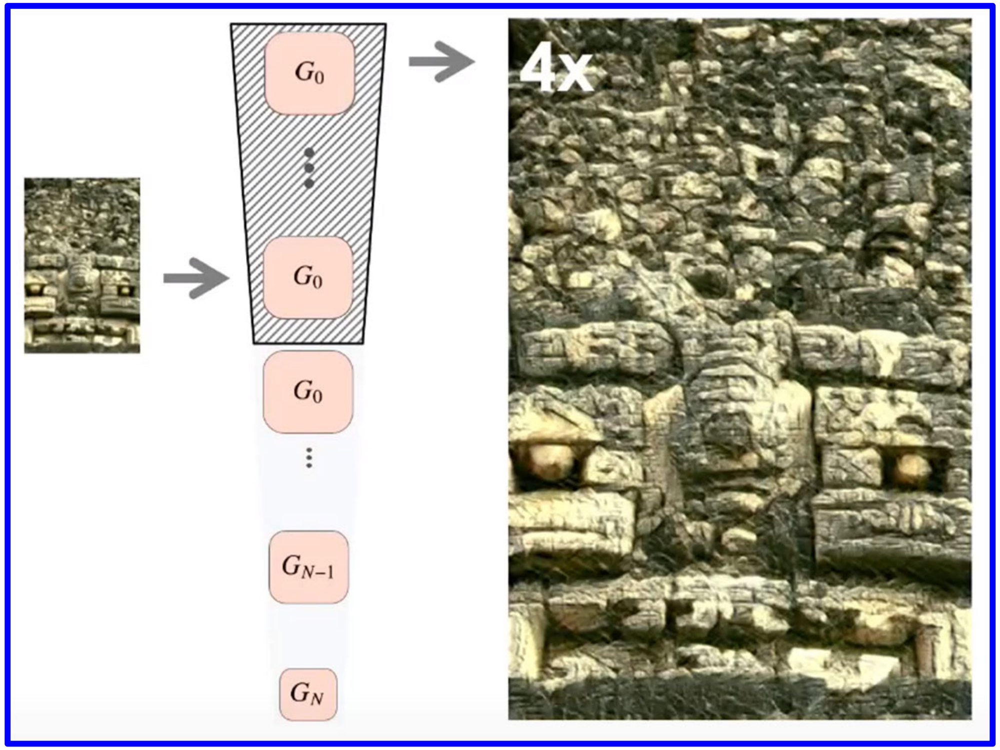


Shaham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV 2019

SinGAN:
Learning a Generative Model from a Single Natural Image

- Super-Resolution

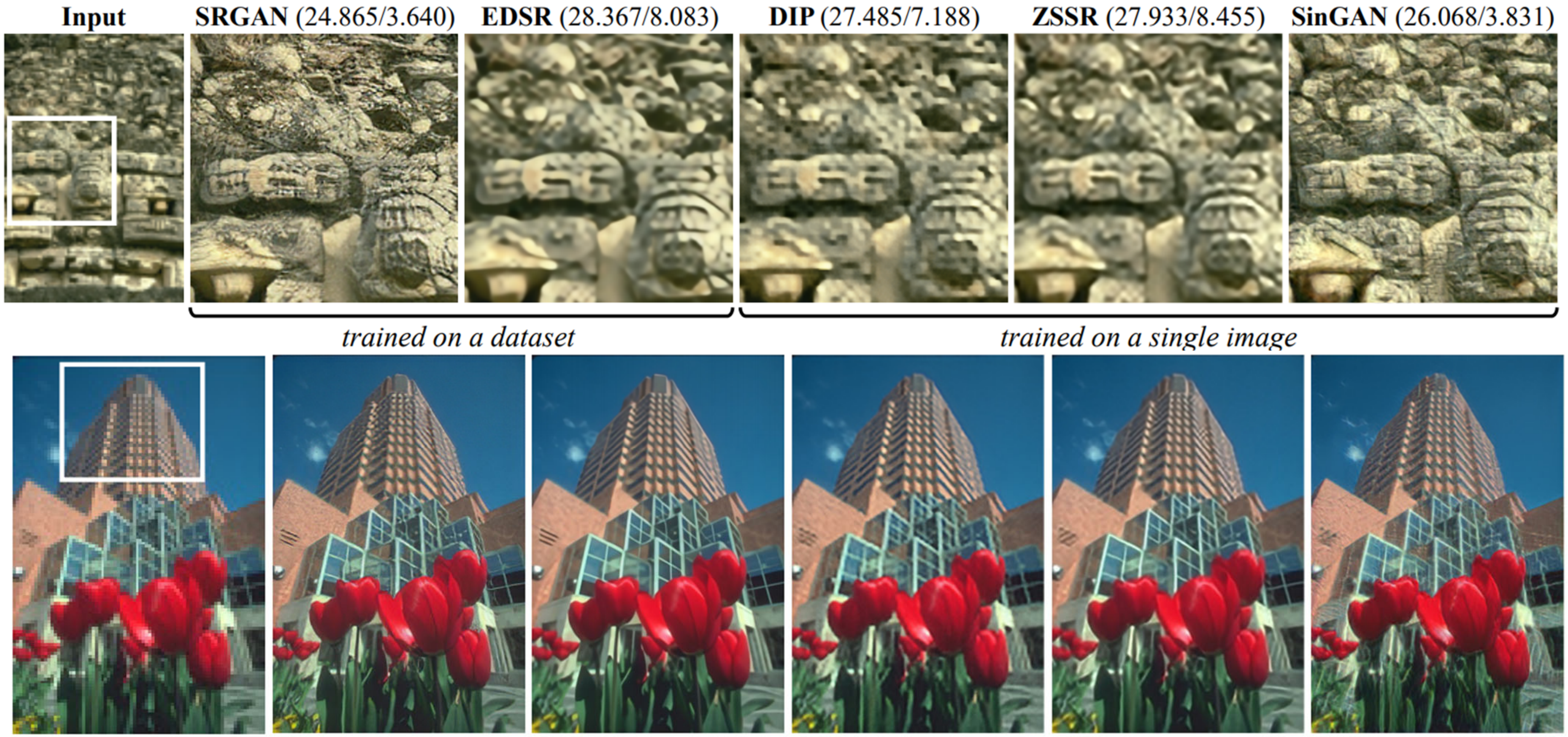




SinGAN:
Learning a Generative Model from a Single Natural Image

- Super-Resolution

	External methods		Internal Methods		
	SRGAN	EDSR	DIP	ZSSR	SinGAN
NIQE	3.4	6.5	6.3	7.1	3.7



Shaham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV 2019

SinGAN:
Learning a Generative Model from a Single Natural Image

- Paint-to-Image

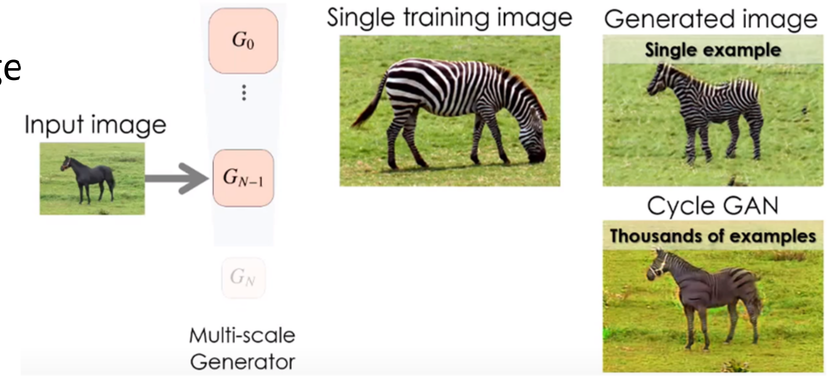
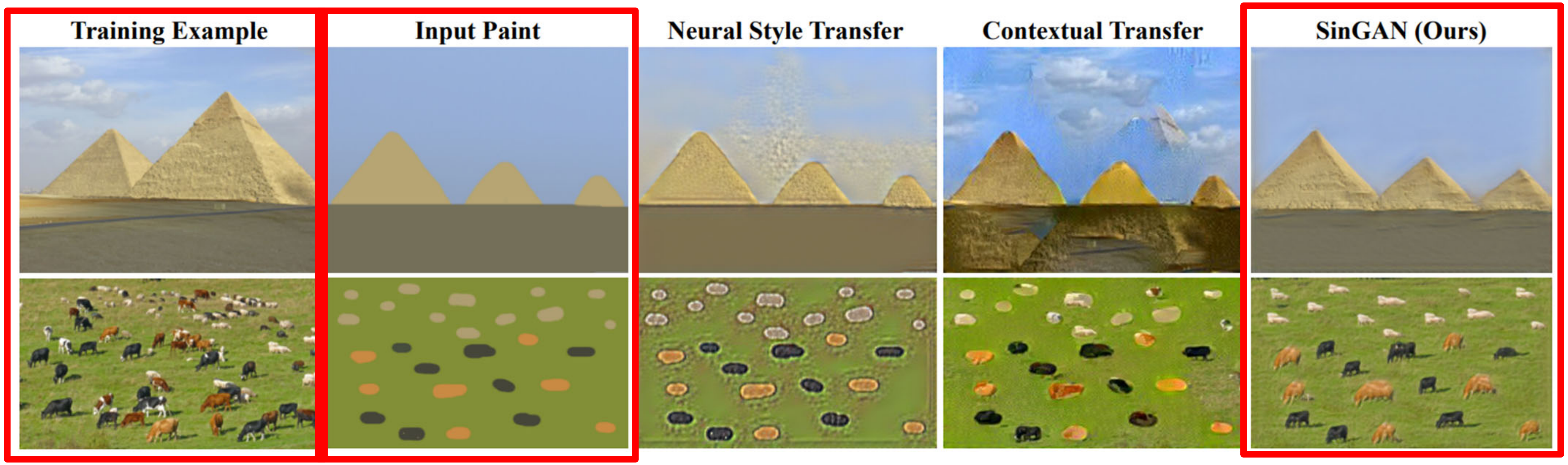


Image	Injection scale	Total number of scales
Balloons1	$n = 7$	$N = 9$
Balloons2	$n = 5$	$N = 9$
Starry night	$n = 6$	$N = 8$
Rock	$n = 6$	$N = 8$
Tree	$n = 6$	$N = 8$
Birds	$n = 5$	$N = 7$
View (Fig. 2, main text)	$n = 7$	$N = 8$
Pyramids (Fig. 11, main text)	$n = 6$	$N = 8$
cows (Fig. 11, main text)	$n = 5$	$N = 7$



SinGAN:
Learning a Generative Model from a Single Natural Image

- Harmonization

Image	Injection scale	Total number of scales
Tree (also Fig. 2, main text)	$n = 1$	$N = 9$
Two Dolphins (also Fig. 13, main text)	$n = 3$	$N = 9$
Single Dolphin	$n = 3$	$N = 9$
Fox	$n = 2$	$N = 8$
Airplane	$n = 2$	$N = 8$
Butterfly	$n = 2$	$N = 8$
Eagle	$n = 2$	$N = 8$
Spaceship (also Fig. 13, main text)	$n = 3$	$N = 8$
Hat	$n = 4$	$N = 9$
Lemon	$n = 3$	$N = 7$
Cat	$n = 2$	$N = 8$

Shaham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV 2019

SinGAN:
Learning a Generative Model from a Single Natural Image

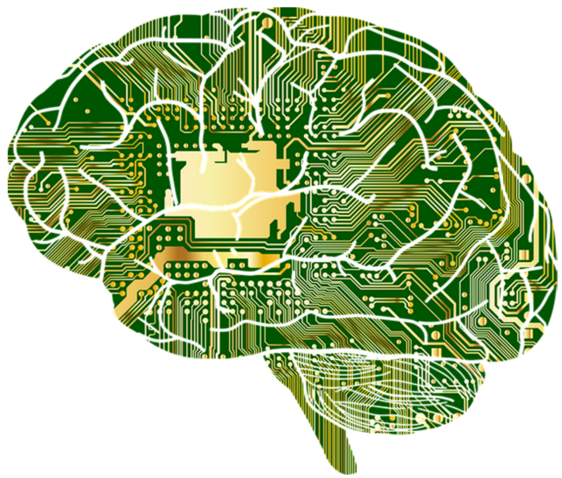
- Editing

Image	Injection scale	Total number of scales
Rock1	$n = 5$	$N = 7$
Rock2	$n = 5$	$N = 7$
Rock3 (also Fig. 12, main text)	$n = 5$	$N = 7$
Tree	$n = 7$	$N = 9$
Mountain	$n = 4$	$N = 8$
Red cliff	$n = 5$	$N = 9$
Hay	$n = 6$	$N = 9$

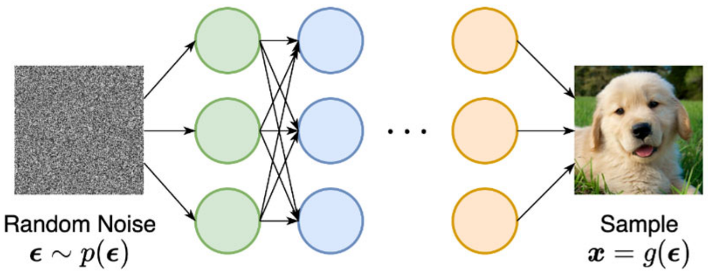
Shaham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV 2019

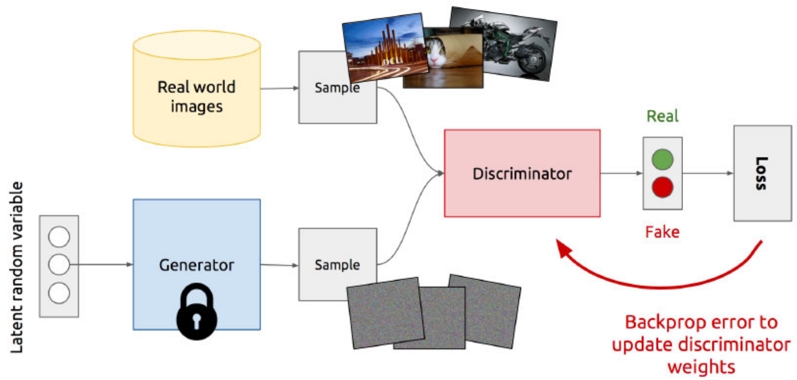
What’s to Be Covered Today…

- Generative Models
    - Auto-Encoder vs. Variational Auto-Encoder
    - Generative Adversarial Network (GAN)
    - Diffusion Model

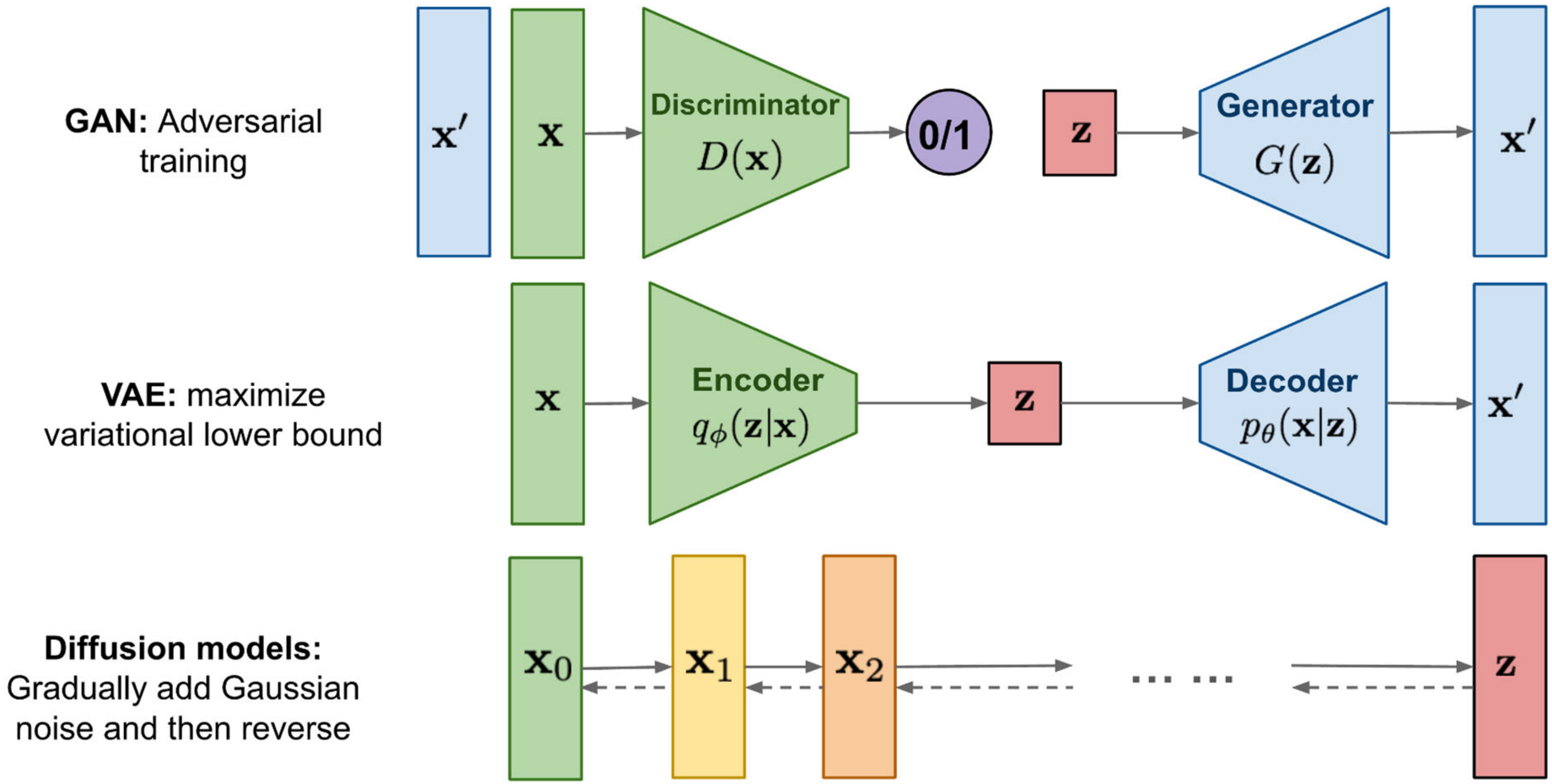


- HW #1 is due Oct. 10th Mon 23:59
- HW #2 will be out next week…





From VAE to Diffusion Model



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Denoising Diffusion Models

- Emerging as powerful generative models
  - Unconditional image synthesis
  - Conditional image synthesis
  - Outperforms GANs

Diffusion Models Beat GANs on Image Synthesis, Dhariwai & Nochol, OpenAI, 2021

Cascaded Diffusion Models for High Fidelity Image Generation, Ho et al., Google, 2021

Slide credit: Kreis, Gao, & Vahdat

Denoising Diffusion Models

- Emerging as powerful generative models
  - Unconditional image synthesis
  - Conditional image synthesis
  - Outperforms GANs

DALL·E 2

“a teddy bear on a skateboard in times square”

Diffusion Models Beat GANs on Image Synthesis, Dhariwai & Nochol, OpenAI, 2021

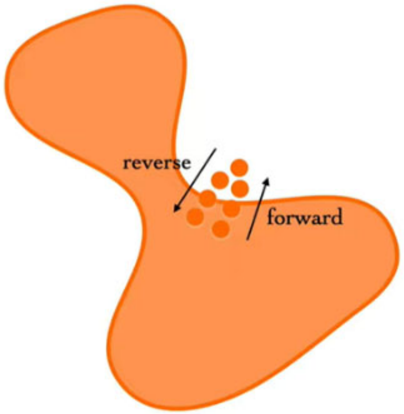
Imagen

A group of teddy bears in suit in a corporate office celebrating the birthday of their friend. There is a pizza cake on the desk.

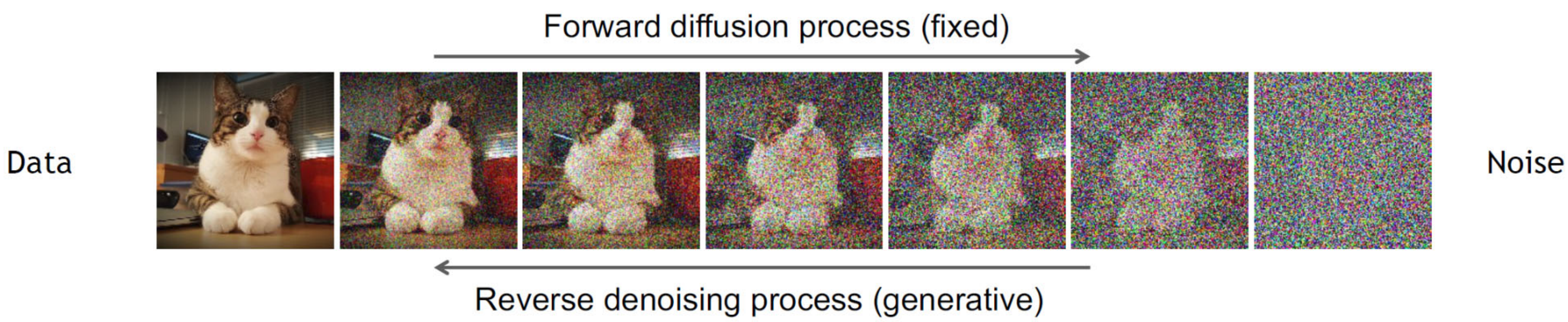
Cascaded Diffusion Models for High Fidelity Image Generation, Ho et al., Google, 2021

Denoising Diffusion Models:

Learning to generate by denoising



- 2 processes required for training:
  - Forward diffusion process – gradually add noise to input
  - Reverse diffusion process – learns to generate/restore data by denoising (typically implemented via a U-net)
  - Comments about noise scheduling (see next slide)

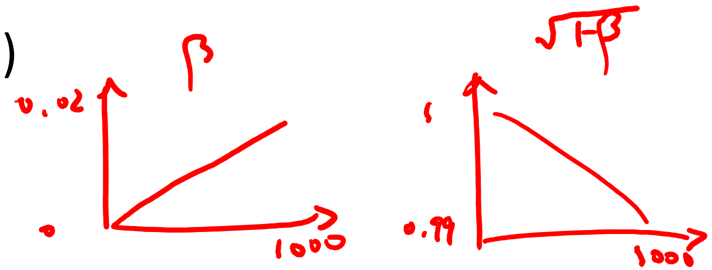


Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

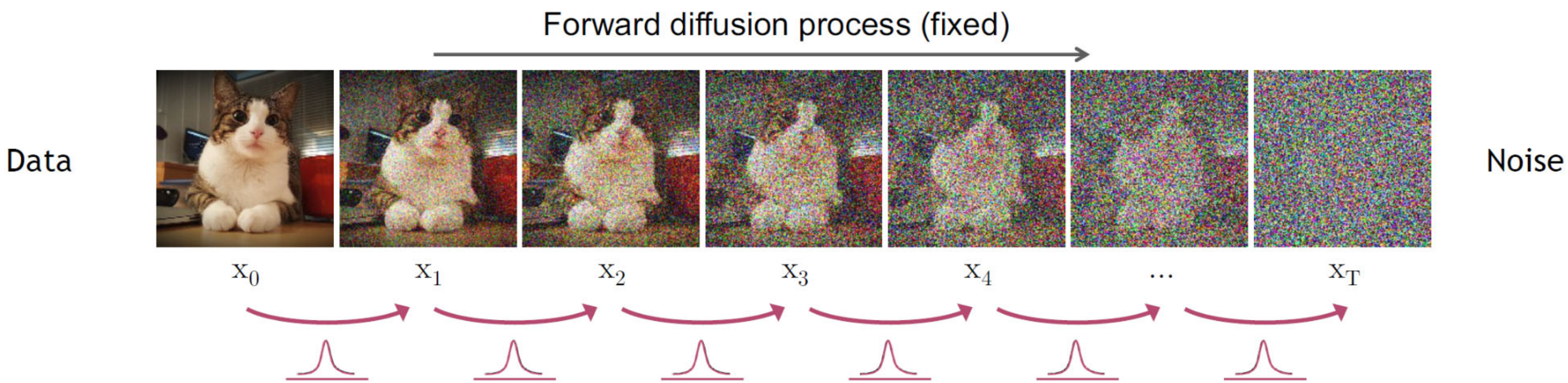
Slide credit: Kreis, Gao, & Vahdat

Denoising Diffusion Models:

Learning to generate by denoising (cont'd)



- Forward diffusion process
  - Gradually add noise to the input in T steps
  - Recall that x_0 denotes clean input image, and x_T is the final noisy one.
  - Comments on $q(x_t|x_{t-1})$



Normal Dist.   mean   var.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad \text{(joint)}$$

output

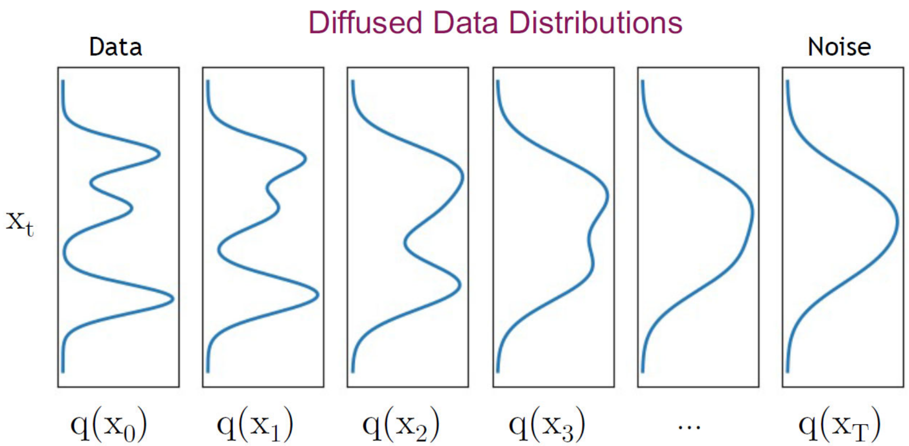
variance schedule (hyperparameter)

Slide credit: Kreis, Gao, & Vahdat

Denoising Diffusion Models:

Learning to generate by denoising (cont'd)

- Forward diffusion process
  - Gradually add noise to the input in T steps (cont'd)
  - Diffusion kernel
  - So what happens to data distribution during this process?



$$\underbrace{q(\mathbf{x}_t)}_{\text{Diffused data dist.}} = \int \underbrace{q(\mathbf{x}_0, \mathbf{x}_t)}_{\text{Joint dist.}}\, d\mathbf{x}_0 = \int \underbrace{q(\mathbf{x}_0)}_{\text{Input data dist.}}\, \underbrace{q(\mathbf{x}_t|\mathbf{x}_0)}_{\text{Diffusion kernel}}\, d\mathbf{x}_0$$

The diffusion kernel is Gaussian convolution.

Denoising Diffusion Models:
Learning to generate by denoising (cont'd)

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t, \sqrt{1-\beta_t}\, x_{t-1}, \beta_t I)$$
$$= \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon$$
$$= \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \epsilon$$
$$= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}}\, \epsilon$$
$$= \dots$$

- Forward diffusion process
  - Gradually add noise to the input in T steps
  - Diffusion kernel:

Forward diffusion process (fixed)

Data

Noise

x_0 x_1 x_2 x_3 x_4 $\dots$ x_T

$\alpha_t = 1 - \beta_t$

Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s) = \prod_{s=1}^{t} \alpha_s$ ➡ $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$ (Diffusion Kernel)

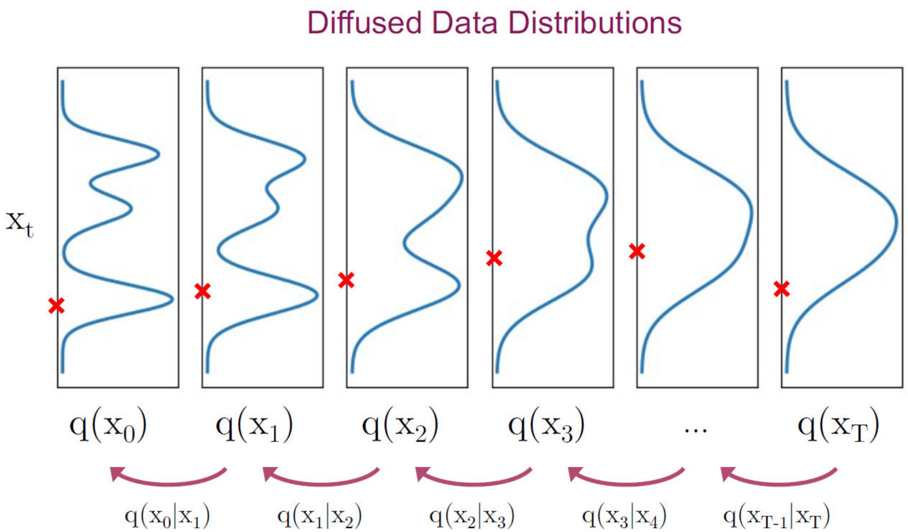
For sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

β_t values schedule (i.e., the noise schedule) is designed such that $\bar{\alpha}_T \to 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Slide credit: Kreis, Gao, & Vahdat

Denoising Diffusion Models:
Learning to generate by denoising (cont'd)

- Generative learning by denoising
  - Diffusion parameters are designed such that: $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$



Generation:

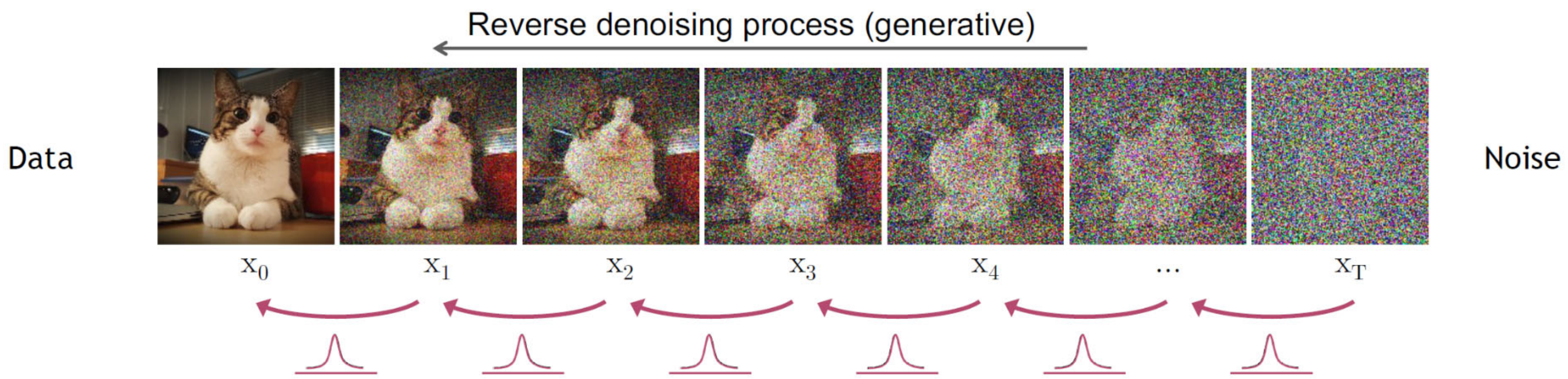
Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$

  - Unfortunately, $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is intractable.
    We approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ by Normal distribution by setting small β_t in each step

Denoising Diffusion Models:
Learning to generate by denoising (cont'd)

- Reverse diffusion process
  - Learn to denoise in T steps
  - Let the model θ predict $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$
  - To conclude the learning process first, we need to predict the noise in image.

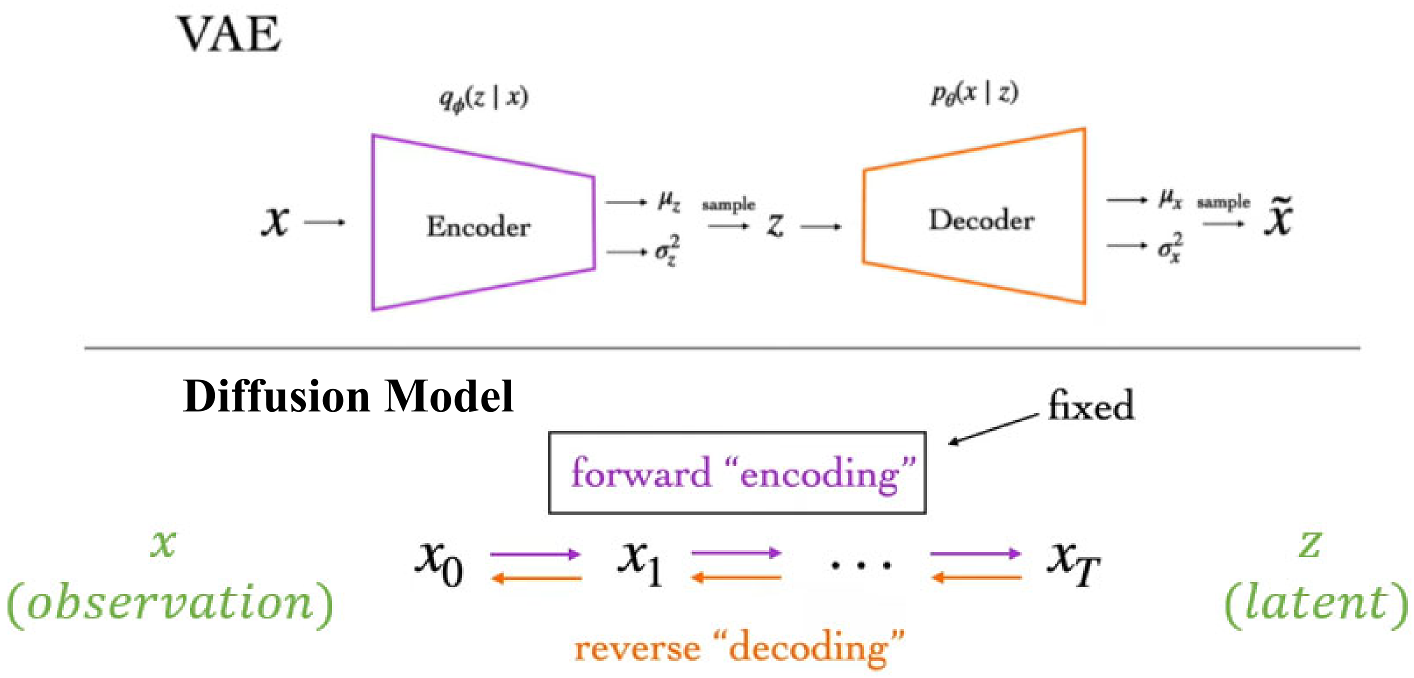


$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}_{\substack{\text{Trainable network} \\ \text{(U-net, Denoising Autoencoder)}}}, \sigma_t^2\mathbf{I})$$

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

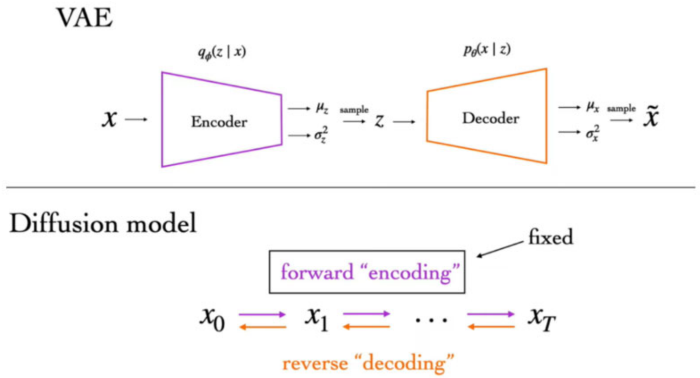
Learning of Diffusion Models

- $\boxed{\log p_\theta(x) \geq \text{ variational lower bound}}$



$$\log p_\theta(x) \geq \text{ variational lower bound}$$

Learning of Diffusion Models



- $\boxed{\log p_\theta(x) \geq \text{ variational lower bound}}$
  - Recall that we exploit variational bound for optimizing VAE models

$$\log p_\theta(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

vs. $$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] =: L$$

  - In Ho et al. NeurIPS'20, it is shown that

$N(x_{t-1}; \mu_\theta(x_t, t), \beta I)$

$$L = \mathbb{E}_q\left[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\,||\,p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\,||\,p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\right]$$

Can ignore. Why?

$N(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I)$

fixed

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0$$

$$= \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$ (slide #78)

Ref: https://youtu.be/HoKDTa5jHvg

Learning of Diffusion Models (cont'd)

- Recall that

$$L = \mathbb{E}_q\Big[\overset{\text{Not trainable}}{\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\,\|\,p(\mathbf{x}_T))}_{L_T}} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \overset{\{0,\,1,\,\ldots,\,255] \to [-1,1]}{\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}}\Big]$$

- Still working on it…
  - Only care about KL divergence between two Gaussian distributions

$$\begin{cases} q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0)\dfrac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} := \mathcal{N}(x_{t-1};\, \tilde{\mu}_t(x_t, x_0),\, \tilde{\beta}_t I) \\ p_\theta(x_{t-1}\,|\,x_t) := \mathcal{N}(x_{t-1};\, \underset{\text{learned}}{\mu_\theta(x_t,t)},\, \underset{\text{fixed}}{\Sigma_\theta(x_t,t)}) \end{cases}$$

$$\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) \quad (1)$$

(actual μ)

$$\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) \quad (2)$$

(predicted μ)

  - As a result,

$$\frac{1}{2\sigma_t^2}\|(1)-(2)\|^2 \rightarrow \mathbb{E}_{\mathbf{x}_0,\epsilon}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2\right]$$

  - For simplicity, we calculate

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left[\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2\right]$$

Slide credit: Kreis, Gao, & Vahdat
and https://youtu.be/HoKDTa5jHvg

Learning of Diffusion Models

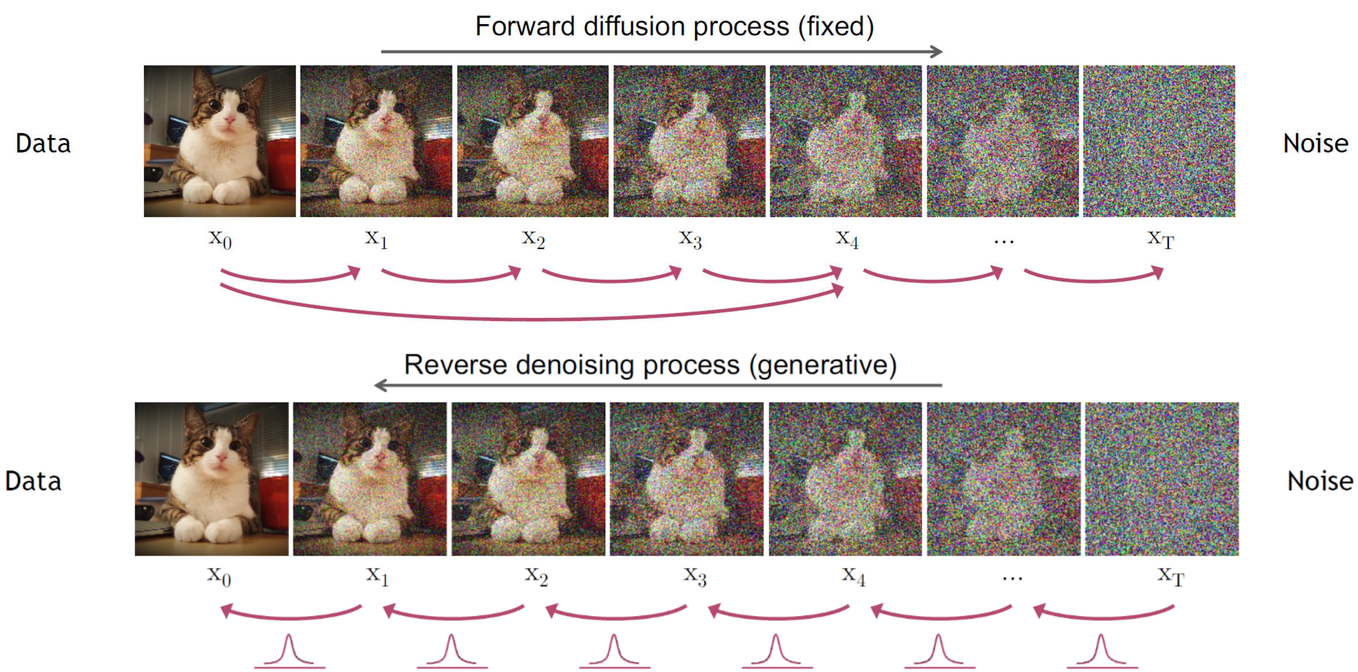
- Summary
  - Training and sample generation

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boxed{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}, t) \right\|^2$$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $$\mathbf{x}_{t-1} = \boxed{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)} + \sigma_t \mathbf{z}$$
end for
return $\mathbf{x}_0$



Slide credit: Kreis, Gao, & Vahdat

Learning of Diffusion Models

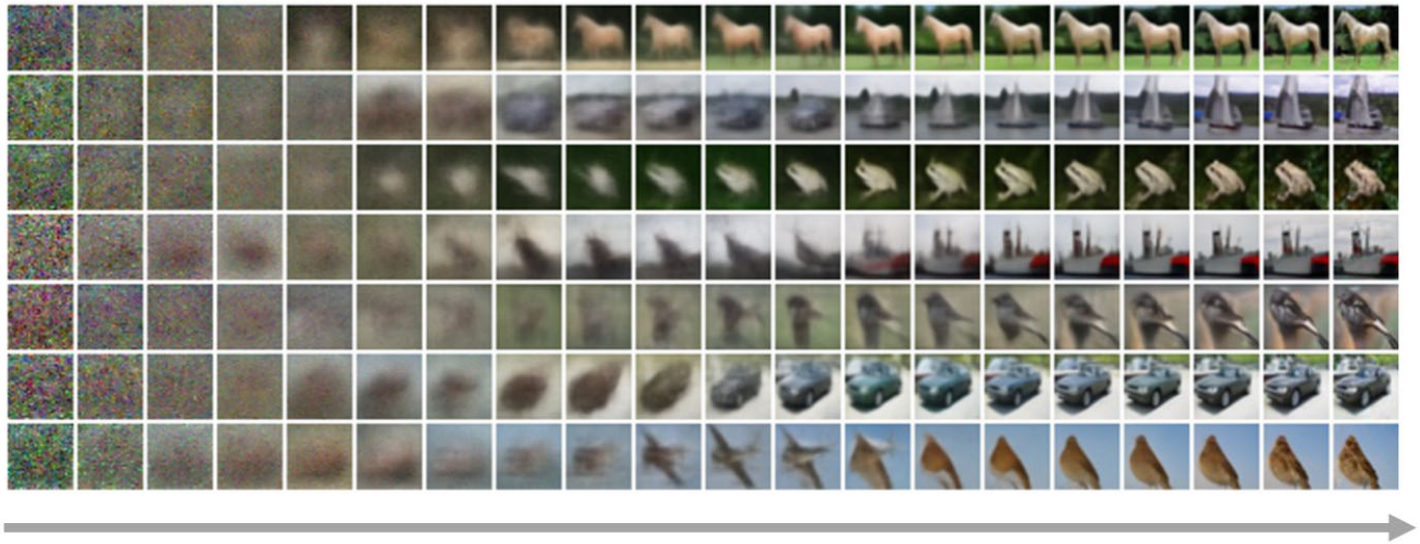
- Summary
  - Training and sample generation

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boxed{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}, t) \right\|^2$$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $$\mathbf{x}_{t-1} = \boxed{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)} + \sigma_t \mathbf{z}$$
end for
return $\mathbf{x}_0$



More steps

Slide credit: Kreis, Gao, & Vahdat
https://medium.com/ai-blog-tw/%E9%82%8A%E5%AF%A6%E4%BD%9C%E9%82%8A%E5%AD%B8%E7%BF%92diffusion-model-%E5%BE%9Eddpm%E7%9A%84%E7%B0%A1%E5%8C%96%E6%A6%82%E5%BF%B5%E7%90%86%E8%A7%A3-4c565a1c09c

What We've Covered Today…

- Generative Models
  - Auto-Encoder vs. Variational Auto-Encoder
  - Generative Adversarial Network (GAN)
  - Diffusion Model

- HW #1 is due Oct. 10th Mon 23:59
- HW #2 will be out next week…