Deep Generative Models

Xiao Wang

Department of Statistics
Purdue University
wangxiao@purdue.edu

Table of contents

Supervised Learning

- Supervised Learning
  - Data: (x, y) — x is predictor, y is label
  - Goal: Learning a function to map $x \to y$
  - Examples: classification, regression, object detection, semantic segmentation, image captioning, etc.

Supervised Learning

Cat

Classification

DOG, DOG, CAT

Object Detection

GRASS, CAT,
TREE, SKY

Semantic Segmentation

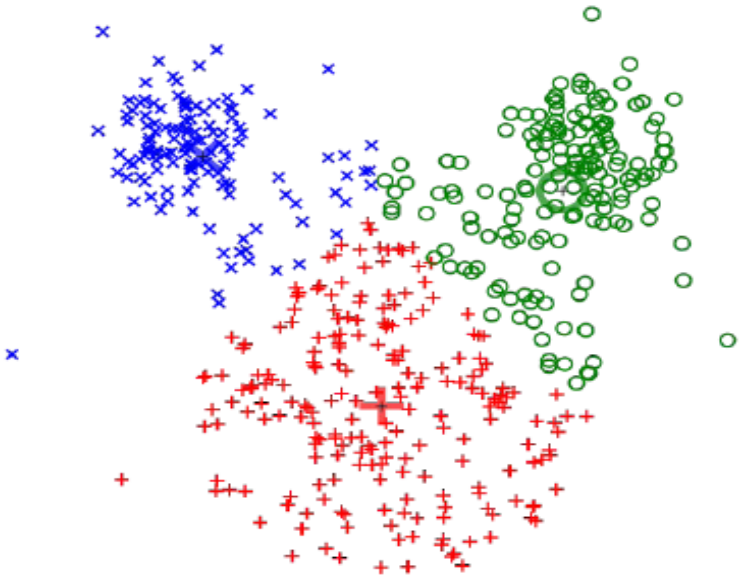
A cat sitting on a suitcase on the floor

Image captioning

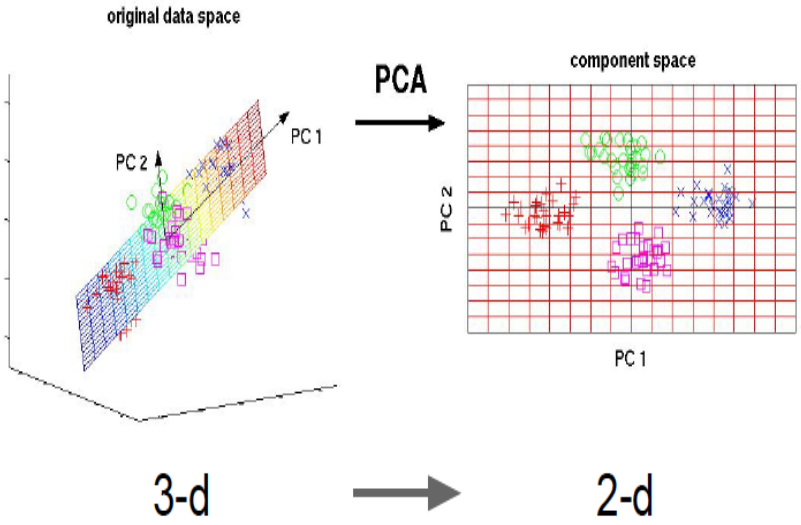
Unsupervised Learning

- Unsupervised Learning
  - Data: X — Just data, no labels
  - Goal: Learn some underlying hidden structure of the data
  - Examples: clustering, dimension reduction, feature learning, density estimation, etc.

Unsupervised Learning

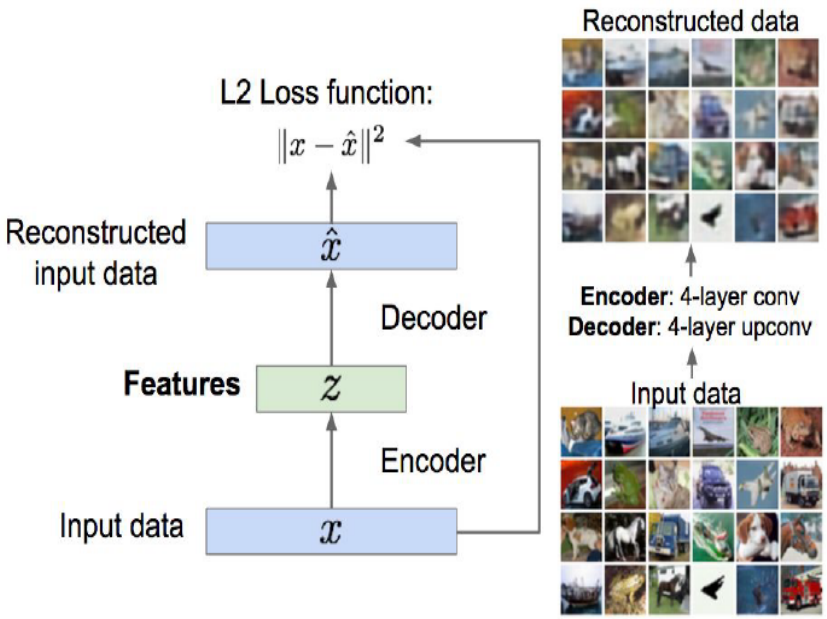


K-means clustering



Principal Component Analysis
(Dimensionality reduction)

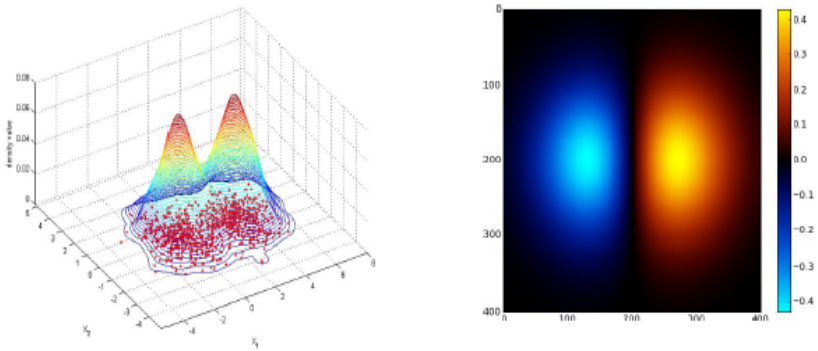
Unsupervised Learning



Autoencoders
(Feature learning)

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Training data is cheap

Solve unsupervised learning

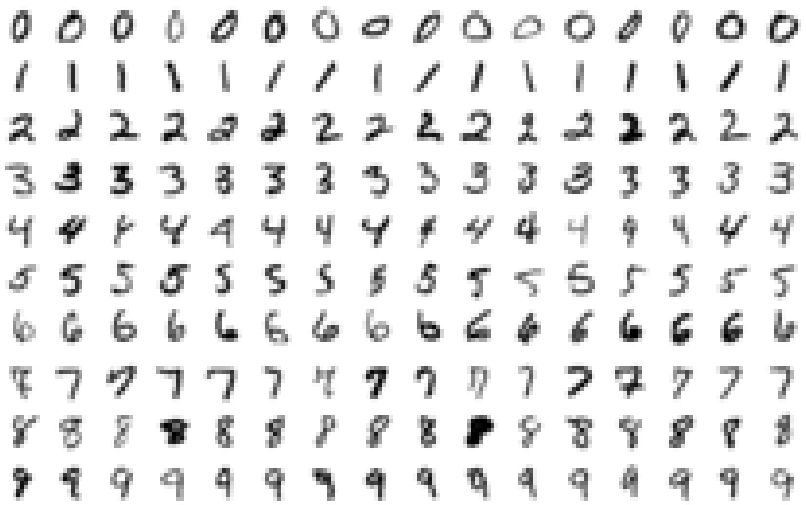
$\Rightarrow$ understand structure of visual world

Not easy!

Polupar ML Packages

Benchmark Datasets

- MNIST, CIFAR10, CelebA, ImageNet...



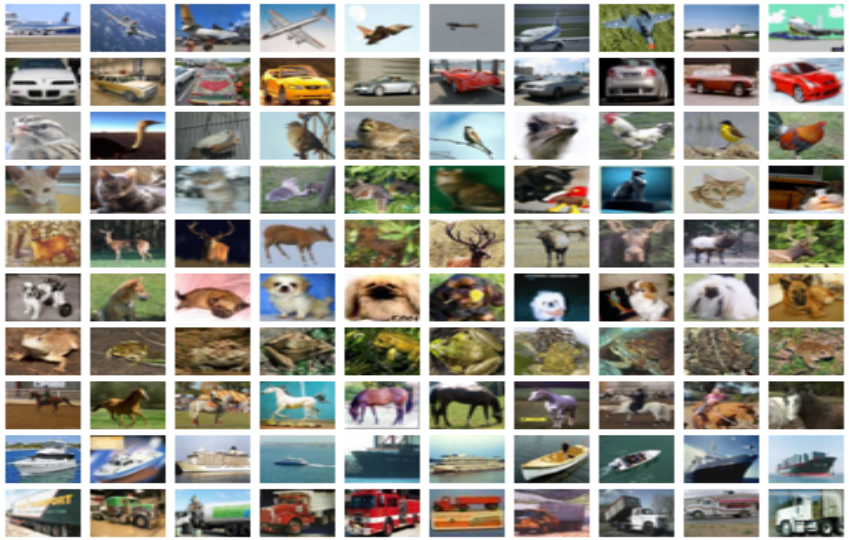
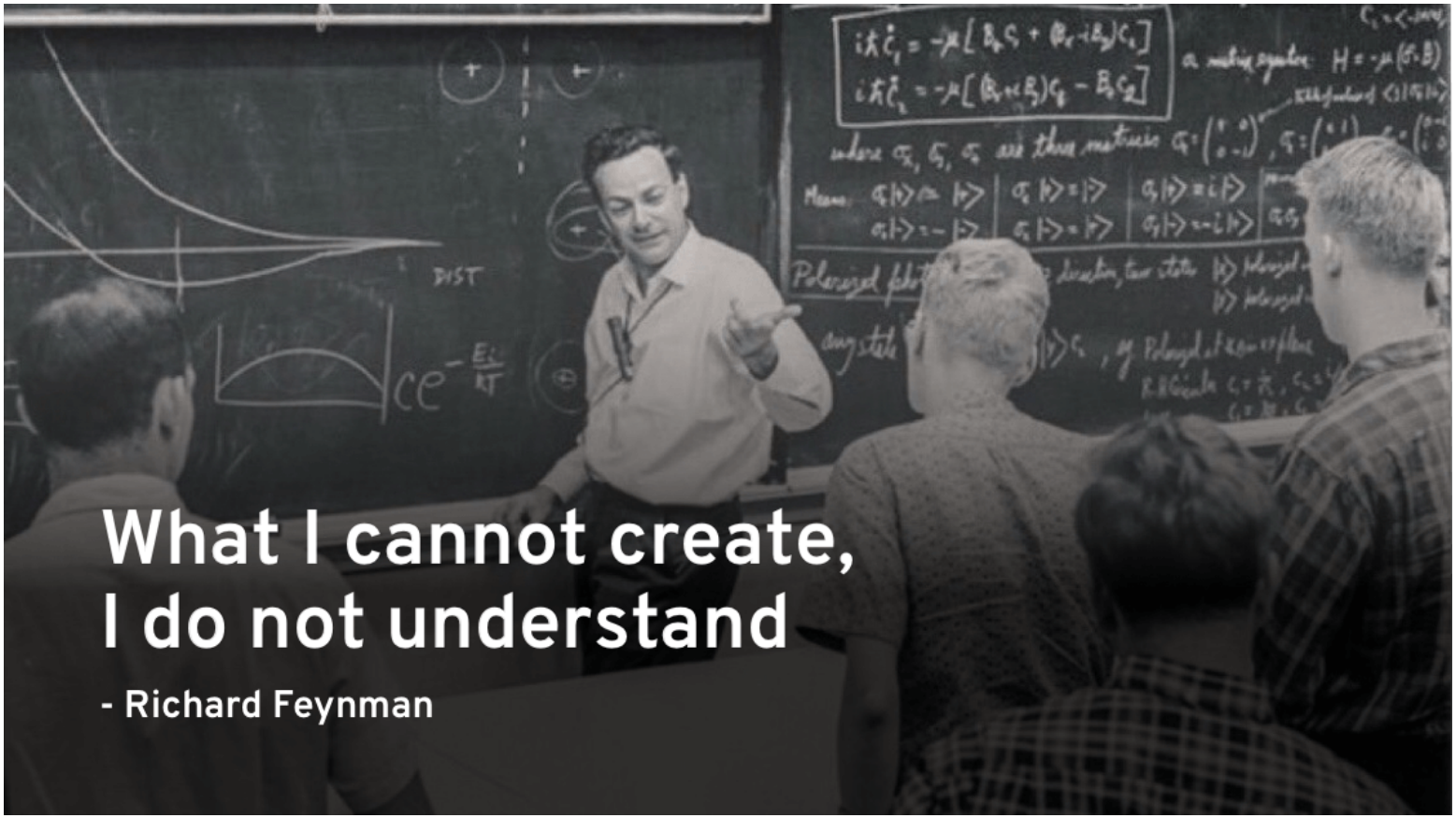


Table of contents


What I cannot create,
I do not understand
- Richard Feynman

Generative Models

- Given training data, generate new samples from the same distribution
- Training data ~ $P_{data}(x)$, generated samples ~ $P_{model}(x)$; Want to learning $P_{model}(x)$ similar to $P_{data}(x)$.

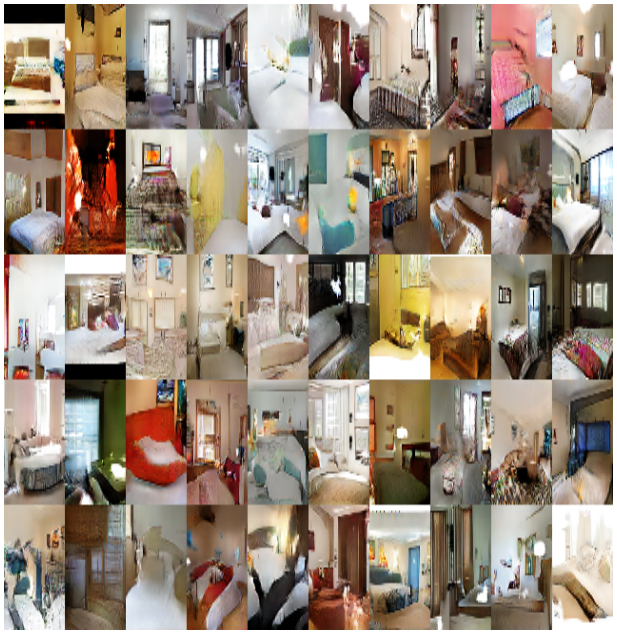
Figure: 1024×1024 images generated using the CELEBA-HQ dataset.

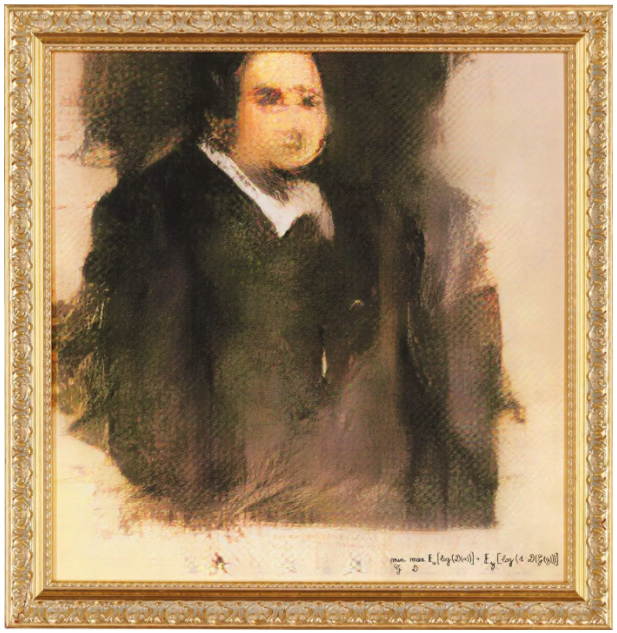
Generative Models

- Density estimation in statistics: a core problem in unsupervised learning
  - Explicit density estimation: explicitly define and solve for $P_{model}(x)$
  - Implicit density estimation: learn model that can sample from $P_{model}(x)$ without explicitly defining it

Why Generative Models?

- Generative models can be used by experts as an assistive tool, e.g. ChatGPT
- Realistic samples for artwork, super-resolution, colorization, cybersecurity, etc.
- Training generative models can also enable inference of latent representations that can be useful as anomaly detection and representation learning

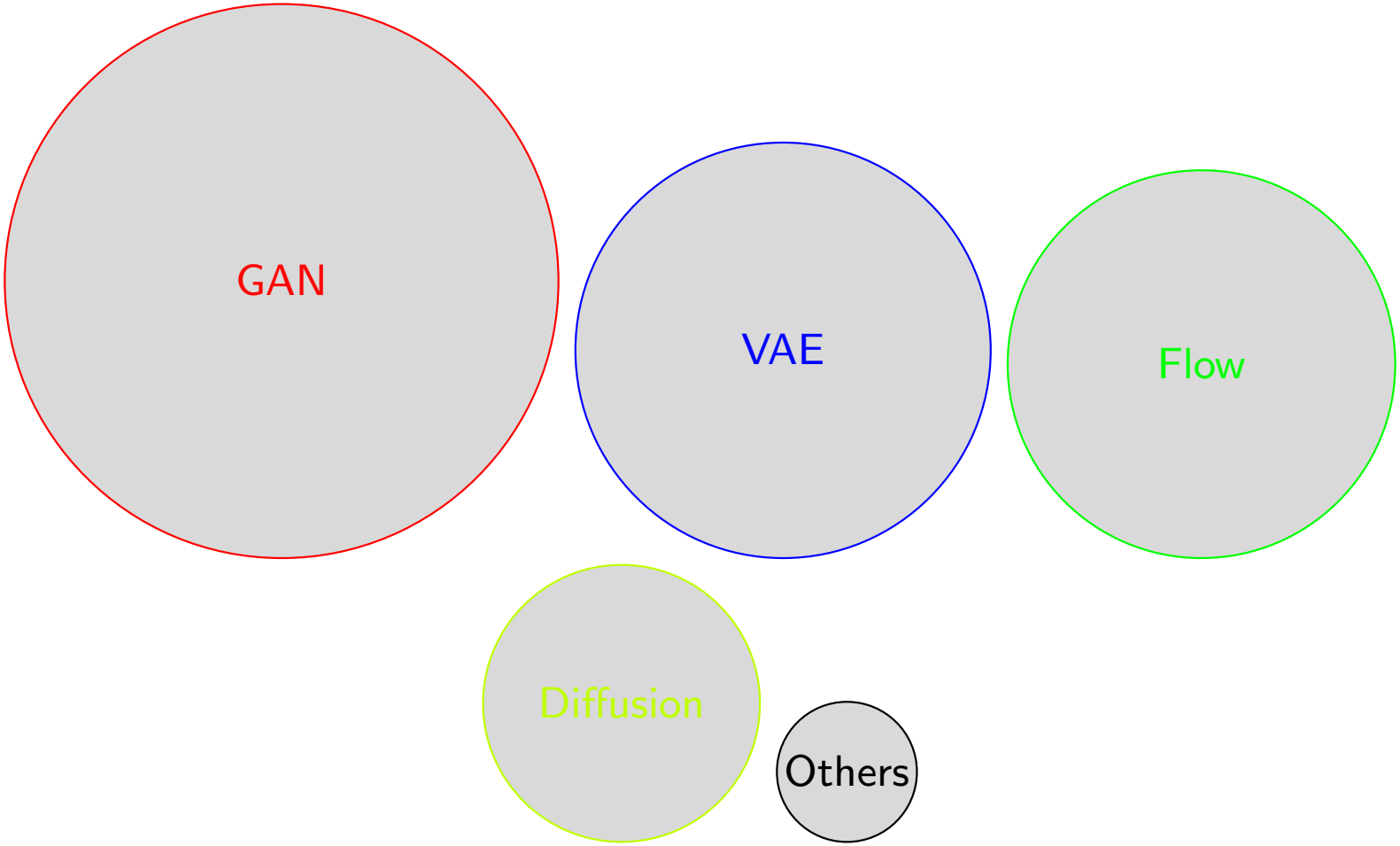




Taxonomy of Generative Models

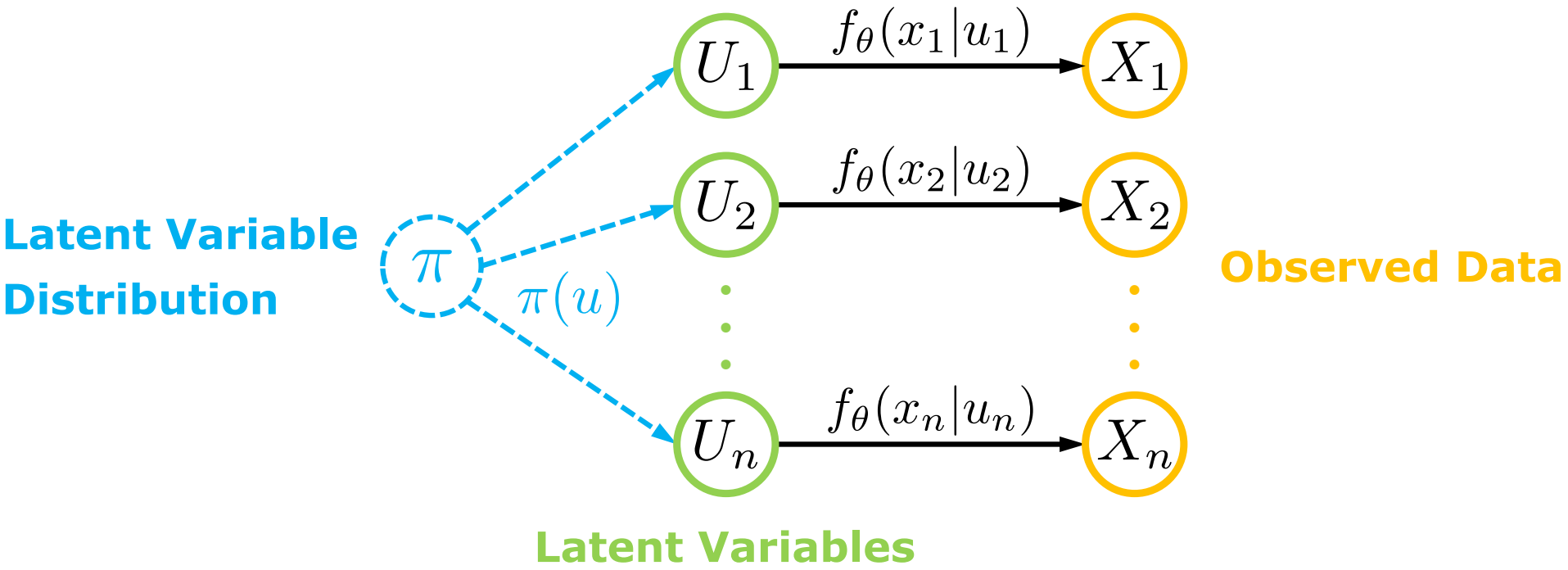
- Generative models
  - Explicit density
    - ★ Tractable density
      - — Fully visible belief nets
      - — PixelRNN/CNN
      - — **Flow-based models**
      - — Energy-based models
    - ★ Approximate density
      - — **Variational autoencoder**
      - — **Diffusion model**
      - — Boltzmann machine
  - Implicit density
    - ★ Markov chain Monte carlo
    - ★ **GAN**

Generative Models in Deep Learning



Most generative models belong to latent variable models!

Connection to Latent Variable Models



Many statistical and machine learning models can be treated as special cases of latent variable models!

— linear mixed models, Gaussian mixture models

— most generative models

Existing Inference Frameworks

- **Bayesian inference** (Bayes, 1764)
  - **Pro**: Widely used in real applications
  - **Pro**: Elegant and well-developed statistical properties
  - **Con**: Requires fully known $\pi(u)$ and $f(x|u)$
  - **Con**: High computational cost with MCMC; nontrivial to scale to large data sets
- **The EM algorithm** (Dempster, et al., 1977)
  - **Pro**: Allows for unknown parameters in $\pi(u)$ and $f(x|u)$, thus bringing more flexibility in modeling
  - **Con**: Mostly used for point estimation
  - **Con**: E-step does not have closed form for complicated models
  - **Con**: M-step is also challenging for big data
- **Variational inference** (Jordan et al., 1999)
  - **Pro**: Very efficient in computation
  - **Pro**: Easy to scale to large data sets
  - **Con**: Lack of accuracy in the inference result

Table of contents

General Goals

- Fit a density model $p_\theta(x)$ for a continuous random variable $X \in \mathbb{R}^D$
  - Good fit to the training data (really, the underlying distribution!)
  - For new x, ability to evaluate $p_\theta(x)$
  - Ability to sample from $p_\theta(x)$
  - And, ideally, a latent representation that's meaningful

How to Fit a Density Model?

- Maximum likelihood

$$\max_{\theta} \sum_{i} \log p_\theta(x^{(i)})$$

- Equivalently:

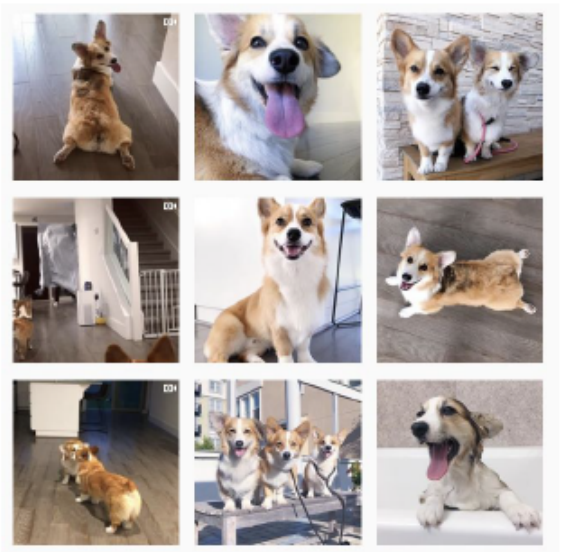
$$\max_{\theta} \mathbb{E}_X \Big[\log p_\theta(X) \Big]$$

- KL between the data distribution and the model distribution
- Example: Mixture of Gaussian $p_\theta(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x; \mu_i, \sigma_i^2)$

Aside on Mixtures of Gaussians

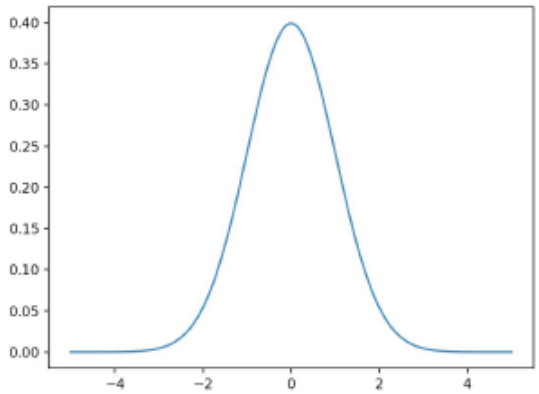
- Do mixtures of Gaussians work for high-dimensional data?
- Not really. The sampling process is:
  1. Pick a cluster center
  2. Add Gaussian noise
- Imagine this for modeling natural images! The only way a realistic image can be generated is if it is a cluster center, i.e. if it is already stored directly in the parameters.

High Dimensional Data



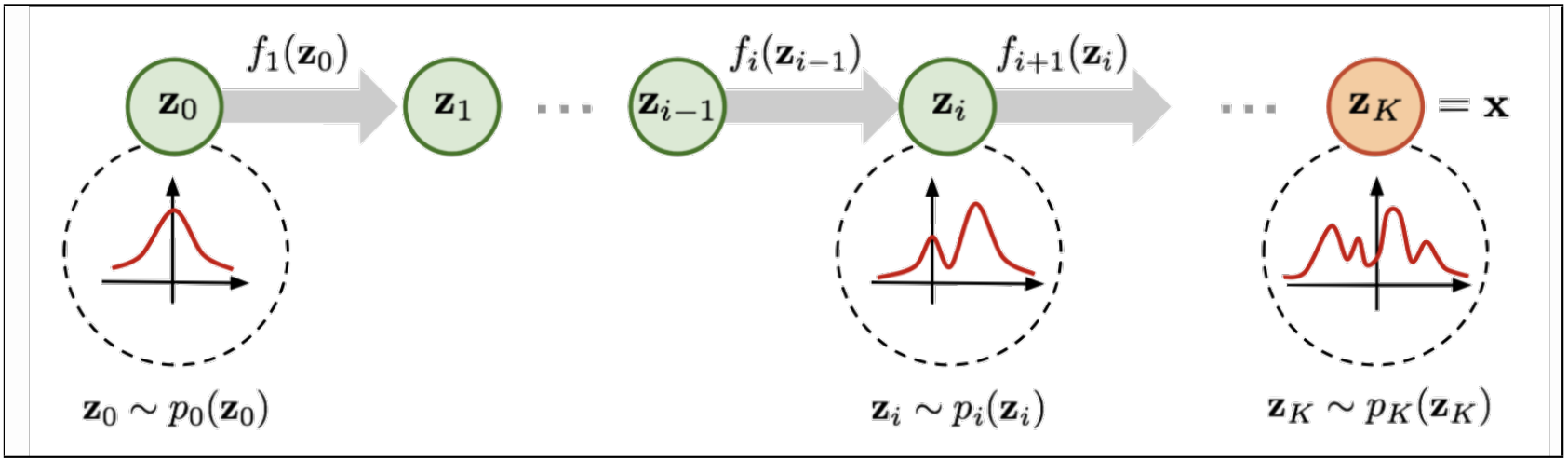
f (inference)

f^{-1} (sampling)



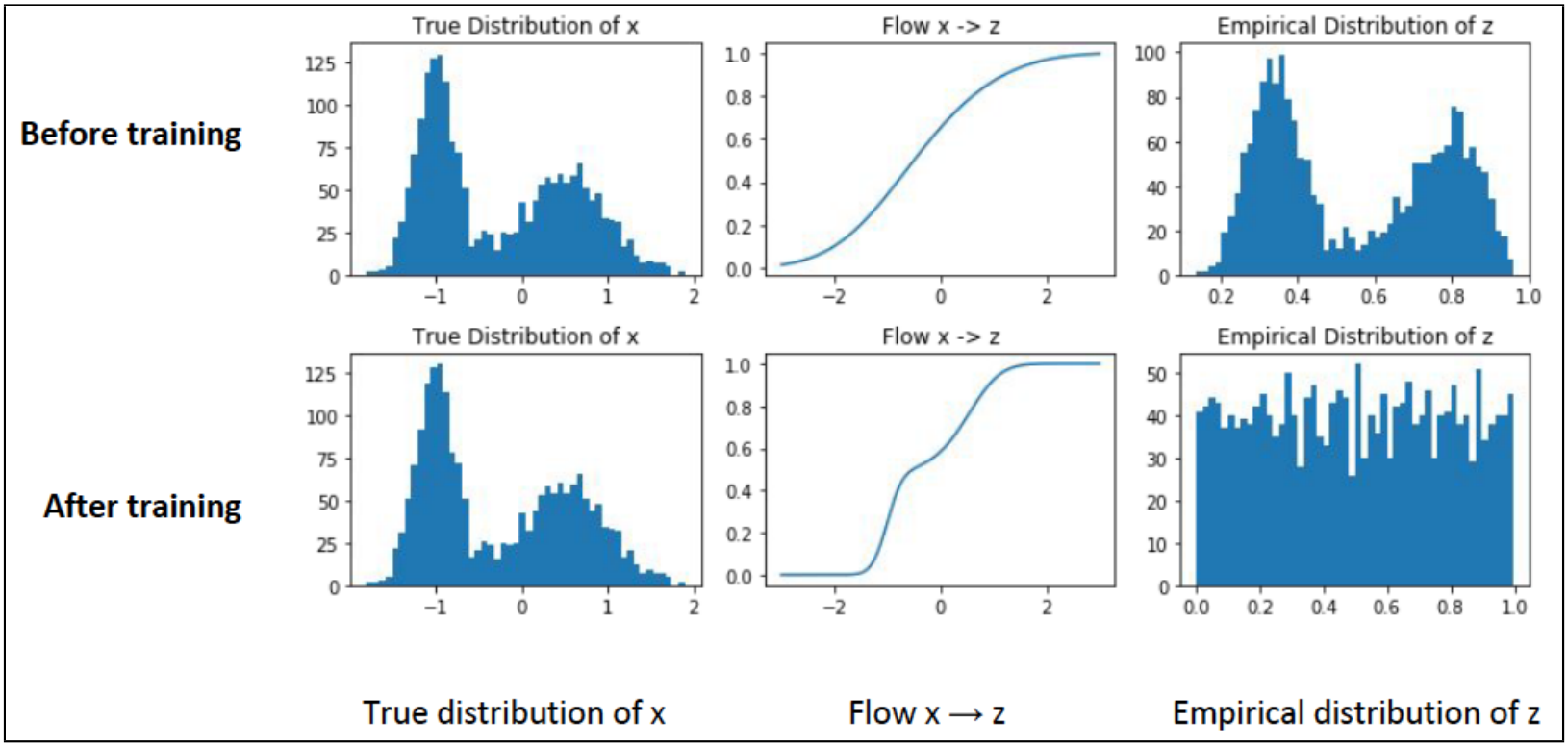
x and z must have the same dimension

Normalizing Flows

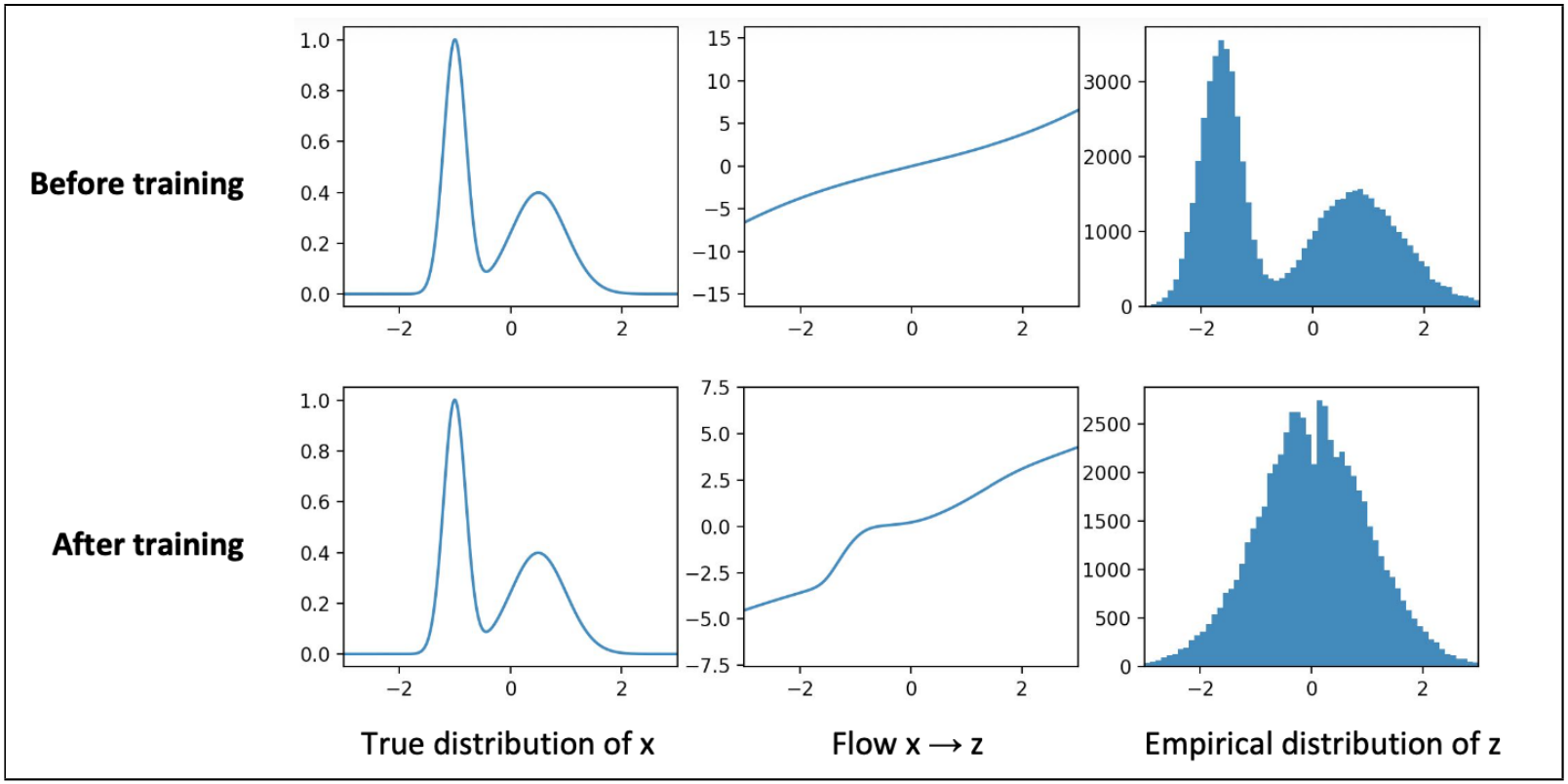


- **Normalizing Flow (NF)**: a flow of invertible transformations that takes $z_0 \sim \mathcal{N}(0, I)$ and outputs x following a complex distribution

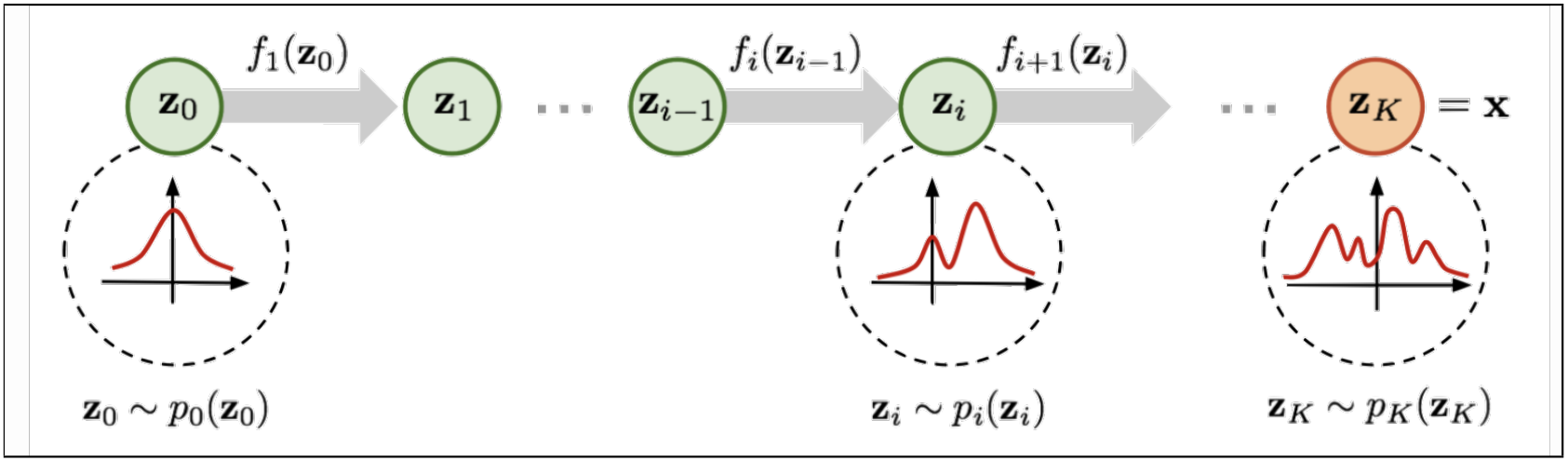
A Simple Illustration: $X \rightarrow$ Uniform



A Simple Illustration: $X \to N(0,1)$



Normalizing Flows



- **A flow-based generative model** takes samples of x and learns $f_1^{-1}, f_2^{-1}, \cdots f_k^{-1}$ such that

$$z_0 = f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_K^{-1}(x)$$

- Once the functions are learned, it uses

$$x = f_k \circ f_{k-1} \circ \cdots \circ f_1(z_0)$$

to approximate the distribution of x

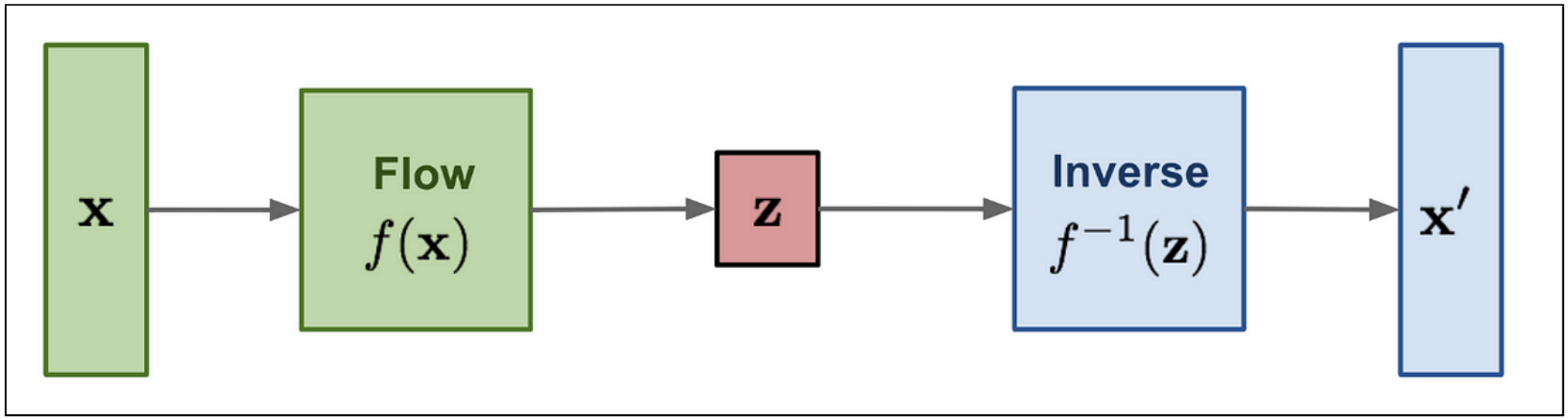
NF - Deriving $p(x)$ explicitly

- **Change of Variables Theorem**:
  Let $z \sim \pi(z)$ and $x = f(z)$, where f is invertible. Then

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

  where we call the matrix $\frac{df^{-1}(x)}{dx}$ the Jacobian of f^{-1}

NF - Deriving $p(x)$ explicitly



- Applying the change of variables theorem recursively and taking the log, we obtain the log-likelihood:

$$\log p(x) = \log \pi_0(z_0) - \sum_{i=1}^{K} \log \left| \det \frac{df_i}{dz_{i-1}} \right|$$

NF - Remarks

- With NF architecture, the objective function arises naturally. For samples $x^{(1)}, x^{(2)}, \cdots, x^{(N)}$,

$$\mathcal{L}(x^{(1)}, x^{(2)}, \cdots, x^{(N)}) = -\frac{1}{N} \sum_{i=1}^{N} \log p(x^{(i)})$$

- The key is in finding a suitable class of the transformations f_i.
- Challenges: Ideally, the transformations are
  — capable of growing arbitrarily complex
  —- yet their inverse and Jacobian are easy to compute

RealNVP

- Short for "Real-valued non-volume preserving" model
- Each transformation $f_i : \mathbb{R}^D \rightarrow \mathbb{R}^D$ for some $i \in \{1, 2, \cdots K\}$ used in this model is termed an **affine coupling layer**

RealNVP - affine coupling layers

Say $f(x) = y$ is an affine coupling layer. Then for some $d < D$ and some functions $s : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and $t : \mathbb{R}^d \to \mathbb{R}^{D-d}$,

$$\begin{aligned} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}). \end{aligned}$$

The inverse is

$$\begin{aligned} x_{1:d} &= y_{1:d} \\ x_{d+1:D} &= (y_{d+1:D} - t(y_{1:d})) \odot \exp(-s(y_{1:d})). \end{aligned}$$

RealNVP - affine coupling layers

- The Jacobian of f is

$$J = \begin{bmatrix} \mathbb{I}_n & 0_{d\times(D-d)} \\ \dfrac{\partial y_{d+1:D}}{\partial x_{1:d}} & diag\Big[\exp\big(s(x_{1:d})\big)\Big] \end{bmatrix}$$

- Thus

$$det(J) = \prod_{i=1}^{D-d} \exp\Big(s(x_{1:d})_i\Big) = \exp\left(\sum_{i=1}^{D-d} s(x_{1:d})_i\right)$$

- Remark: s and t does not contribute to the complexity of the inverse or the Jacobian matrix
  - s and t can be arbitrarily complex – deep neural nets

RealNVP – importance of good masking

- Each affine flow involves partitioning the elements of the input vector into two groups
  - Termed *masking* by the original paper

- Let b be a *binary mask*, a vector with d ones and $D - d$ zeros. Then we can rewrite y as

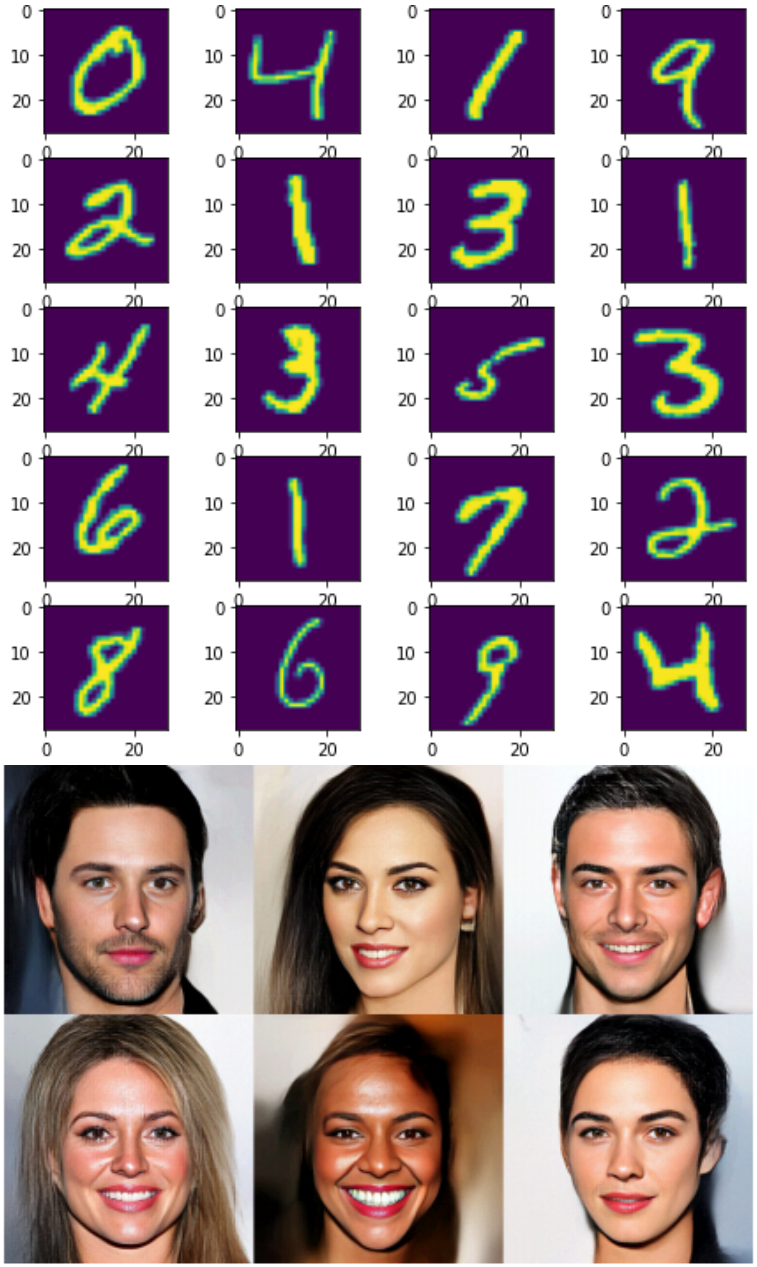
$$y = b \odot x + (1 - b) \odot (x \odot \exp(s(b \odot x)) + t(b \odot x))$$

RealNVP – importance of good masking

Checkerboard x4; channel squeeze; channel x3; channel unsqueeze; checkerboard x3

(Mask top half; mask bottom half; mask left half; mask right half) x2

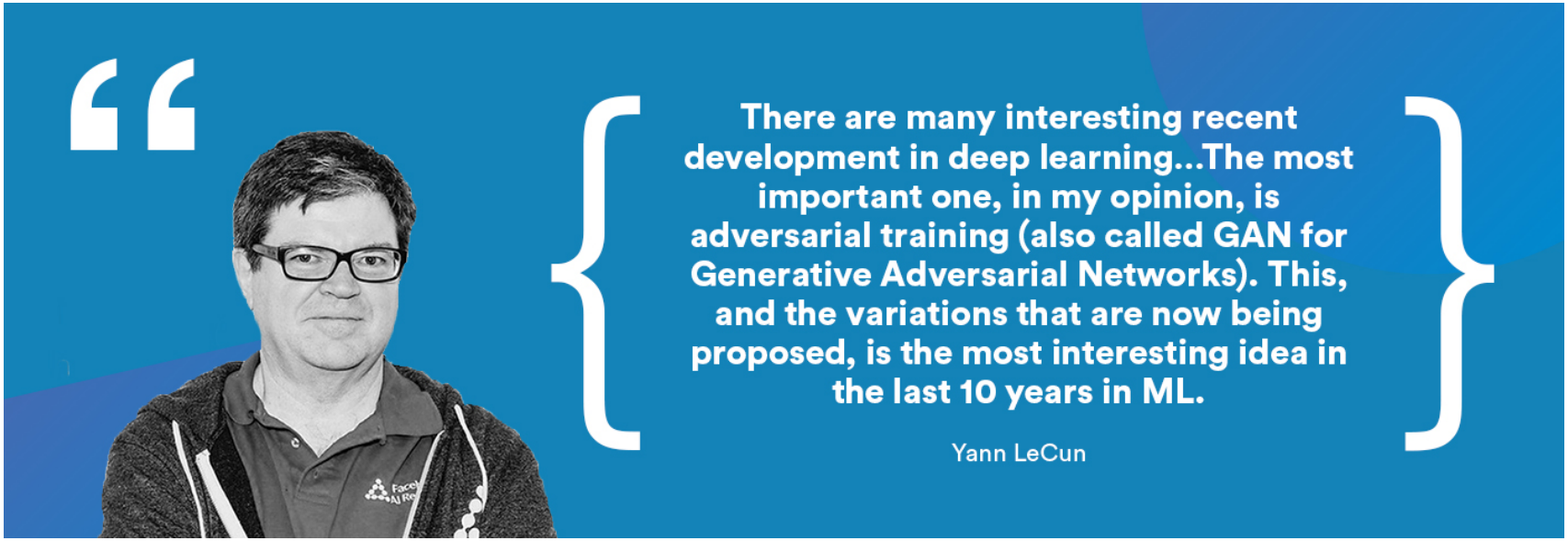
Generative Images



Flow-Based Models Summary

- The ultimate goal: a likelihood-based model with
  - fast sampling
  - fast inference
  - fast training
  - good samples
  - good compression
- Flows seem to let us achieve some of these criteria.
- But how exactly do we design and compose flows for great performance? That is an open question.

Table of contents

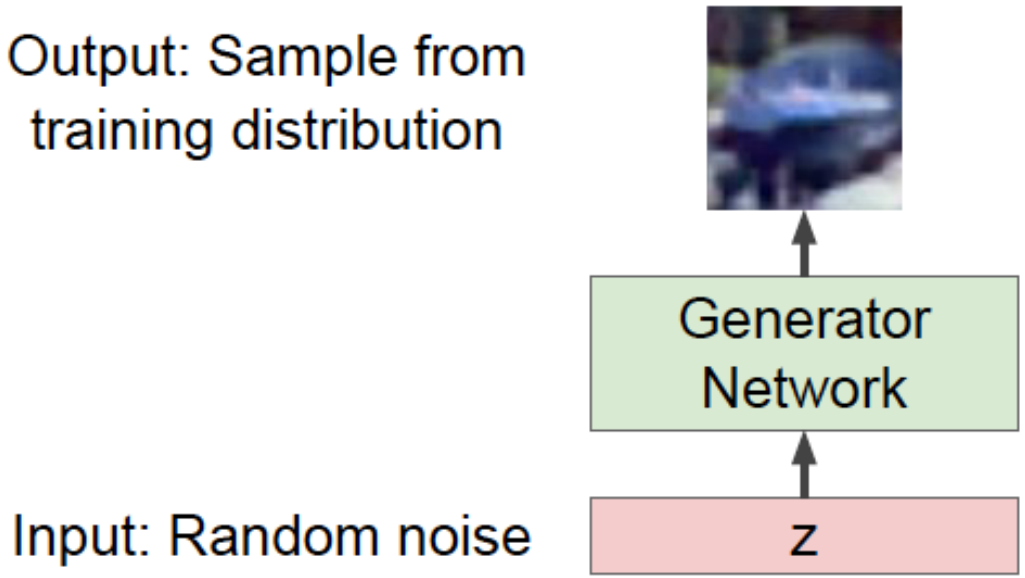

There are many interesting recent development in deep learning...The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks). This, and the variations that are now being proposed, is the most interesting idea in the last 10 years in ML.
Yann LeCun

So far ...

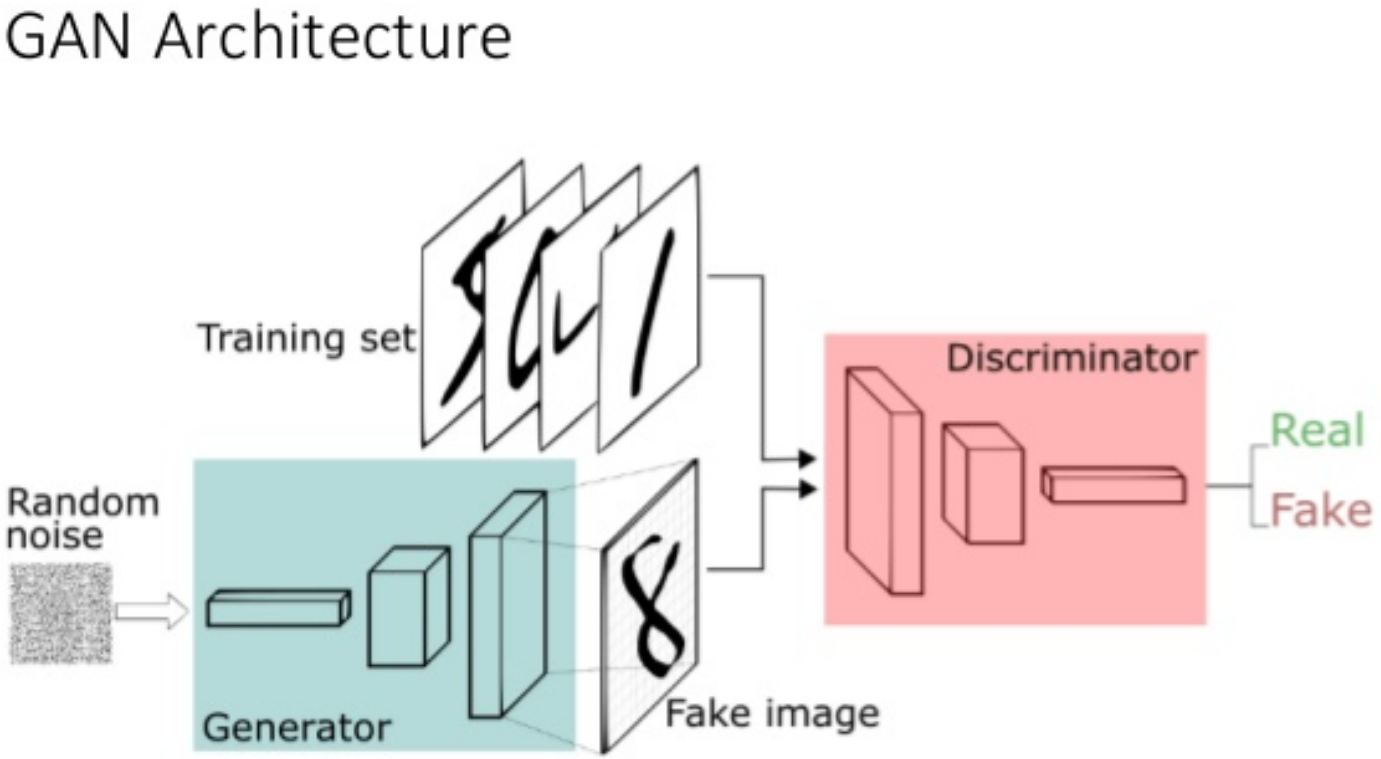
- Flow-based models: $X = f(Z)$ where f is invertible and $Z \sim N(0, I)$. X and Z have the same dimension.
- What if we give up on explicitly modeling density and just want ability to sample?
  — GANs: don't work with any explicit density function!
  — Instead, take the game-theoretic approach: learn to generate from the training distribution through a 2-player game

Generative Adversarial Networks (Ian Goodfellow at al. 2014)

- Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!
- Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution. The dimension of Z is much smaller than that of X.
- **Question**: What can we use to represent this complex transformation?
  Answer: A neural network



Training GANs: Two-player game



- Generator network: try to fool the discriminator by generating real-looking images
- Discriminator network: try to distinguish between real and fake images

Training GANs: Two-player game

Train jointly in minimax game, and the minimax objective function is

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}(G_{\theta_g}(z))\right) \right]$$

- $D_{\theta_d}(\cdot)$: Discriminator outputs likelihood in $(0, 1)$ of real image
  — $D_{\theta_d}(x)$: Discriminator output for real data x
  — $D_{\theta_d}(G_{\theta_g}(z))$: Discriminator output for generated fake data $G(z)$
- Discriminator (θ_d) wants to maximize objective such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)
- Generator (θ_g) wants to minimize objective such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)

Training GANs: Two-player game

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}(G_{\theta_g}(z))\right) \right]$$

1. Gradient ascent on discriminator

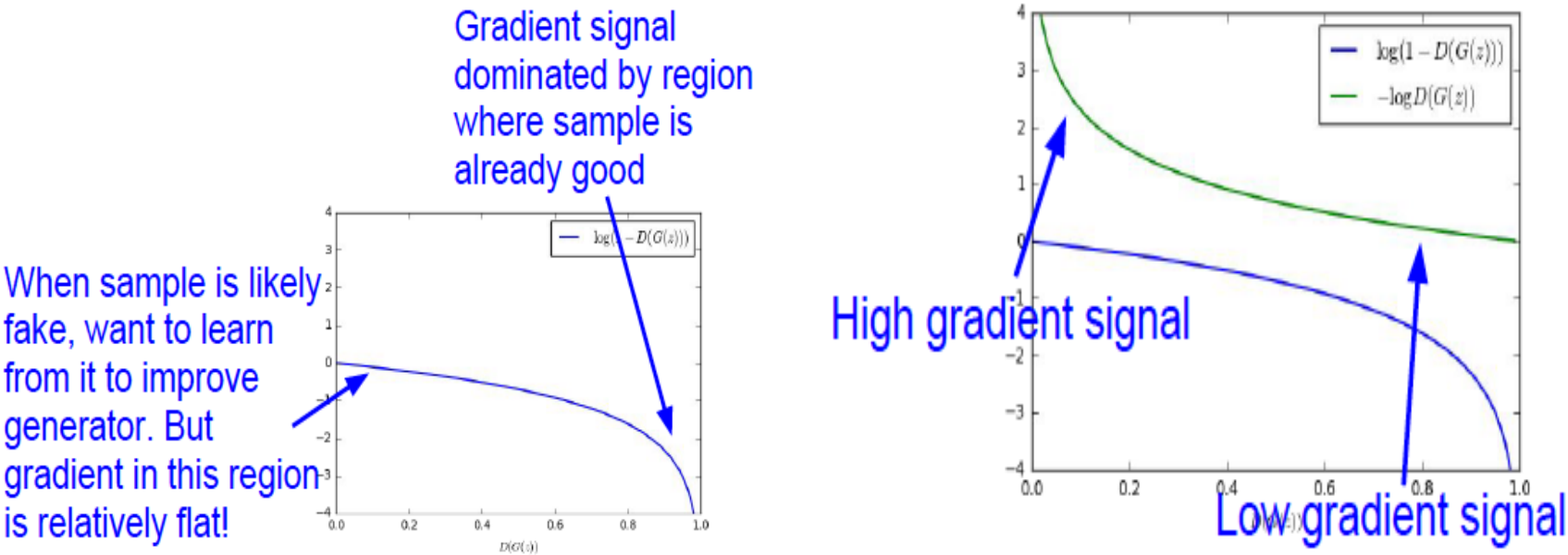
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}(G_{\theta_g}(z))\right) \right]$$

2. Gradient descent on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}(G_{\theta_g}(z))\right)$$

Training GANs: Two-player game

- In practice, optimizing this generator objective does not work well!



- Instead: Gradient ascent on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

- Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong ⇒ works much better! Standard in practice.

Training GANs: Two-player game

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k 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:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  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 descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Generative Model Framework: GAN

- Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, and is an active area of research.
- $X \sim P_X$ vs. $G(Z) \sim P_G$ with $Z \sim N(0, I)$.
- Training GAN is equivalent to minimizing Jensen-Shannon divergence between generator and data distributions.
- $D(P_X, P_G) = \sup_{f \in \mathcal{F}} \left\{ \mathbb{E}_{X \sim P_X} \phi_1(f(X)) - \mathbb{E}_{Y \sim P_G} \phi_2(f(Y)) \right\}$

Training GANs: Two-player game

Choosing other divergences with better loss landscapes helps training

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1))^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x - \mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$

The GAN Zoo

"The GAN Zoo"

See also: https://github.com/soumith/ganhacks for tips and tricks for trainings GANs

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

Comparing GANs

- GANs provide objectively good images (especially under direct comparison to other generative models like VAEs)
- How do we compare GAN performance between models?
  — No way to compute probability $p_g(x)$, can't compute log-likelihood
- Original paper: "Are GANs Created Equal? A Large-Scale Study" by Lucic et al. at Google Brain
- Contributions:
  — Comparison of state-of-the-art GANs
  — Empirical evidence showing GAN comparison needs summary of results not just best
  — Assess Fréchet Inception Distance (FID) for GAN comparison

Comparing GANs

- Inception Score (IS): the higher, the better
  — Use pretrained Inception v3 network trained on ImageNet-1k , calculate the score on $G(z)$
  — Well correlated with human perception but has issues, see "A Note on the Inception Score"

$$IS(G(z)) = \exp\left(\mathbb{E}_{x\sim p_g}[D_{KL}(p(y|x)\|p(y))]\right)$$

- Fréchet Inception Distance (FID): the lower, the better
  — $G(z)$ and x embedded by specific layer in Inception v3
  — Embedding layer viewed as multivariate Gaussian; Mean and covariance for layer with real data (μ_x, Σ_x) and generated data (μ_g, Σ_g)

$$FID(x, g) = \|\mu_x - \mu_g\|_2^2 + Tr(\Sigma_x + \Sigma_g - 2\sqrt{\Sigma_x\Sigma_g})$$

  — Consistent with humans, more robust to noise than IS, and can detect intra-class mode dropping
  — Negative correlation between FID and visual quality

Comparing GANs: Fairness

- How did they compare so many different GANs while keeping it fair?
  — Architecture kept the same for all models
  — Four Test Sets: MNIST, FASHION-MNIST, CIFAR-10, CelebA
  — Hyperparameters searches were random and done for each dataset and also inferred across models
  — Comparisons were made across computational budget range - THIS WAS VERY IMPORTANT!!

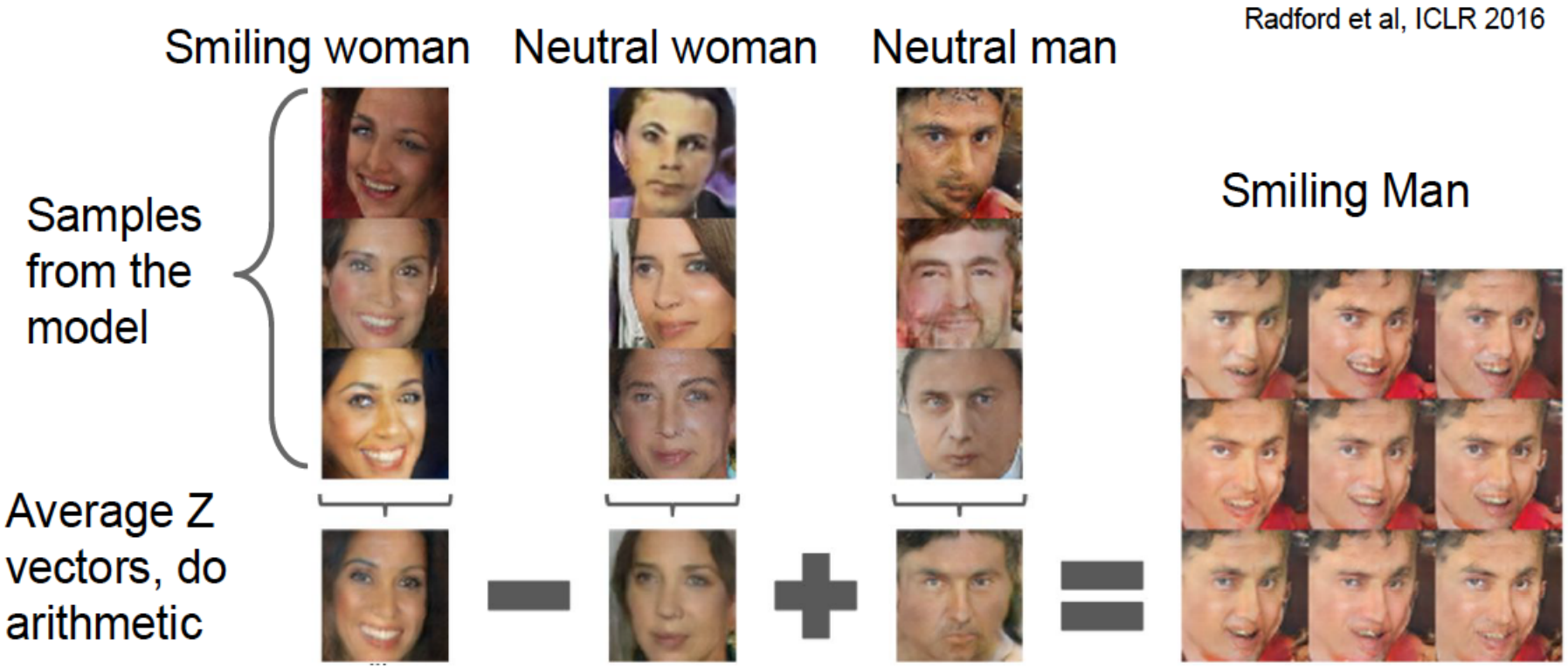
Beautiful Pictures from GANs

Andrew Brock, Jeff Donahue, Karen Simonyan, Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019

Beautiful Pictures from GANs

ALL OF THESE FACES ARE FAKE CELEBRITIES SPAWNED BY GANS!

GANs: Interpretable Vector Math



GANs: Interpretable Vector Math

GANs

- Do not work with an explicit density function. Take game-theoretic approach: learn to generate from training distribution through 2-player game
- Pros:
  — Beautiful, state-of-the-art samples!
- Cons:
  — Trickier / more unstable to train
  — Can't solve inference queries such as $p(x)$, $p(z|x)$
- Other related research:
  — Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
  — Conditional GANs, GANs for all kinds of applications

Table of contents

So far ...

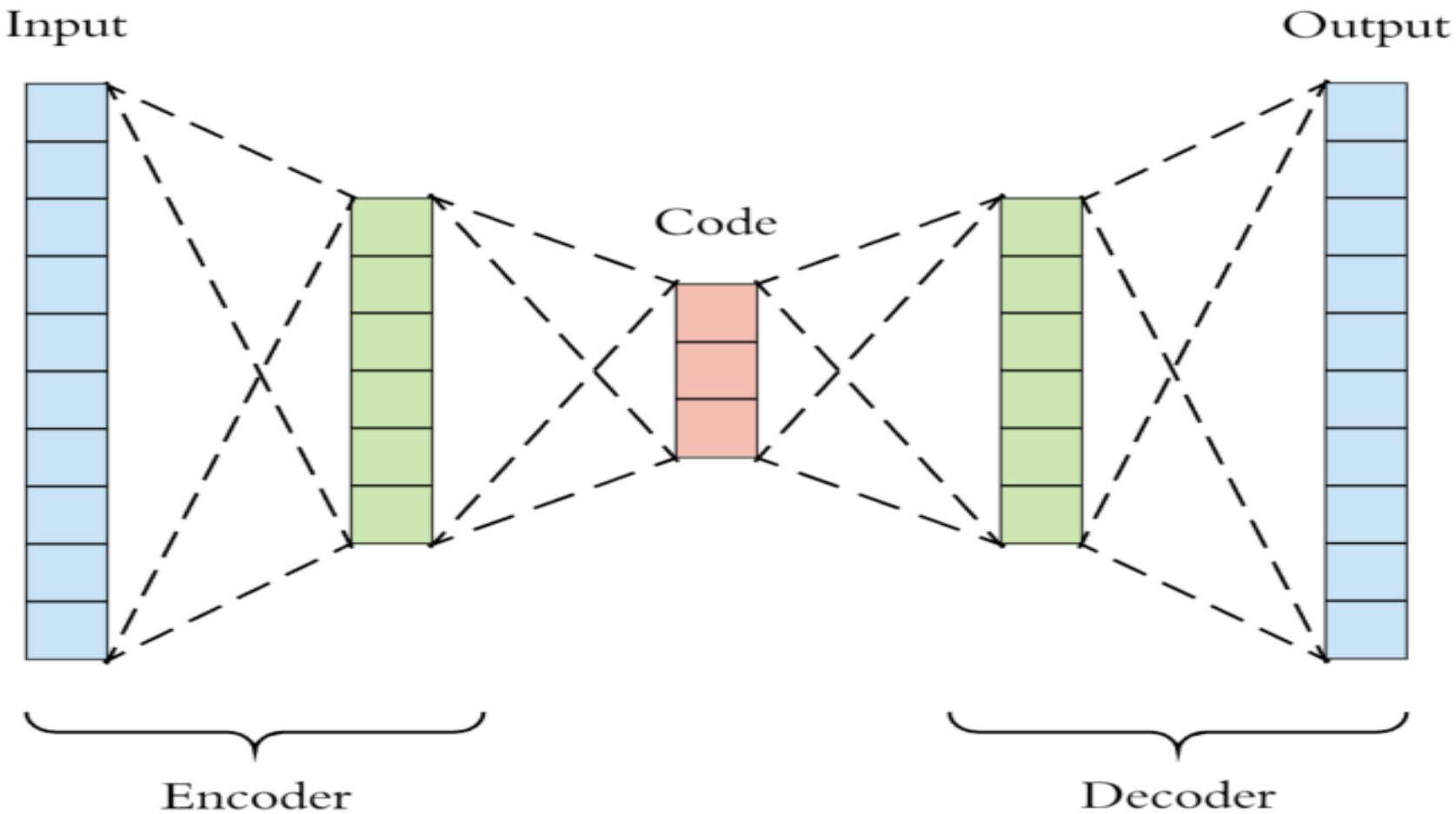
- Flow-based models: $X = f(Z)$ where f is invertible and $Z \sim N(0, I)$. X and Z have the same dimension.
- GANs: $X = G(Z)$. The dimension of Z is much smaller than that of X. Take the game-theoretic approach.
- VAEs define an intractable density function with latent z:

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

$\Rightarrow$ Cannot optimize directly, derive and optimize the lower bound on likelihood instead.

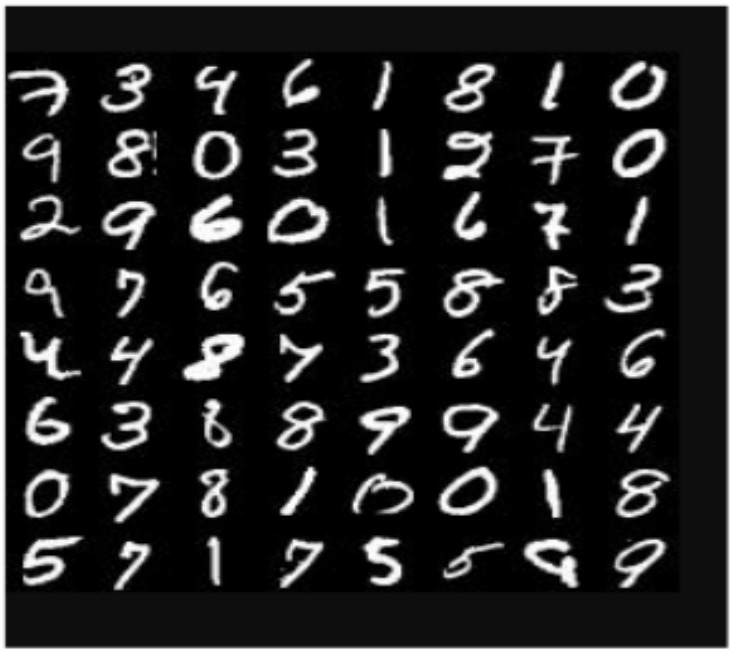
Some background on autoencoder

- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
- **Question**: Why dimensionality reduction?
  Answer: Want features to capture meaningful factors of variation in data

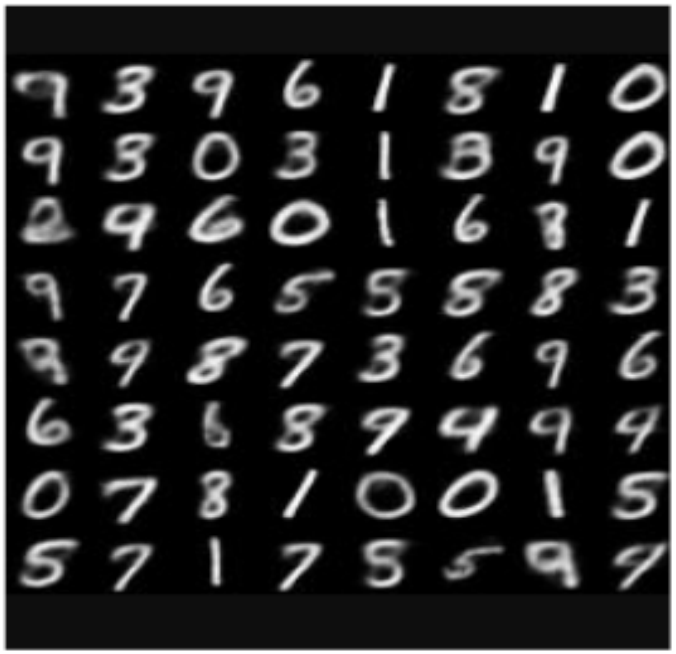


Some background on autoencoder

- How to learn this feature representation?
  $\Longrightarrow$ Train such that features can be used to reconstruct original data
  "Autoencoding" — encoding itself
- L_2-loss function: $\|x - \hat{x}\|^2$
- Train such that features can be used to reconstruct original data
- Doesn't use labels. After training, throw away decoder



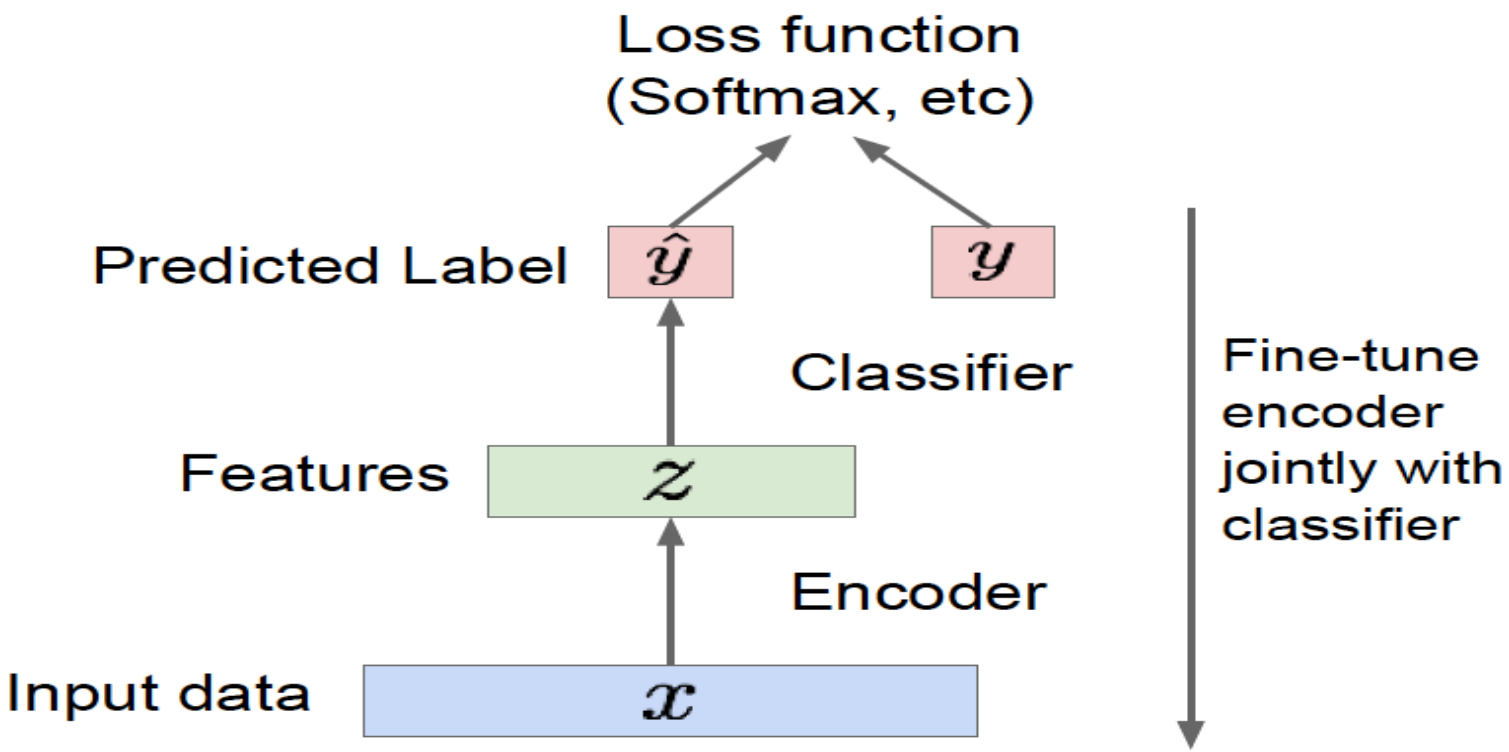
Original Input



Reconstructed Input

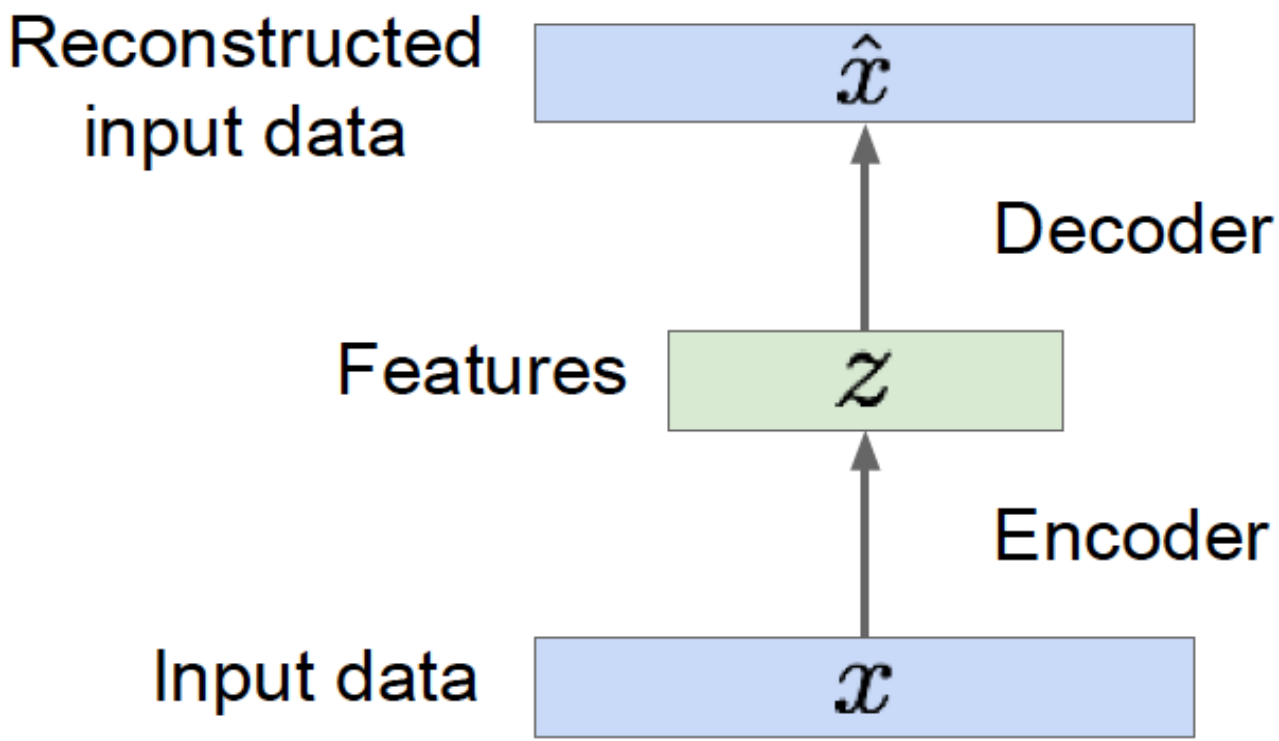
Some background on autoencoder

- Encoder can be used to initialize a supervised model



Some background on autoencoder

- Autoencoders can reconstruct data, and can learn features to initialize a supervised model
- Features capture factors of variation in training data. Can we generate new images from an autoencoder?



Variational autoencoder (Kingma and Welling 2014)

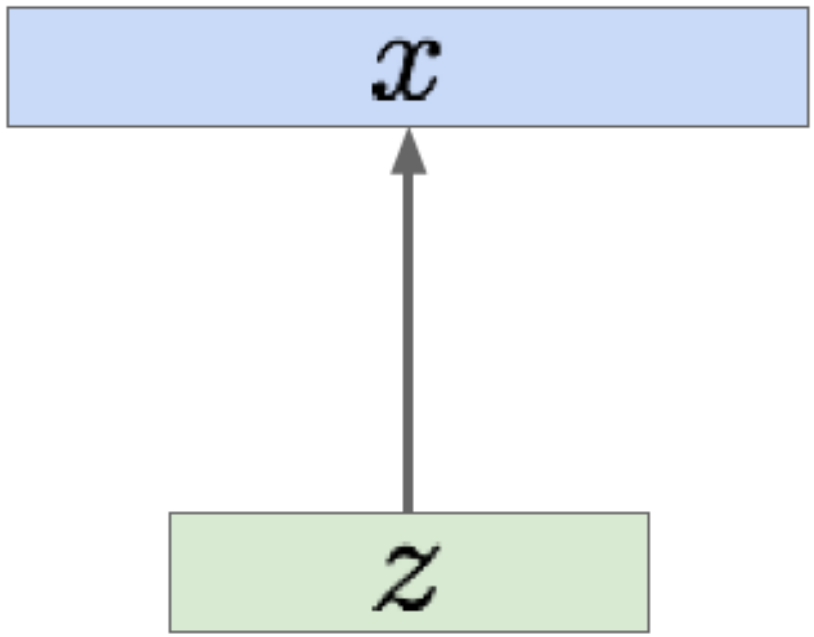
- Probabilistic spin on autoencoders - will let us sample from the model to generate data!
- Assume training data $\{x_i\}_{i=1}^n$ is generated from underlying unobserved (latent) representation z
- Intuition (remember from autoencoders!): x is an image, z is latent factors used to generate x: attributes, orientation, etc.

Sample from true conditional

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

Sample from true prior

$p_{\theta^*}(z)$



Variational autoencoder (Kingma and Welling 2014)

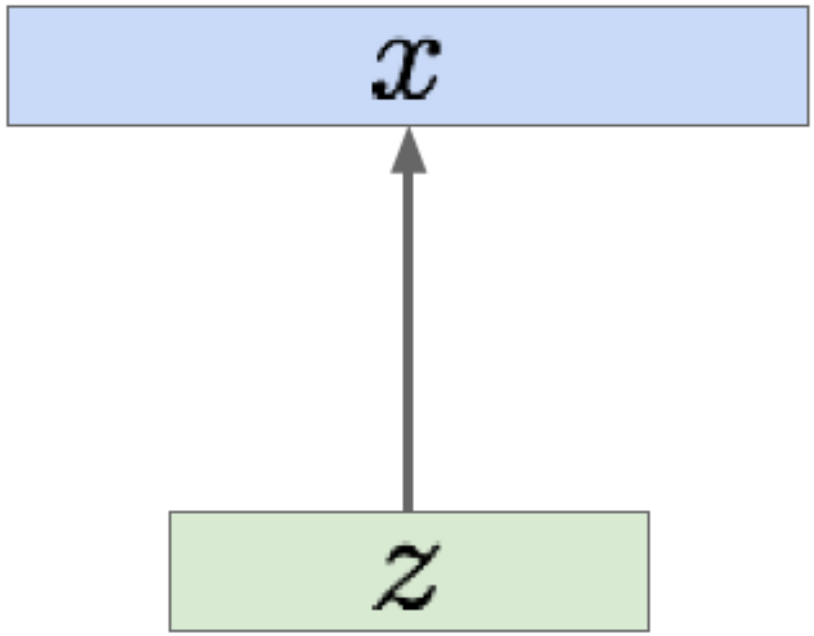
- We want to estimate the true parameters of this generative model.
- How should we represent this model?
  — Choose prior p(z) to be simple, e.g. Gaussian
  — Conditional $p(x|z)$ is complex (generates image) ⇒ represent with neural network

Sample from true conditional

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

Sample from true prior

$$p_{\theta^*}(z)$$



Variational autoencoder (Kingma and Welling 2014)

- How to train the model?
  - Learn model parameters to maximize likelihood of training data

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

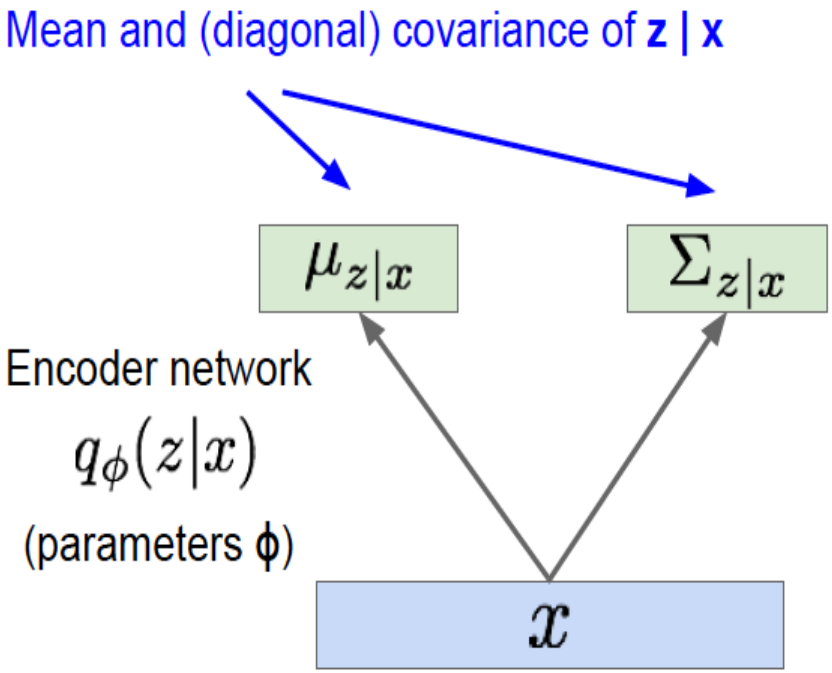
  - **Question**: What is the problem with this?
    Answer: Intractable!

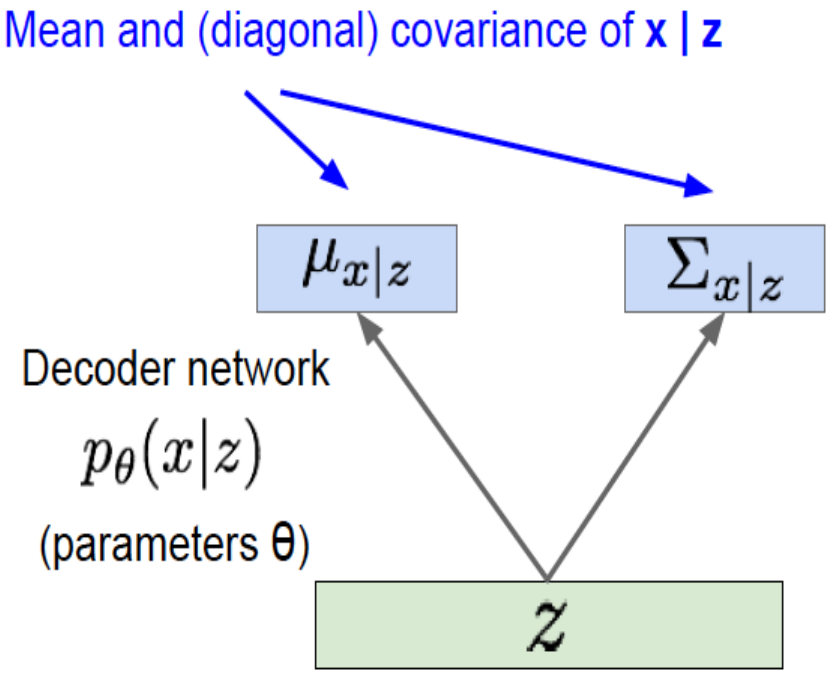
Variational autoencoder: Intractability

- Data likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$ ☹
- Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$ ☹
- Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$.
- Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Variational autoencoder (Kingma and Welling 2014)

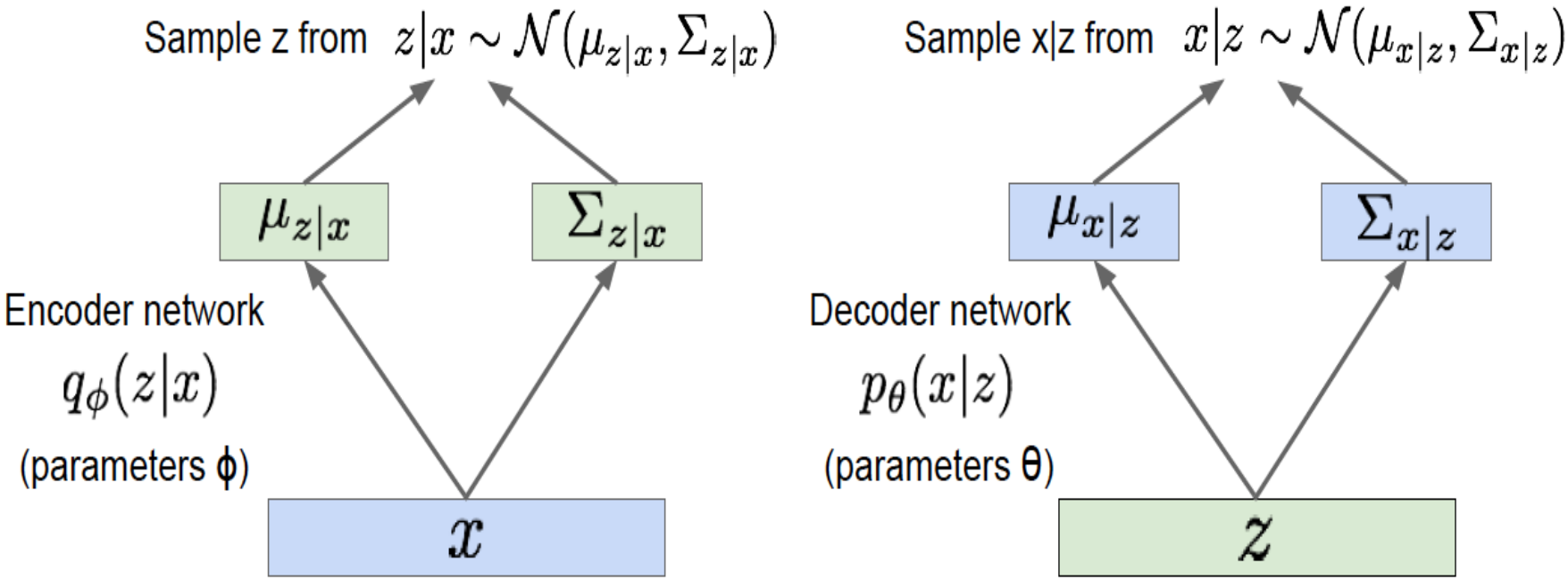
- Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic





Variational autoencoder (Kingma and Welling 2014)

- Allow us to use sampling techniques to solve the optimization problem.
- Allow a generative model



Variational autoencoder (Kingma and Welling 2014)

- Let's work out the (log) data likelihood:

$$
\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x)\Big] \quad (p_\theta(x)\ does\ not\ depend\ on\ z) \\
&= \mathbb{E}_z\Big[\log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)}\Big] \quad (Bayes\ Rule) \\
&= \mathbb{E}_z\Big[\log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)}\frac{q_\phi(z|x)}{q_\phi(z|x)}\Big] \quad (Multiple\ by\ a\ constant) \\
&= \mathbb{E}_z\Big[\log p_\theta(x|z)\Big] - \mathbb{E}_z\Big[\log \frac{q_\phi(z|x)}{p_\theta(z)}\Big] + \mathbb{E}_z\Big[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\Big] \quad (Logarithms) \\
&= \mathbb{E}_z\Big[\log p_\theta(x|z)\Big] - D_{KL}(q_\phi(z|x)\|p_\theta(z)) + D_{KL}(q_\phi(z|x)\|p_\theta(z|x))
\end{aligned}
$$

Variational autoencoder (Kingma and Welling 2014)

- The first term: Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling.
- The second term: This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!
- The third term: $p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term. But we know KL divergence always ≥ 0.

Variational autoencoder (Kingma and Welling 2014)

- Tractable lower bound:

$$\mathcal{L}(x, \theta, \phi) = \mathbb{E}_{z \sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] - D_{KL}(q_\phi(z|x) \| p_\theta(z)),$$

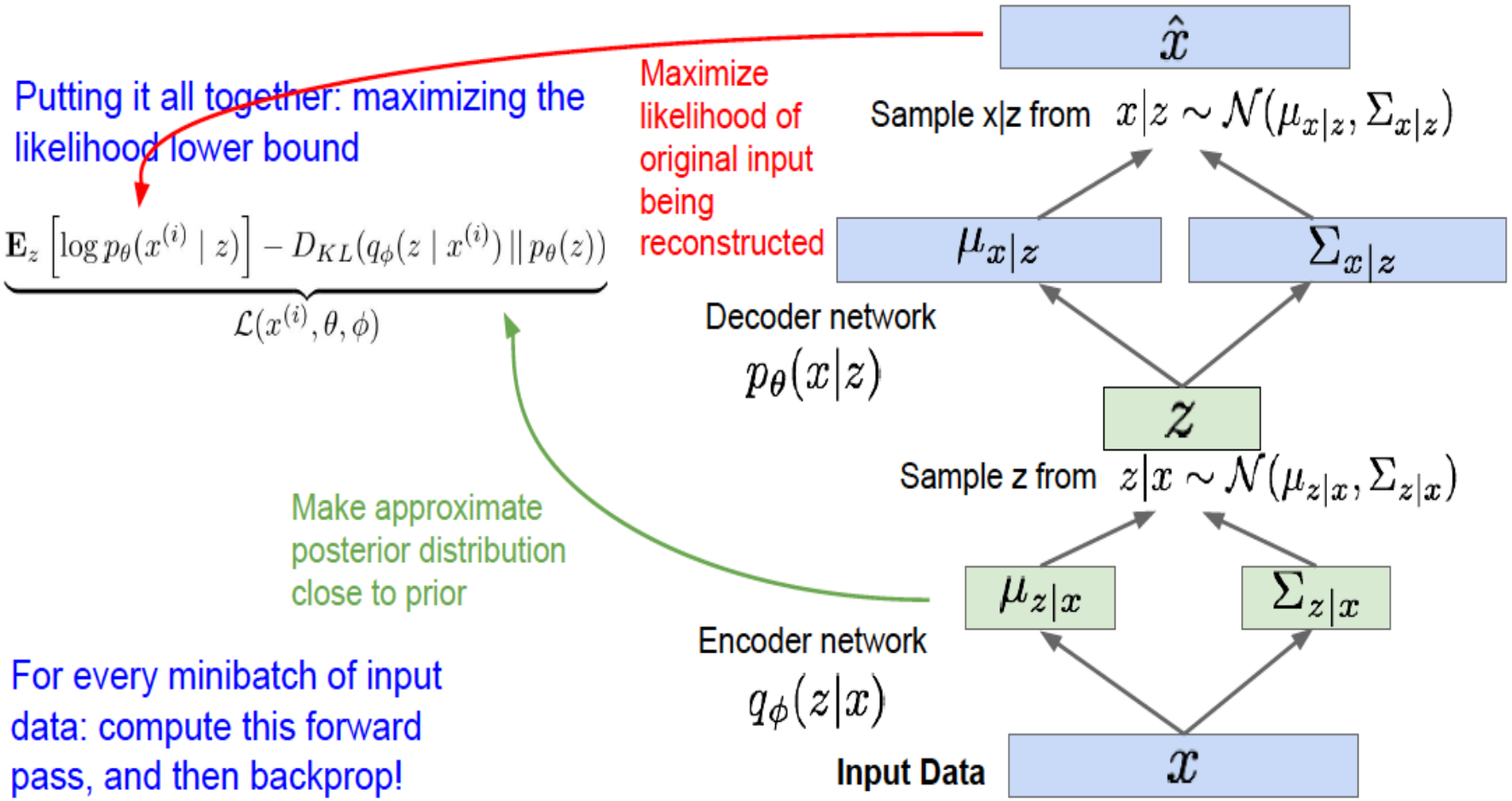
  which we can take gradient of and optimize!
- Variational evidence lower bound (ELBO):

$$\mathcal{L}(x, \theta, \phi) \leq \log p_\theta(x)$$

- Training: Maximize lower bound

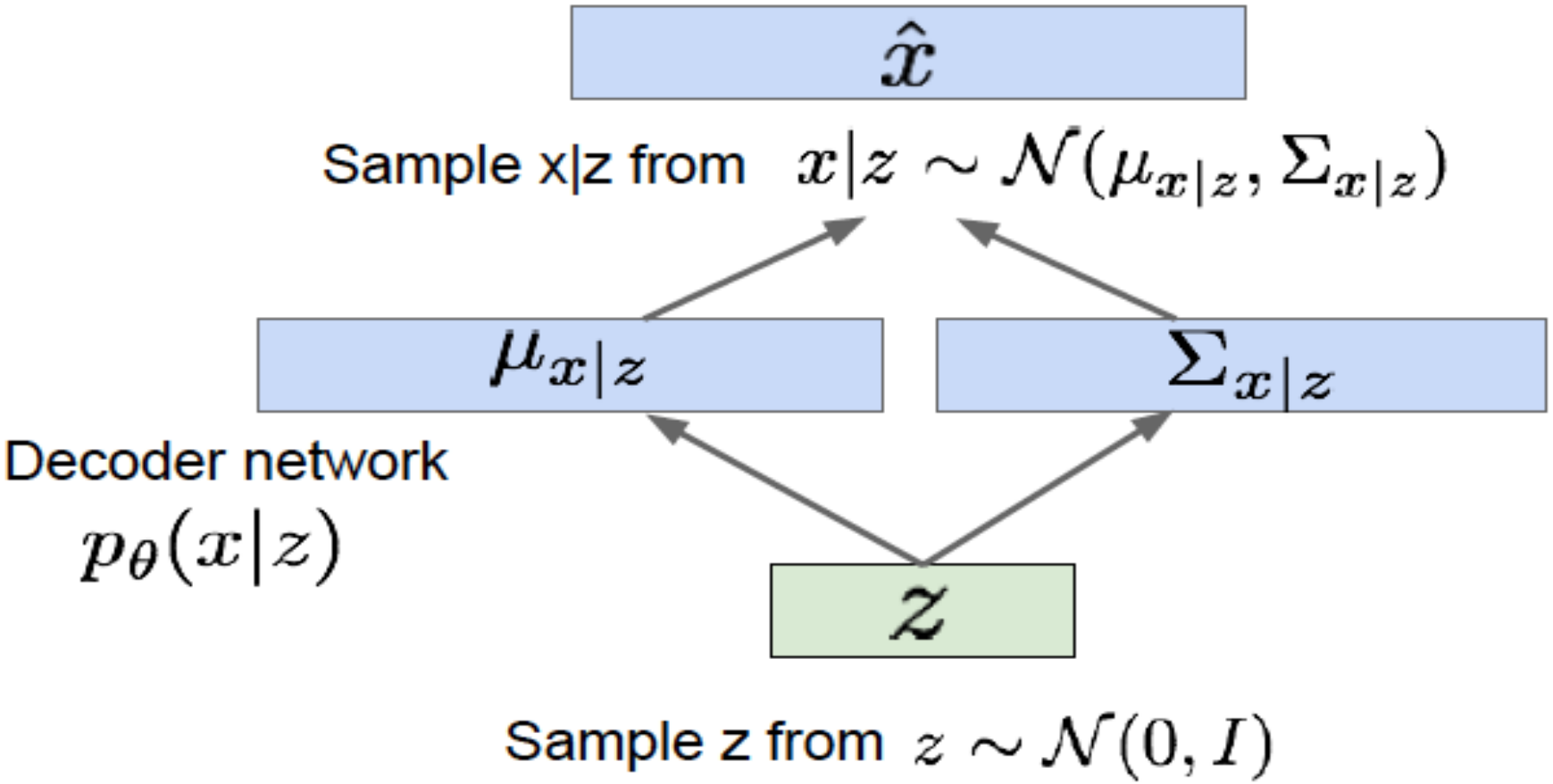
$$\hat{\theta}, \hat{\phi} = \arg\max_{\theta, \phi} \sum_{i=1}^{n} \mathcal{L}(x_i, \theta, \phi)$$

Put it all together



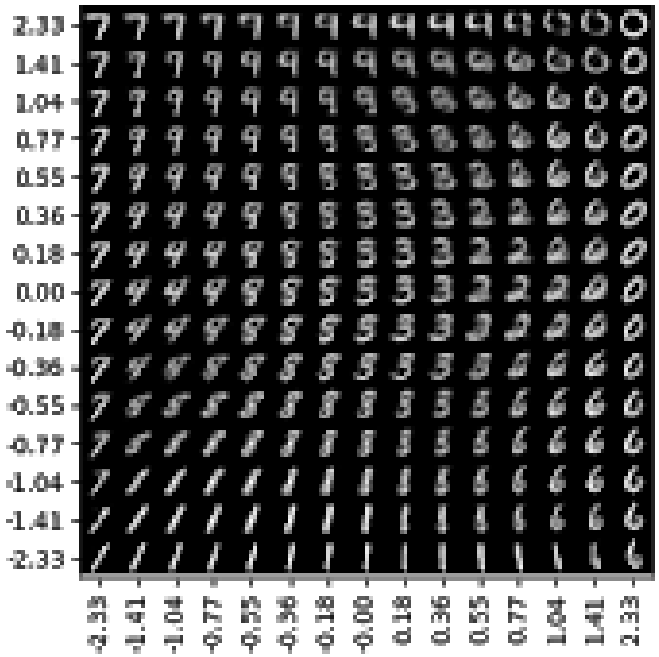
VAEs: Generating Data

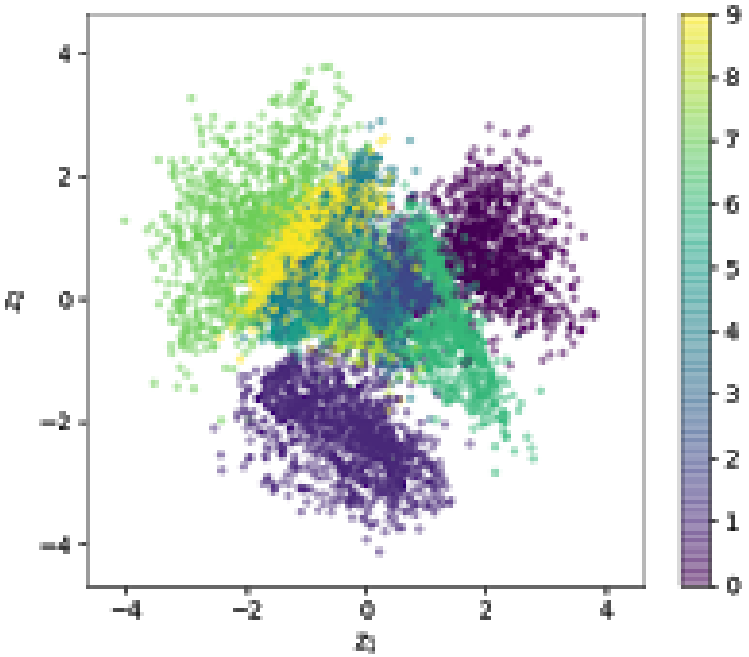
Use decoder network. Now sample z from prior!



VAEs: Generating Data

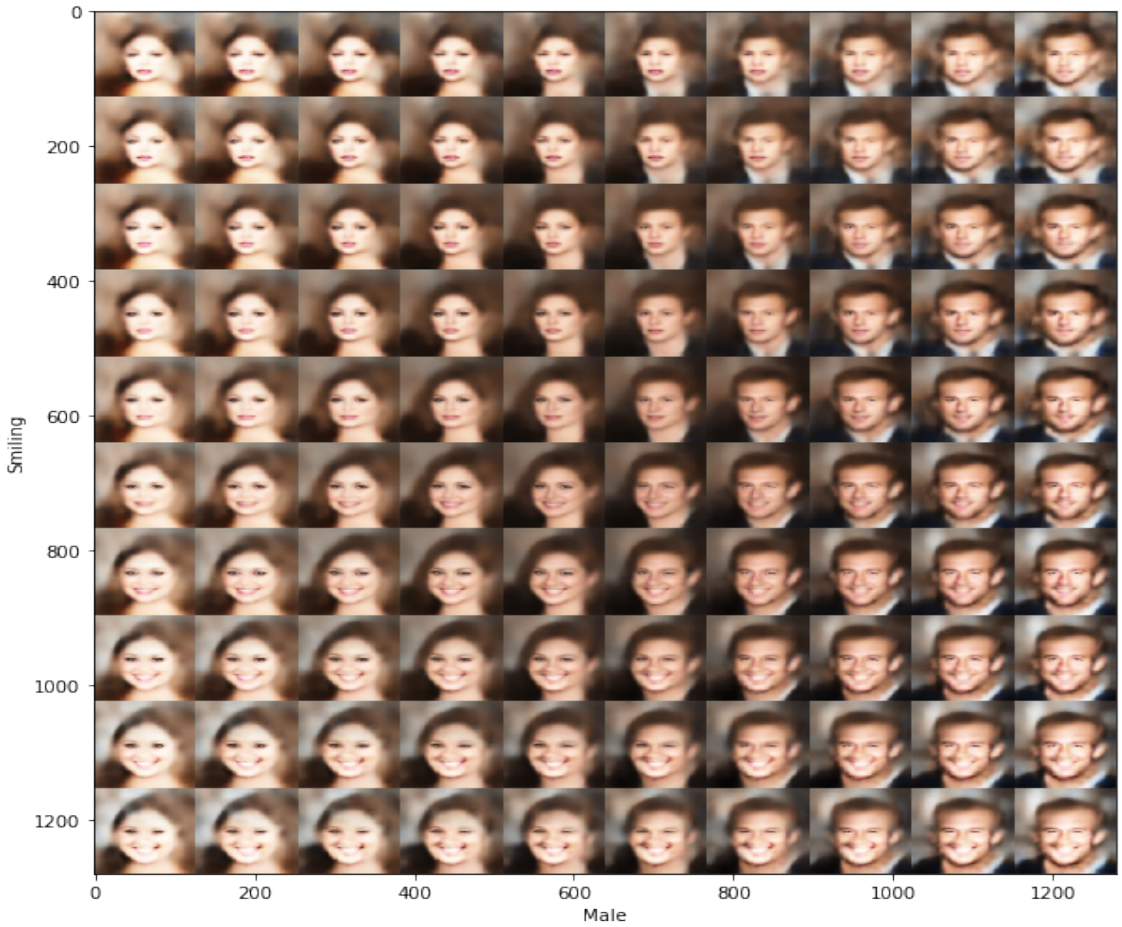
Use decoder network. Now sample z from prior!





VAEs: Generating Data

- Diagonal prior on z $\Rightarrow$ independent latent variables
- Different dimensions of z encode interpretable factors of variation
- Also good feature representation that can be computed using $q_\phi(z|x)$!



VAEs: Generating Data

VAEs

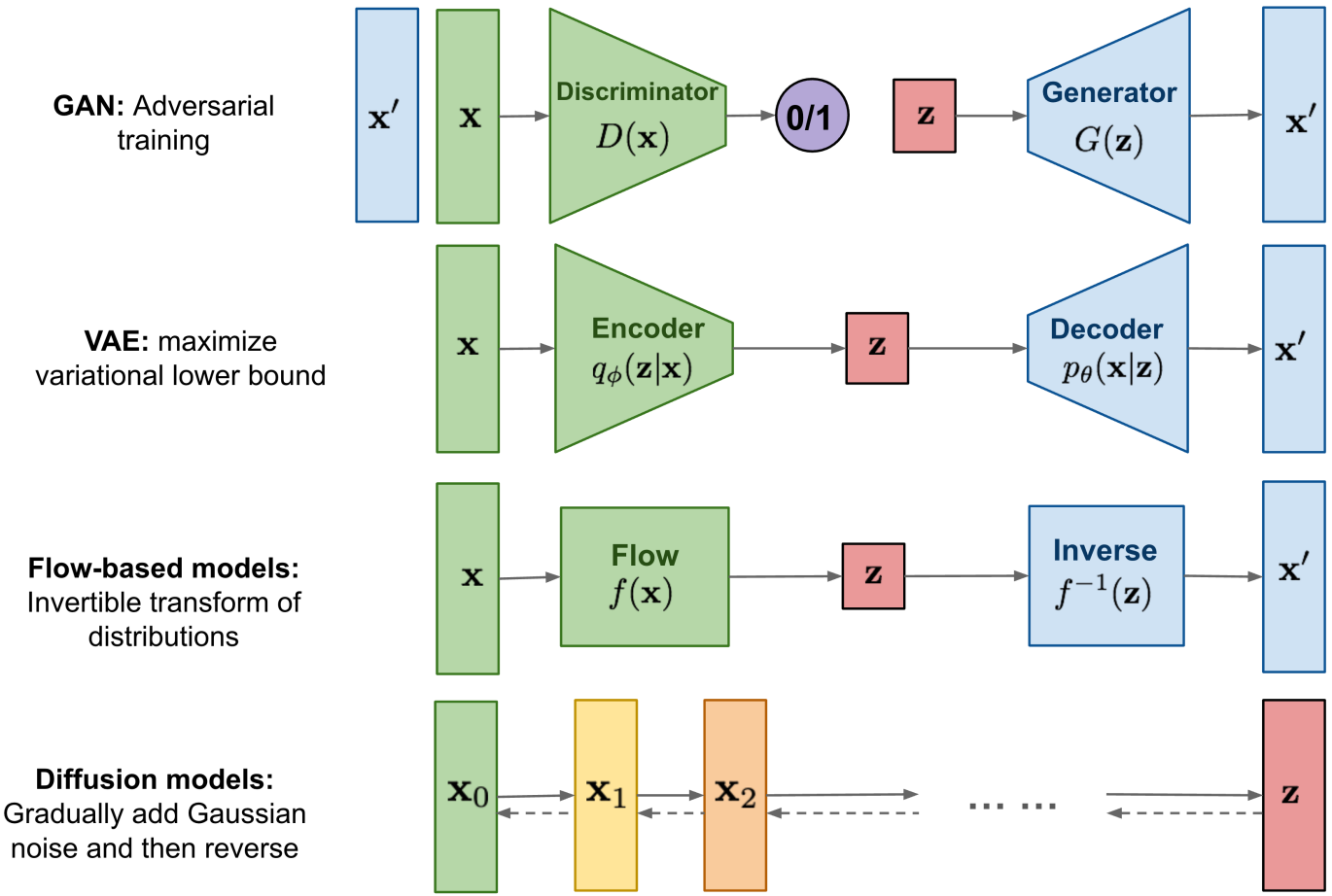
- Probabilistic spin to traditional autoencoders $\Rightarrow$ allows generating data
- Defines an intractable density $\Rightarrow$ derive and optimize a (variational) lower bound
- Pros:
  — Principled approach to generative models
  — Allows inference of $q(z|x)$, can be useful feature representation for other tasks
- Cons:
  — Maximizes lower bound of likelihood: okay, but not good enough
  — Samples blurrier and lower quality compared to state-of-the-art (GANs)

Table of contents

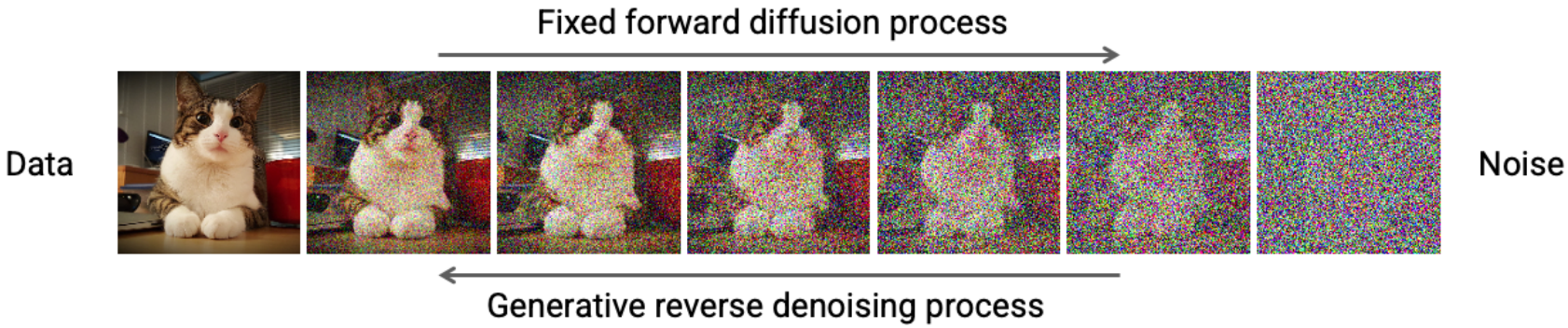
So far ...

- Flow-based models: $X = f(Z)$ where f is invertible and $Z \sim N(0, I)$. X and Z have the same dimension.
- GANs: $X = G(Z)$. The dimension of Z is much smaller than that of X. Take the game-theoretic approach.
- VAEs: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$. Maximize the ELBO.
- Diffusion models consist of two processes:
  (a) Forward diffusion process that gradually adds noise to input
  (b) Reverse denoising process that learns to generate data by denoising

GAN, VAE, Flow, and DM



Denoising Diffusion Models

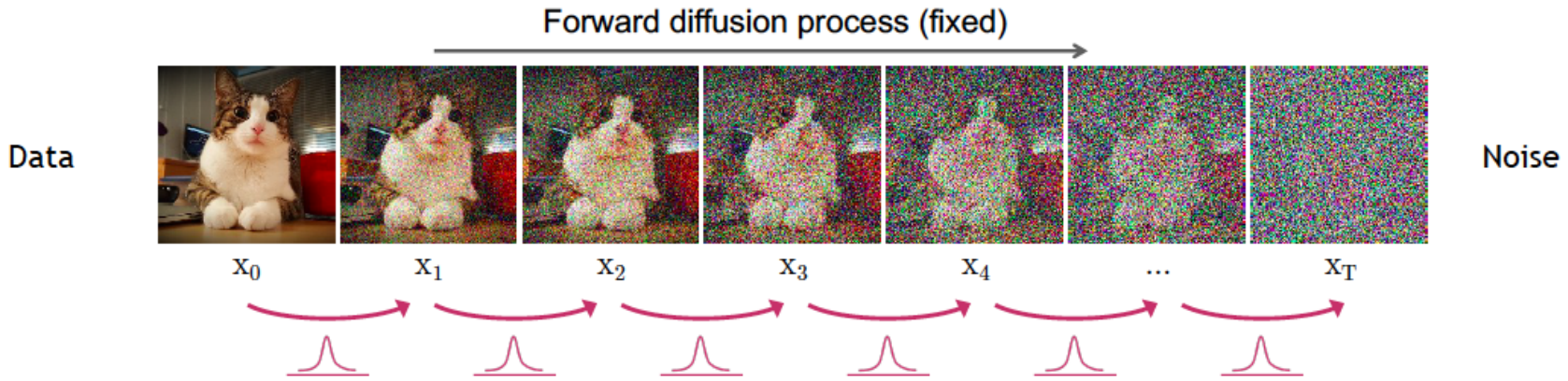


Forward Diffusion Process

- The latent dimension is exactly equal to the data dimension
- The structure of the latent encoder at each timestep is not learned; it is pre-defined as a linear Gaussian model. In other words, it is a Gaussian distribution centered around the output of the previous timestep
- The Gaussian parameters of the latent encoders vary over time in such a way that the distribution of the latent at final timestep T is a standard Gaussian

Forward Diffusion Process

The formal definition of the forward process in T steps:

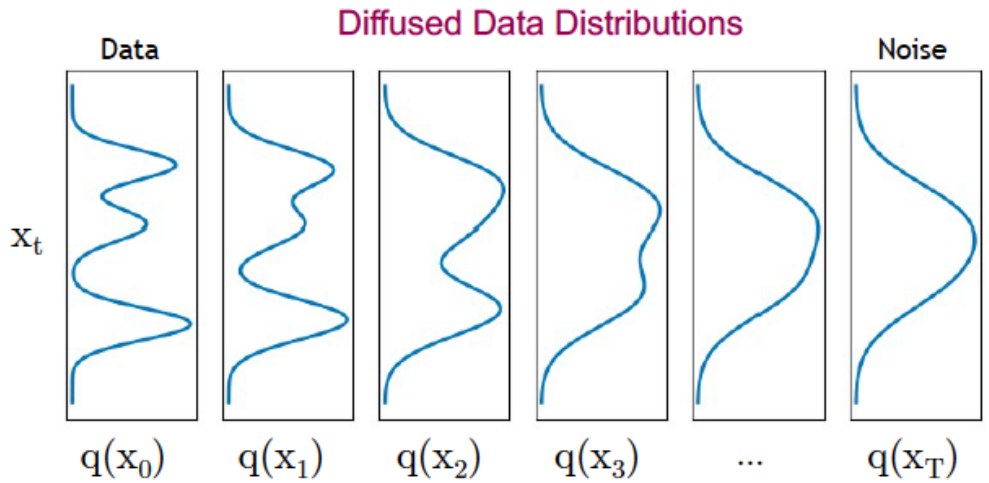


$$x_t \Big| x_{t-1} \sim N(\sqrt{1-\beta_t}\ x_{t-1}, \beta_t I)$$

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

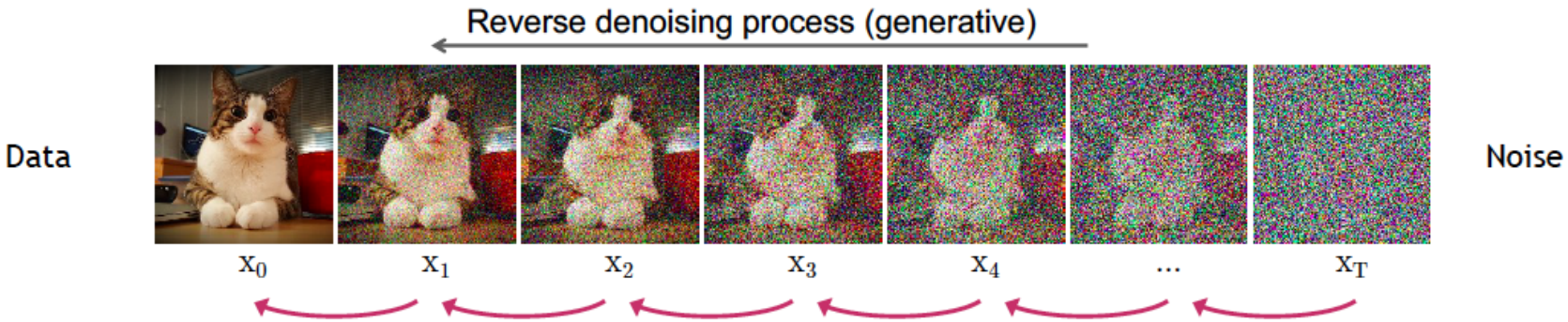
Diffusion Kernel

- Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$
- $x_t|x_0 \sim N(\sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)I)$
- For sampling: $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon$, where $\epsilon \sim N(0, I)$
- β_t is designed such that $\bar{\alpha}_T \to 0$ and $x_T|x_0 \dot{\sim} N(0, I)$
- Marginally, $q(x_t) = \int q(x_0)q(x_t|x_0)dx_0$



Reverse Denoising Process

Formal definition of reverse processes in T steps:



$$x_T \sim N(0, I), \quad x_{t-1}|x_t \sim N(\mu_\theta(x_t, t), \sigma_t^2 I)$$

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

Learning Denoising Model

- For training, we can form the ELBO

$$\mathbb{E}_{q(x_0)}[-\log p_\theta(x_0)] \leq \mathbb{E}_{q(x_0)q(x_{1:T}|x_0)}[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}] := L$$

- It is shown (Sohl-Dickstein et al. 2015 and Ho et al. 2020)that

$$\begin{aligned} L &= \mathbb{E}_q\Big[-\log p_\theta(x_0|x_1) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)) \\ &\qquad + D_{KL}(q(x_T|x_0)\|p(x_T))\Big] \\ &:= L_0 + \sum_{t>1} L_{t-1} + L_T \end{aligned}$$

where

$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)}$$

and

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

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

Learning Denoising Model

- $\mathbb{E}_{q_{x_1|x_0}}[-\log p_\theta(x_0|x_1)]$ can be interpreted as a reconstruction term; like its analogue in the ELBO of a vanilla VAE, this term can be approximated and optimized using a Monte Carlo estimate.
- $D_{KL}(q(x_T|x_0)\|p(x_T))$ represents how close the distribution of the final noisified input is to the standard Gaussian prior. It has no trainable parameters, and is also equal to zero under our assumptions.
- $\mathbb{E}_{q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t,x_0)\|p_\theta(x_{t-1}|x_t))]$ is a denoising matching term. We learn desired denoising transition step $p_\theta(x_{t-1}|x_t)$ as an approximation to tractable, ground-truth denoising transition step $q(x_{t-1}|x_t,x_0)$.

Reparameterizing the Denoising Model

- Due to the normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{KL}(q(x_{t-1}|x_t, x_0) \| p_\theta(x_{t-1}|x_t)) = \mathbb{E}_q\Big[\frac{1}{2\sigma_t^2}\|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\|^2\Big] + C$$

- Ho et al. (2020) observe that:

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

- Ho et al. (2020) propose to represent the mean of the denoising model using a noise-prediction network:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{1-\beta_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t))$$

Reparameterizing the Denoising Model

- With this reparametrization

$$L_{t-1} = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim N(0,I)} \Big[\frac{\beta_t^2}{2\sigma_t^2 (1-\beta_t)(1-\bar{\alpha}_t)} \|\epsilon - \epsilon_\theta(x_t, t)\|^2 \Big] + C$$

  where $x_t = \sqrt{\bar{\alpha}_t}\ x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$

- Ho et al. (2020) observe that simply setting the time dependent coefficient being 1 improves sample quality. So, they propose to use:

$$L_{\text{simple}} = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim N(0,I), t \sim Unif[1,T]} \Big\| \epsilon - \epsilon_\theta\big(\sqrt{\bar{\alpha}_t}\ x_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon, t\big) \Big\|^2$$

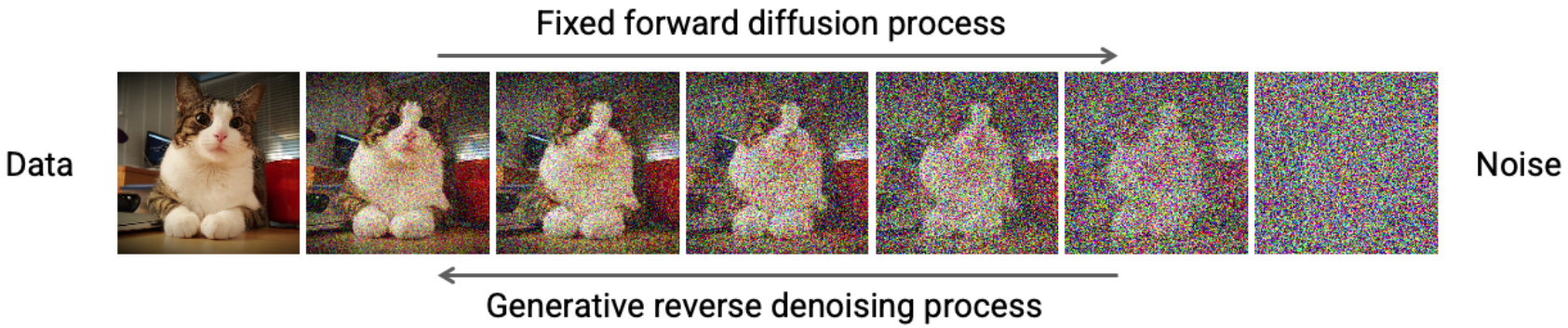
Algorithm

Algorithm 1 Training

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

Algorithm 2 Sampling

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



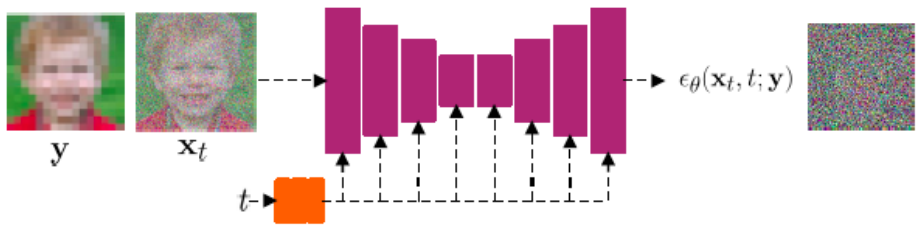
Application: Super-Resolution via Repeated Refinement (SR3)

Image super-resolution can be considered as training $p(\mathbf{x}|\mathbf{y})$ where $\mathbf{y}$ is a low-resolution image and $\mathbf{x}$ is the corresponding high-resolution image

Train a score model for $\mathbf{x}$ conditioned on $\mathbf{y}$ using:

$$\mathbb{E}_{\mathbf{x},\mathbf{y}}\, \mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\, \mathbb{E}_t\, ||\epsilon_\theta(\mathbf{x}_t, t; \mathbf{y}) - \epsilon||_p^p$$

The conditional score is simply a U-Net with $\mathbf{x_t}$ and $\mathbf{y}$ (resolution image) concatenated.



Saharia et al., Image Super-Resolution via Iterative Refinement, 2021

Application: Super-Resolution via Repeated Refinement (SR3)

Application: Image-to-Image Translation

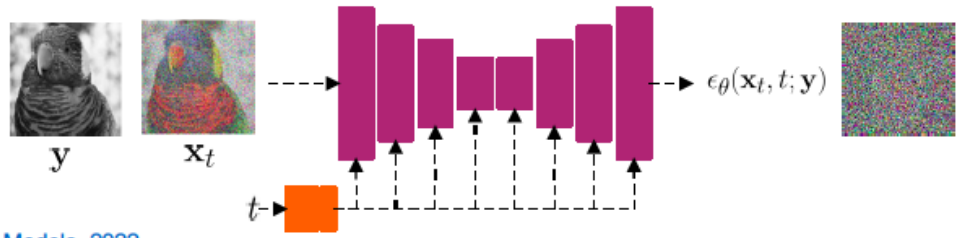
Many image-to-image translation applications can be considered as training $p(\mathbf{x}|\mathbf{y})$ where $\mathbf{y}$ is the input image.

For example, for colorization, $\mathbf{x}$ is a colored image and $\mathbf{y}$ is a gray-level image.

Train a score model for $\mathbf{x}$ conditioned on $\mathbf{y}$ using:

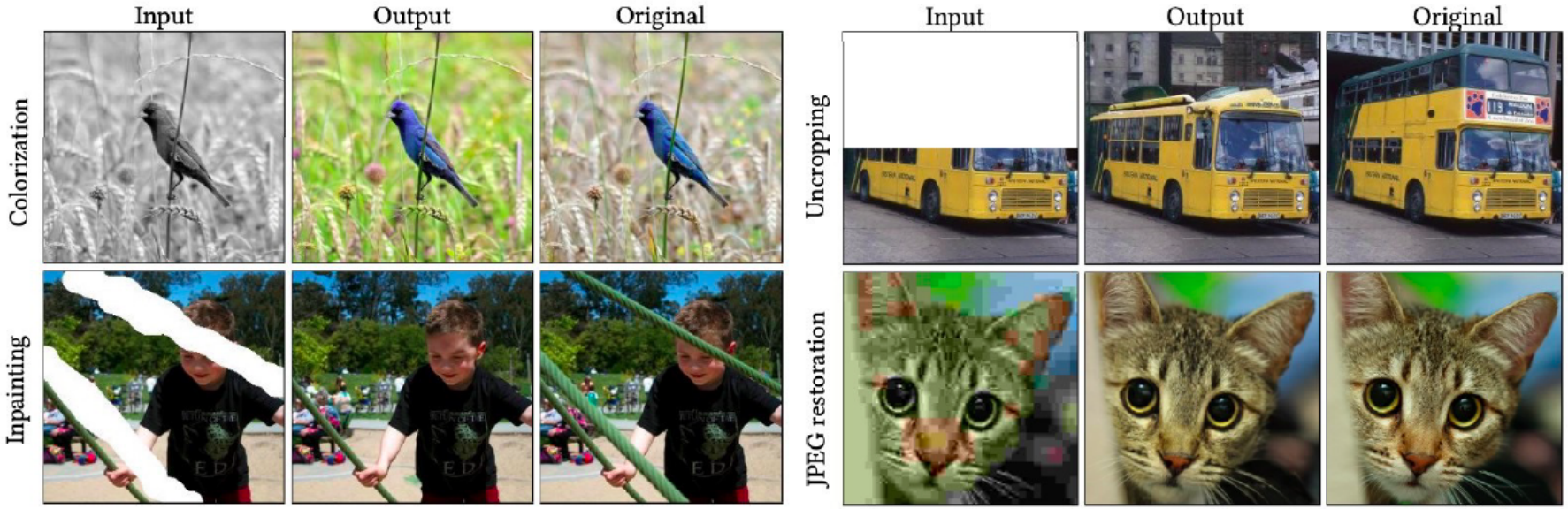
$$\mathbb{E}_{\mathbf{x},\mathbf{y}}\, \mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\, \mathbb{E}_t\, ||\epsilon_\theta(\mathbf{x}_t, t; \mathbf{y}) - \epsilon||_p^p$$

The conditional score is simply a U-Net with $\mathbf{x_t}$ and $\mathbf{y}$ concatenated.



Saharia et al., Palette: Image-to-Image Diffusion Models, 2022

Application: Image-to-Image Translation



Application: Text-to-Image Generation

- Text-to-Image generation: Midjourney, Stable Diffusion, DALL·E 2, Imagen,

Imagen

Google Research, Brain team

A relaxed garlic with a blindfold reading a newspaper while floating in a pool of tomato soup.

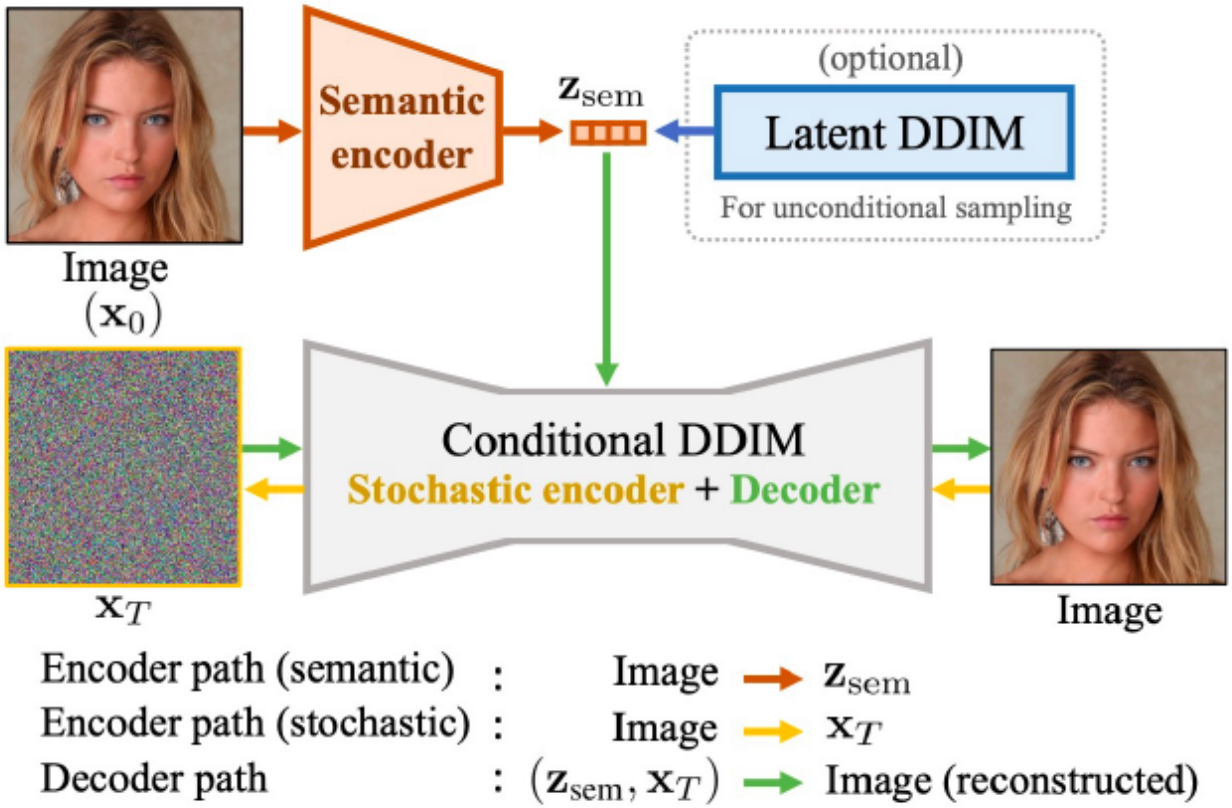
Imagen

Google Research, Brain team

A cute hand-knitted koala wearing a sweater with 'CVPR' written on it.

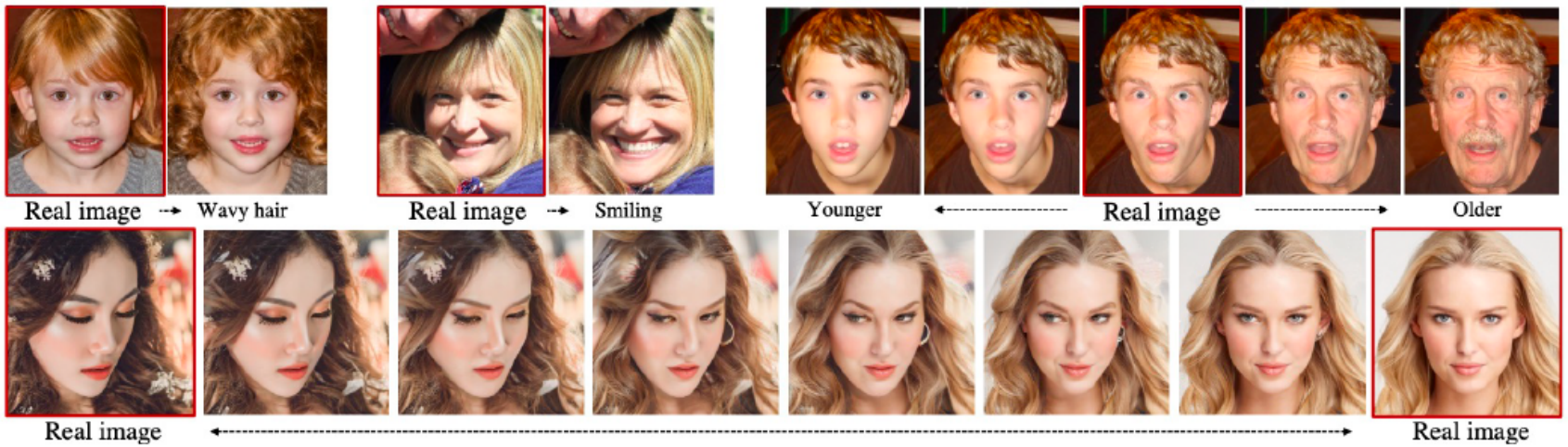
Diffusion Autoencoders

Learning semantic meaningful latent representations in diffusion models (Preechakul et al. 2022)



Diffusion Autoencoders

Learning semantic meaningful latent representations in diffusion models (Preechakul et al. 2022)



Diffusion Autoencoders

Learning semantic meaningful latent representations in diffusion models (Preechakul et al. 2022)

Open Problems

- Sampling from diffusion model is still slow — How can one sample with even fewer steps?
- How good is the latent space of diffusion model for downstream tasks?
  — ResNet on ImageNet gives us great image features
  — LLM gives great text features
  — Can diffusion model beat imagenet feature?
  — Can diffusion model help us in discriminative tasks?

Open Problems

- How do we better control diffusion model for conditional generation?
  — With GANs we can do very fine-grained condition and can even use 3D intrinsics.
  — While Diffusion model produces impressive image quality, it often ignores detailed conditioning like segmentation mask, and works best for vague conditioning like text or class.
  — How do we enable that?
- Can we generate more involved videos with diffusion models?
- Many many cool and creative applications that we haven't imagined before due to lack of such a powerful tool!
- Do you think GAN would disappear in few years and Diffusion model would be the de-facto for generative models?

A very partial list of References

1. Feifei Li's course slides in Stanford University
2. Aaron Courville, Ian Goodfellow, and Yoshua Bengio (2015). *Deep Learning*. The MIT Press.
3. `https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html`
4. Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP." ICLR 2017.
5. Lilian Weng. `https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html#vae--flows`
6. Pieter Abbeel, et al., *CS294-158 Deep Unsupervised Learning Lecture 3 Likelihood Models: Flow Models*, Spring 2020, UC Berkeley
7. Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
8. Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
9. Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021
10. `https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html`
11. `https://cvpr2022-tutorial-diffusion-models.github.io/`
12. Calvin Luo. "Understanding Diffusion Models: A Unified Perspective." 2022.

Table of contents

- In this section, we are going to implement and train generative models, including GAN, VAE and DDPM model.
- Please open generative_models folder and open generative_models.py.


Thank You!