Introduction to Computer Vision

Lecture 15

Generative Model

Prof. He Wang

Logistics

- Assignment 4 (Point Cloud Learning, Detection & RNN)
  - Released on 5/24
  - Due on 6/8 11:59PM

Logistics

- Final exam
  - Time: 6/18
  - Scope: all the lectures after midterm including today's lecture.

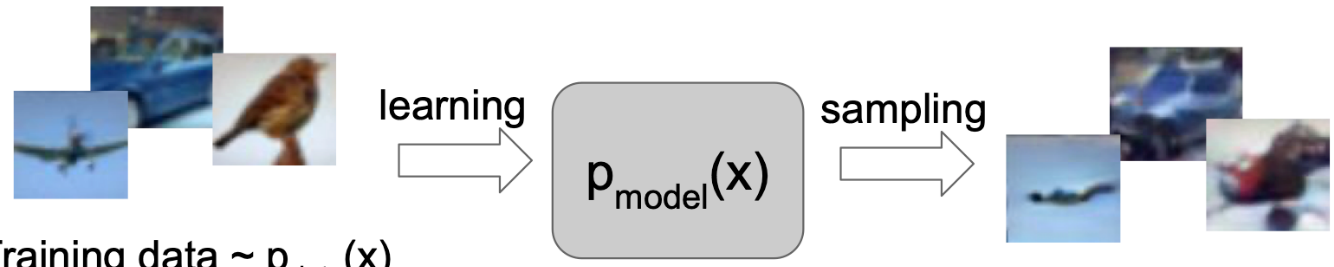
  - Question types: similar to midterm exam.
  - In English, all terms included in our slides won't be explained.
  - 1-page A4-size cheat sheet is allowed .

Generative Models

Some slides are borrowed from Stanford CS231N.

Generative Modeling

Given training data, generate new samples from same distribution



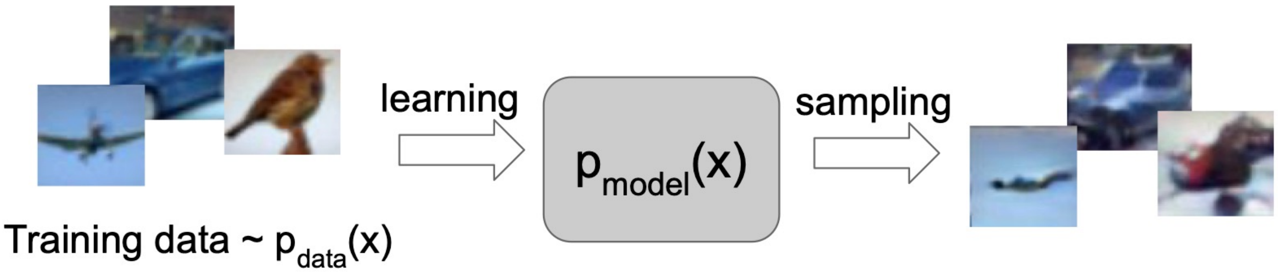
Training data ~ $p_{data}(x)$

Objectives:

1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. **Sampling new x from $p_{model}(x)$**

Generative Modeling

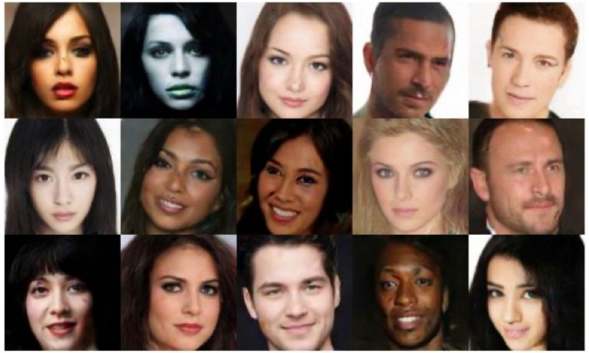
Given training data, generate new samples from same distribution



Formulate as density estimation problems:

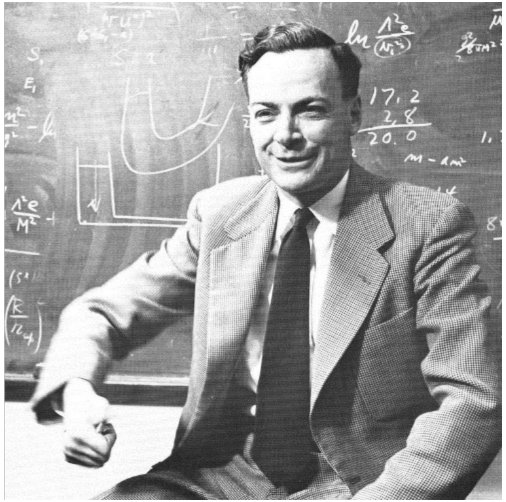
- **Explicit density estimation**: explicitly define and solve for $p_{model}(x)$
- **Implicit density estimation**: learn model that can sample from $p_{model}(x)$ **without explicitly defining it.**

Why Generative Model?


- Realistic samples for artwork, super-resolution, colorization, etc.
- Learn useful features for downstream tasks such as classification.
- Getting insights from high-dimensional data (physics, medical imaging, etc.)
- Modeling physical world for simulation and planning (robotics and reinforcement learning applications)
- Many more ...

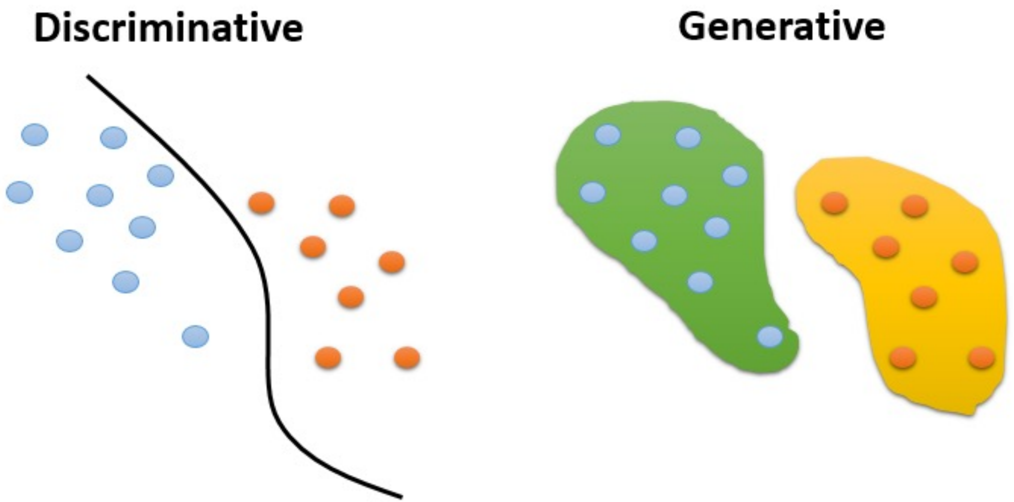
Generative Modeling



Richard Feynman: "*What I cannot create, I do not understand*"

Generative modeling: "*What I understand, I can **create***"

Discriminative vs. Generative



- Y: labels, X: inputs
- Learn $P(Y|X)$

- X is all the variables
- $P(X)$ or $P(X, Y)$ (if labels are available)

Taxonomy of Generative Model

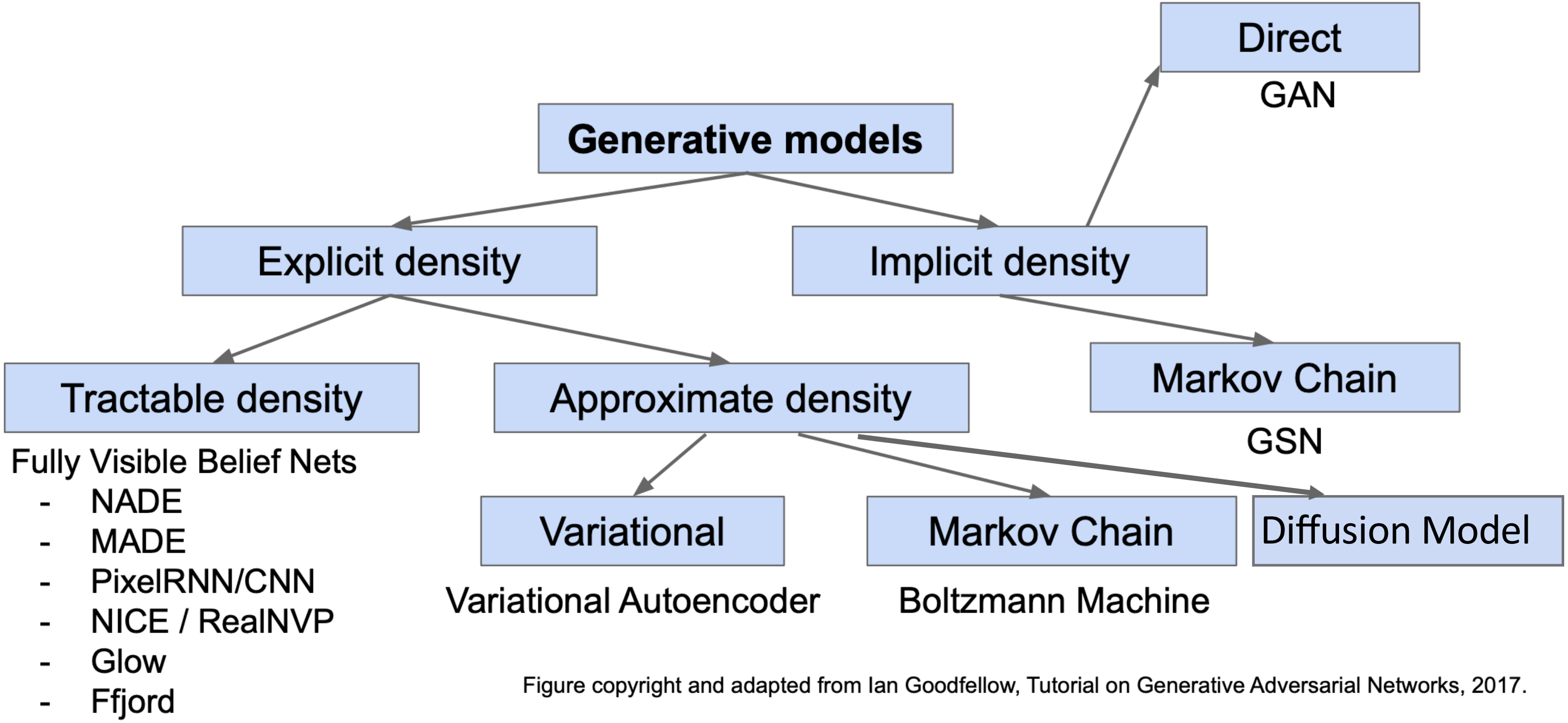


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Model

Today we mainly discuss
the most popular three

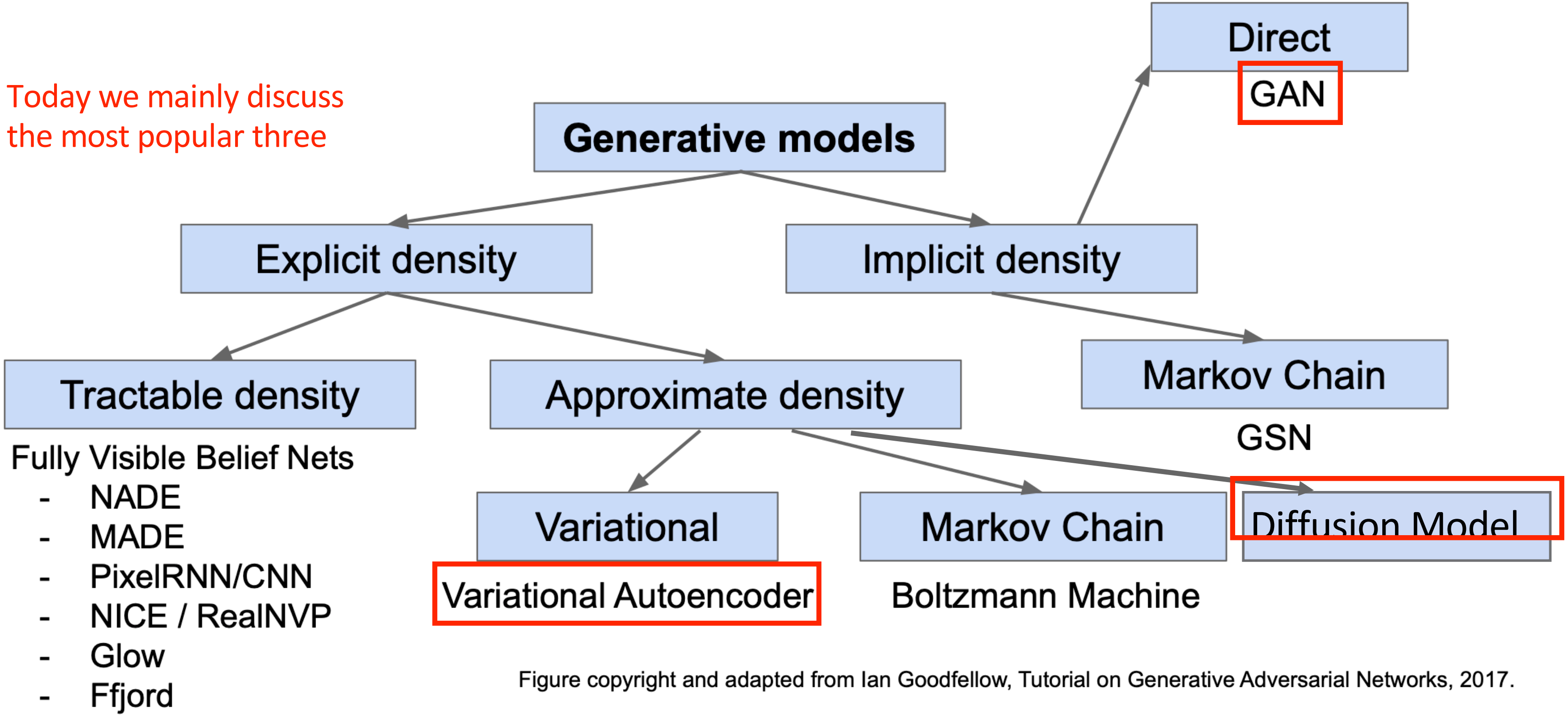


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Fully Visible Belief Network (FVBN)

Explicit density model

$$p(x) = p(x_1, x_2, \ldots, x_n)$$

Likelihood of image x

Joint likelihood of each pixel in the image

Fully Visible Belief Network (FVBN)

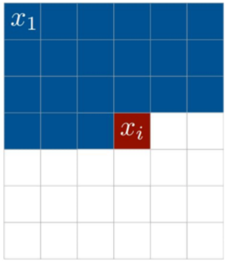
Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels



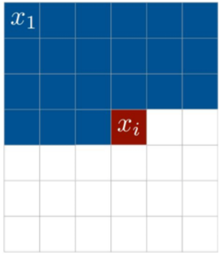
Then maximize likelihood of training data

Fully Visible Belief Network (FVBN)

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1})$$



Likelihood of image x

Probability of i'th pixel value given all previous pixels

Then maximize likelihood of training data

Complex distribution over pixel values => Express using a neural network!

Taxonomy of Generative Model

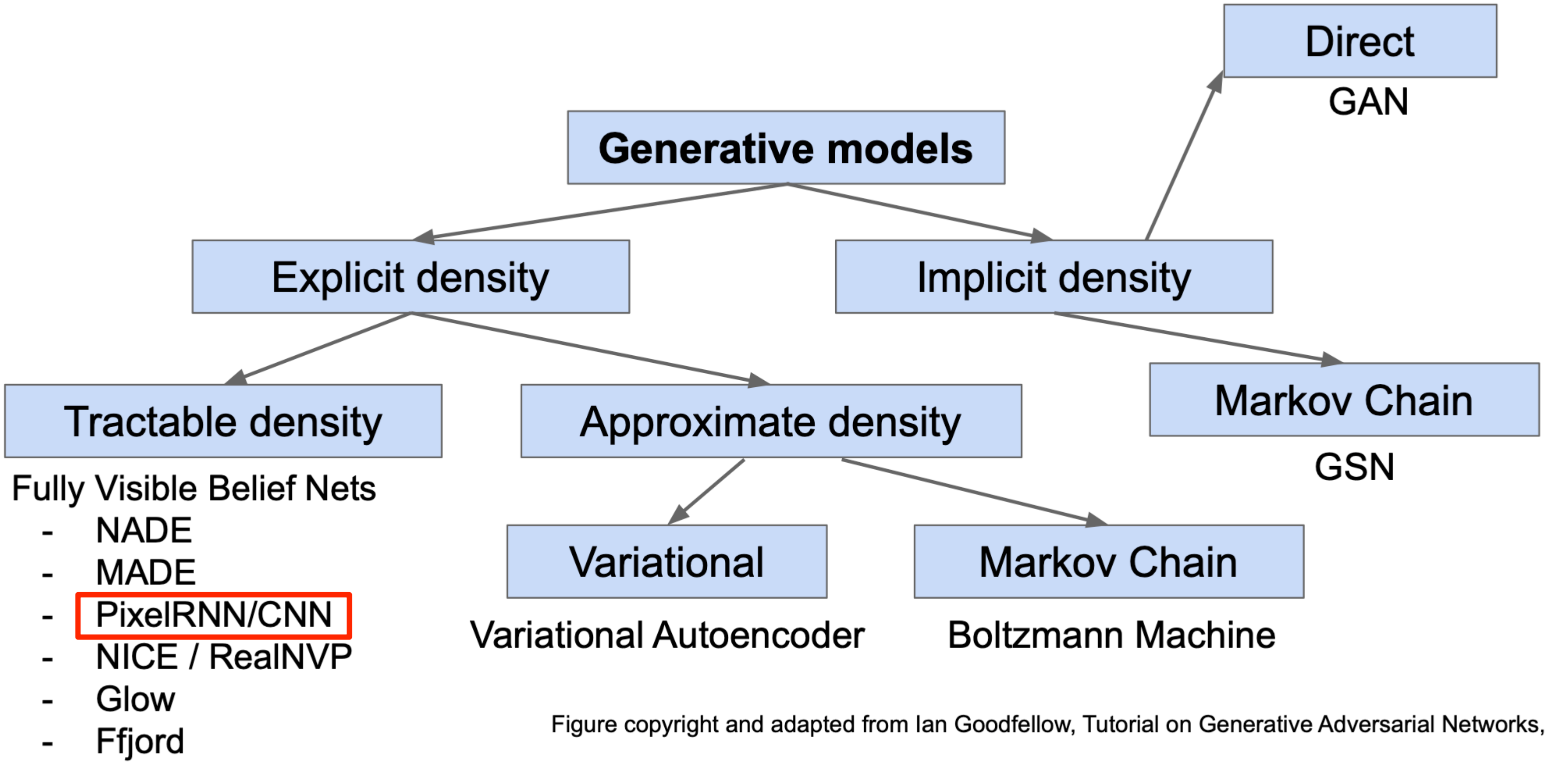


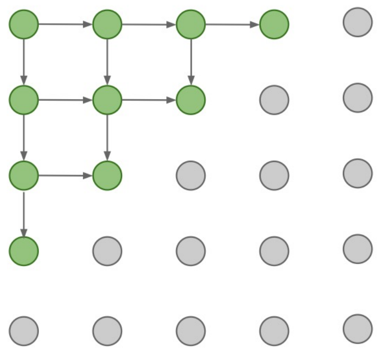
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

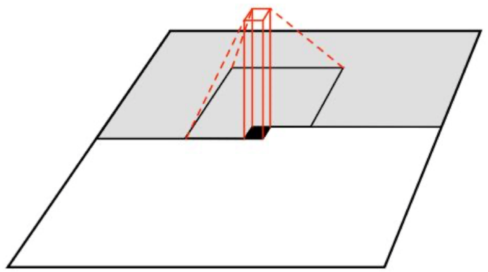
PixelRNN and PixelCNN

Pros:
- Can explicitly compute likelihood p(x)
- Easy to optimize
- Good samples

Con:
- Sequential generation => slow

PixelRNN





PixelCNN

Variational Autoencoders (VAE)

Some slides are borrowed from Stanford CS231N.

Taxonomy of Generative Model

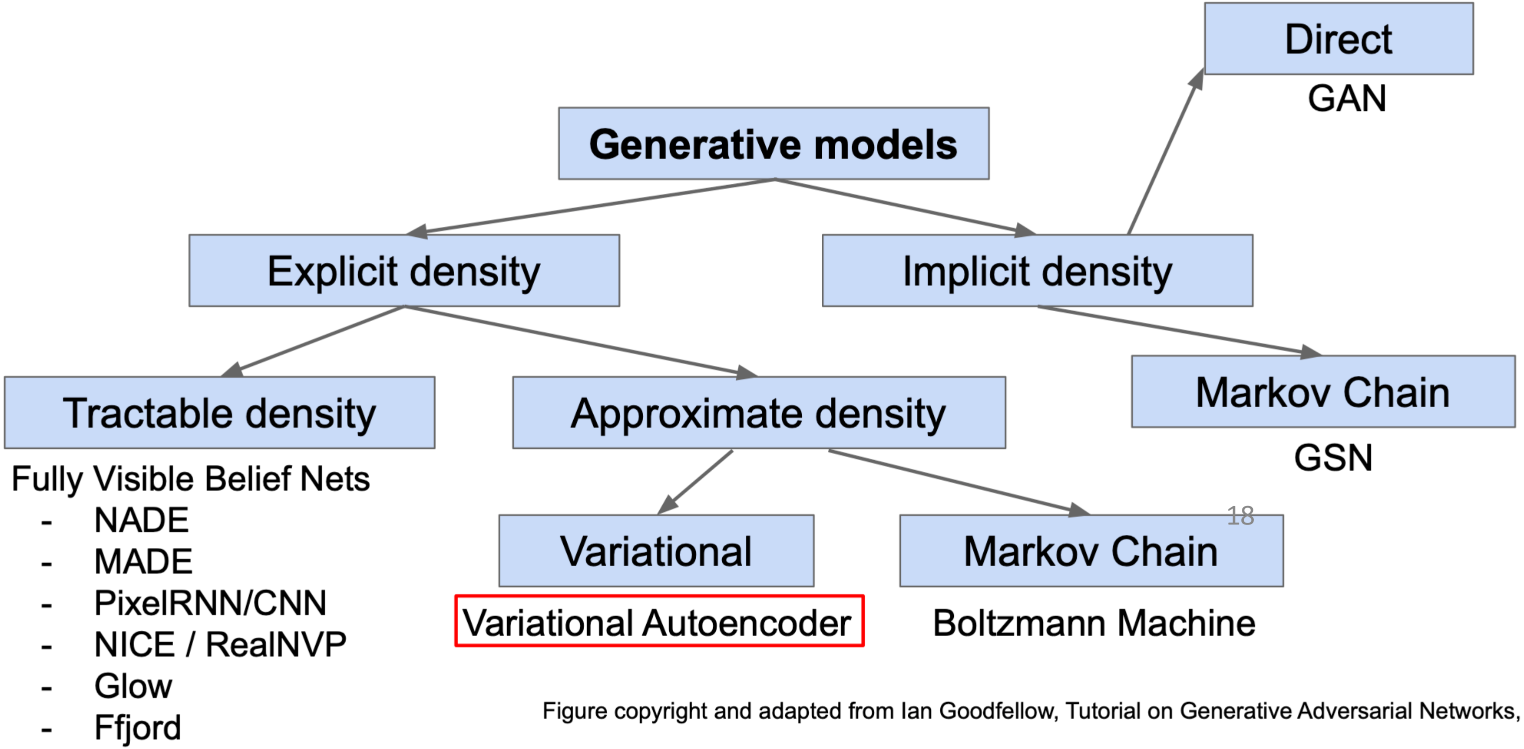


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

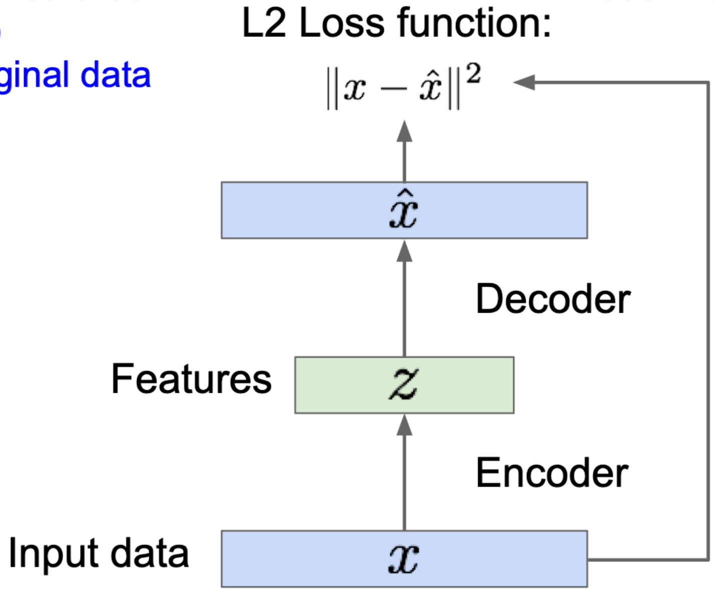
Variational Autoencoders

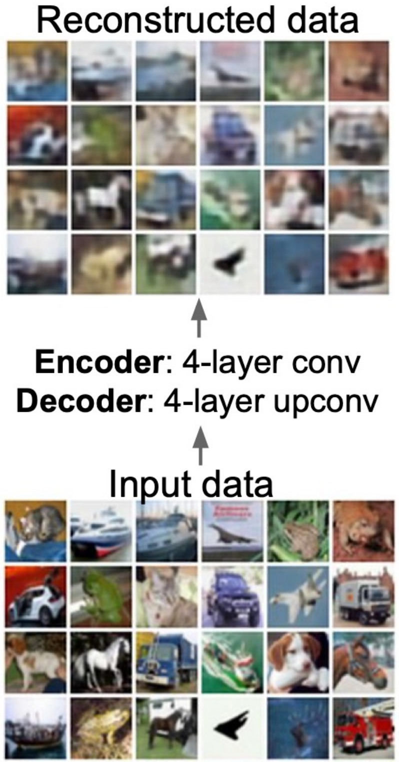
Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Recap of Autoencoder

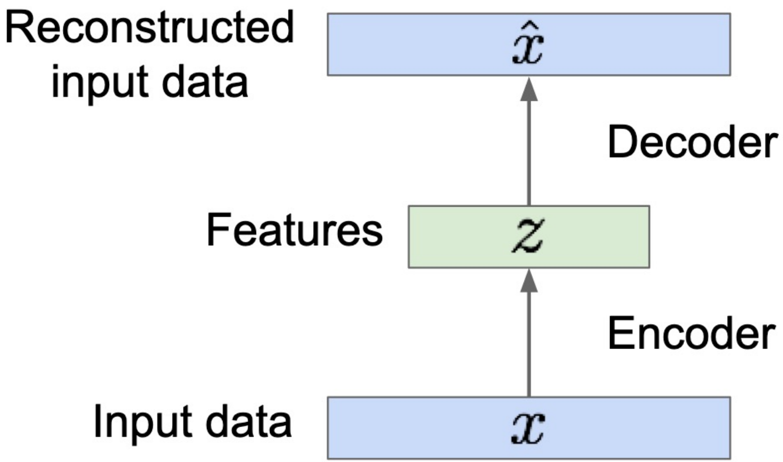
Train such that features can be used to reconstruct original data

Doesn't use labels!





Recap of Autoencoder



Autoencoders can reconstruct data, and can learn features to initialize a supervised model

Features capture factors of variation in training data.

But we can't generate new images from an autoencoder because we don't know the space of z.

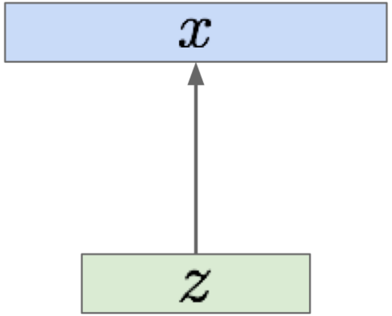
How do we make autoencoder a **generative model**?

Variational Autoencoders

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 the distribution of unobserved (latent) representation **z**

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



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

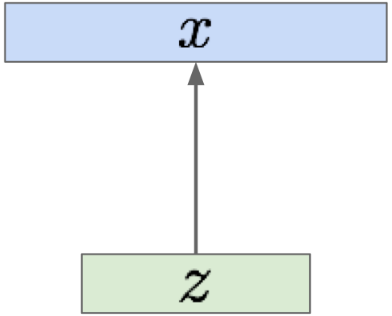
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

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 the distribution of unobserved (latent) representation **z**

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

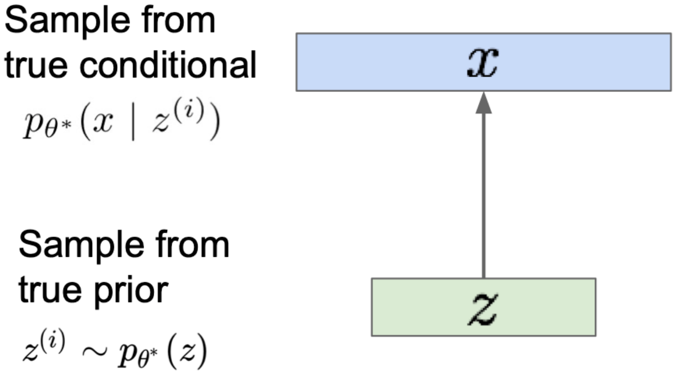


Sample from
true prior
$z^{(i)} \sim p_{\theta^*}(z)$

Intuition (remember from autoencoders!):
x is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders



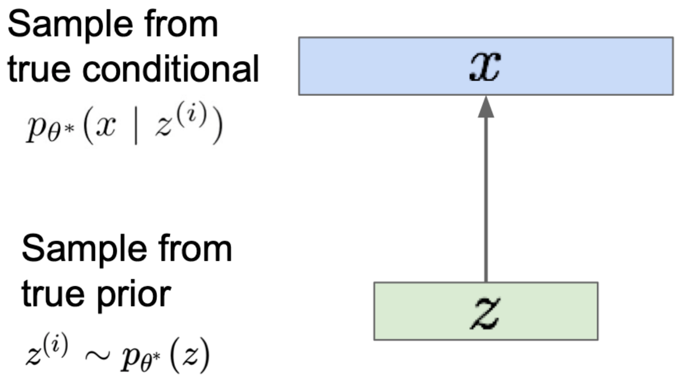
We want to estimate the true parameters θ^* of this generative model given training data x.

How should we represent this model?

Choose prior $p(z)$ to be simple, e.g. a standard normal distribution $\mathcal{N}(0, I)$. Reasonable for latent attributes, e.g. pose, how much smile.

Conditional $p(x|z)$ is complex (generate image) => represent with a probabilistic neural network that is also a Gaussian.

Variational Autoencoders



We want to estimate the true parameters θ^* of this generative model given training data x.

How to train?

Learn model parameters to maximize likelihood of training data

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

Q: What is the problem with this?

Intractable!

Variational Autoencoders: Intractability

Data likelihood: $p_\theta(x) = \int p(z) p_\theta(x|z) dz$

Standard normal distribution $\mathcal{N}(0, I)$

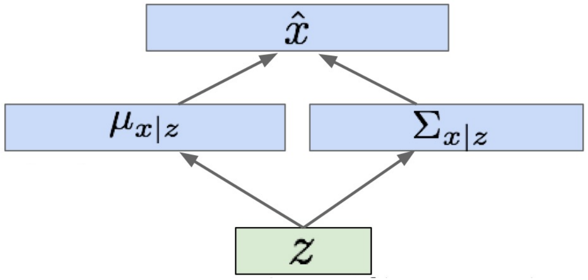
Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

Variational Autoencoders: Intractability

Data likelihood: $$p_\theta(x) = \int p(z) p_\theta(x|z) dz$$

Decoder neural network

Note that this decoder $p_\theta(x|z)$ needs to be a probabilistic function, we will assume this probability distribution is also a Gaussian and then this decoder network only needs to predict $\mu_{x|z}, \Sigma_{x|z}$.



Probabilistic decoder $p_\theta(x|z)$

Variational Autoencoders: Intractability

Data likelihood: $$p_\theta(x) = \int p(z)p_\theta(x|z)dz$$

Intractable to compute p(x|z) for every z!

Variational Autoencoders: Intractability

Data likelihood: $$p_\theta(x) = \int p(z) p_\theta(x|z) dz$$

Intractable to compute p(x|z) for every z!

$$p_\theta(x) = \mathbf{E}_{z \sim p(z)}[p_\theta(x|z)]$$

Can we use Monto Carlo estimation?

$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^{k} p_\theta(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z)$$

Unbiased but the variance is very high!

Variational Autoencoders: Intractability

Data likelihood: $p_\theta(x) = \int p(z) p_\theta(x|z) dz$ Computing integral is intractable

Try another way:

$$p_\theta(x) = \frac{p_\theta(x,z)}{p_\theta(z|x)} = \frac{1}{p_\theta(z|x)} p(z) p_\theta(x|z)$$

Standard normal distribution $\mathcal{N}(0, I)$

Variational Autoencoders: Intractability

Data likelihood: $p_\theta(x) = \int p(z)p_\theta(x|z)dz$ Computing integral is intractable

Try another way:

$$p_\theta(x) = \frac{p_\theta(x,z)}{p_\theta(z|x)} = \frac{1}{p_\theta(z|x)}p(z)p_\theta(x|z)$$

✔

Probabilistic decoder

Variational Autoencoders: Intractability

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$ Computing integral is intractable

Try another way:

$$p_\theta(x) = \frac{p_\theta(x,z)}{p_\theta(z|x)} = \frac{1}{p_\theta(z|x)} p(z) p_\theta(x|z)$$

?

Variational Autoencoders: Intractability

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$ Computing integral is intractable

Try another way: $p_\theta(x) = \dfrac{p_\theta(x,z)}{p_\theta(z|x)} = \dfrac{p_\theta(z) p_\theta(x|z)}{p_\theta(z|x)}$

Unfortunately, all we know about this term is

$$p_\theta(z|x) = \frac{p_\theta(x,z)}{p_\theta(x)} = \frac{p(z) p_\theta(x|z)}{p_\theta(x)}$$

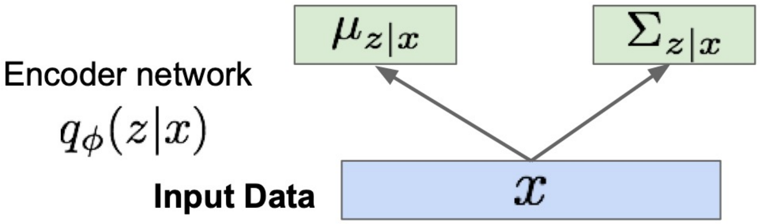
Intractable data likelihood

Variational Autoencoders

Can we learn a distribution $q_{\phi}(z|x)$ to approximate $p_{\theta}(z|x)$?

Probabilistic encoder

The probabilistic encoder $q_{\phi}(z|x)$ will also be a Gaussian distribution, which takes input x and outputs $\mu_{z|x}, \Sigma_{z|x}$.

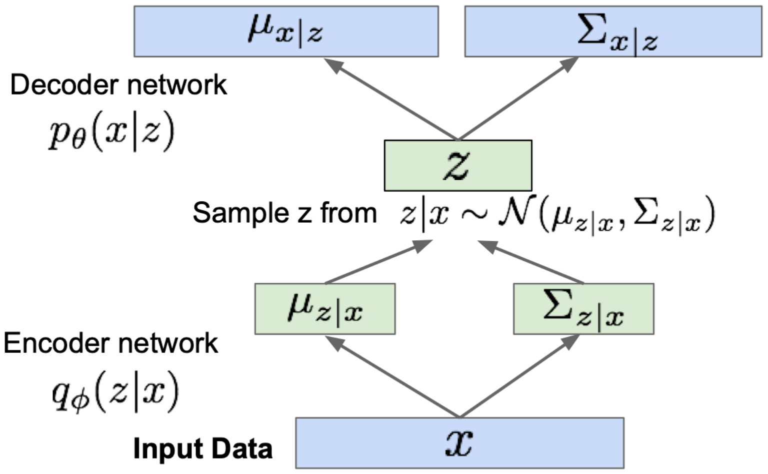


How to Learn Variational Autoencoders

- VAE is a probabilistic autoencoder.

- How to learn:
    - Build a loss (negative log-likelihood) $\mathcal{L} = -\log p_{\theta,\phi}(x)$
    - Minimize $\mathcal{L}$ with respect to ϕ and θ (or maximize $\log p_{\theta,\phi}(x)$)
    - However, this term $\log p_{\theta,\phi}(x)$ is still intractable

Variational Autoencoder (VAE)



Variational Autoencoders

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p\ (z)}{p_\theta(z \mid x^{(i)})}\right] \qquad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p\ (z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \qquad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \qquad \text{(Logarithms)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p\ (z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

The expectation wrt. z (using encoder network) let us write nice KL terms

Variational Autoencoders

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)\, p(z)}{p_\theta(z \mid x^{(i)})}\right] \qquad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)\, p(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \qquad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \qquad \text{(Logarithms)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

We want to maximize the data likelihood

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling (need some trick to differentiate through sampling).

Variational Autoencoders

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)\, p\,(z)}{p_\theta(z \mid x^{(i)})}\right] \qquad (\text{Bayes' Rule}) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)\, p\,(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \qquad (\text{Multiply by constant}) \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \qquad (\text{Logarithms}) \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p\,(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

We want to maximize the data likelihood

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling (need some trick to differentiate through sampling).

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

ELBO

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(z \mid x^{(i)})}\right] \qquad (\text{Bayes' Rule}) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \qquad (\text{Multiply by constant}) \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \qquad (\text{Logarithms}) \\
&= \boxed{\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\boxed{\geq 0}}
\end{aligned}
$$

We want to maximize the data likelihood

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

This lower bound is widely referred as **Evidence Lower BOund** (**ELBO**).

ELBO

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

Encoder: make approximate posterior distribution close to prior

Decoder: reconstruct the input data

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \boxed{\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\boxed{\geq 0}}$$

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

This lower bound is widely referred as Evidence Lower BOund (**ELBO**).

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound (**ELBO**)

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \boxed{D_{KL}(q_\phi(z \mid x^{(i)}) \mid\mid p(z))}}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

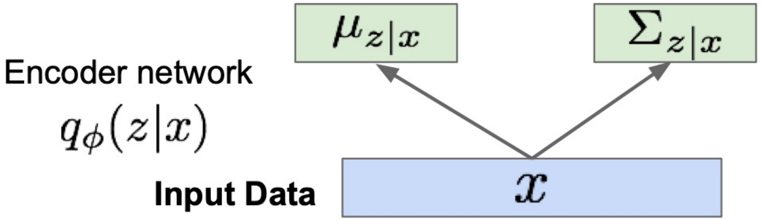
Let's look at computing the KL divergence between the estimated posterior and the prior given some data

Input Data x

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \boxed{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \boxed{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

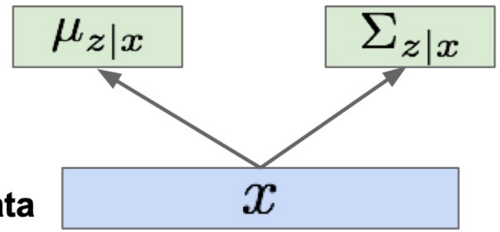
$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) || \mathcal{N}(0, I))$$

Have analytical solution

Make approximate posterior distribution close to prior

Encoder network

$q_\phi(z|x)$



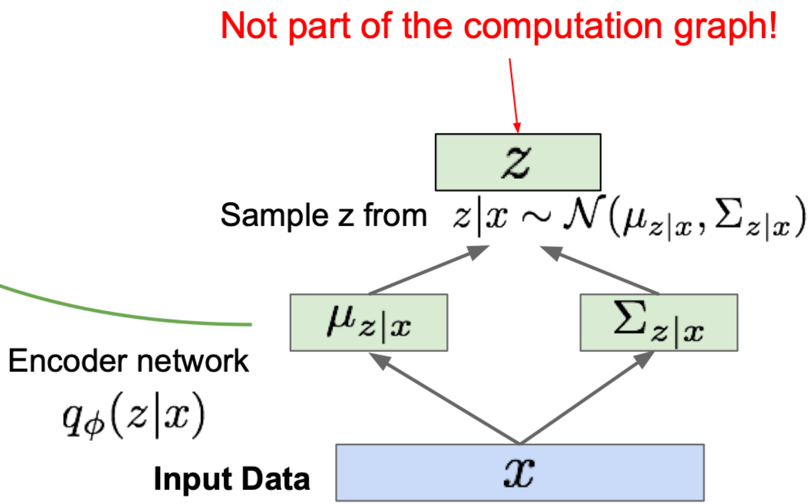
Input Data

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Make approximate posterior distribution close to prior



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

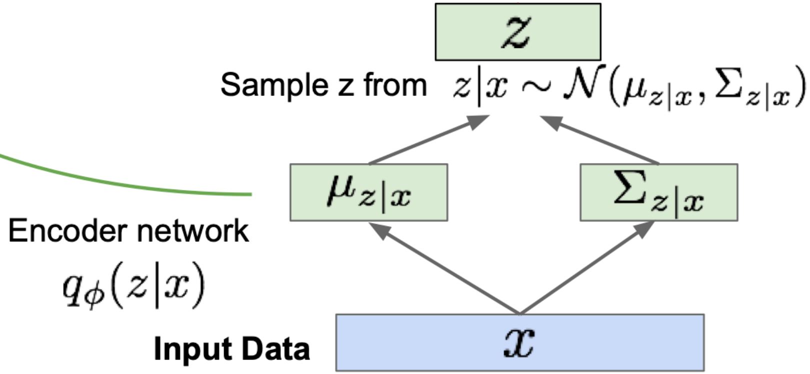
$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Reparameterization trick to make sampling differentiable:

Sample $\epsilon \sim \mathcal{N}(0, I)$

$$z = \mu_{z|x} + \epsilon\sigma_{z|x}$$

Make approximate posterior distribution close to prior



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Make approximate posterior distribution close to prior

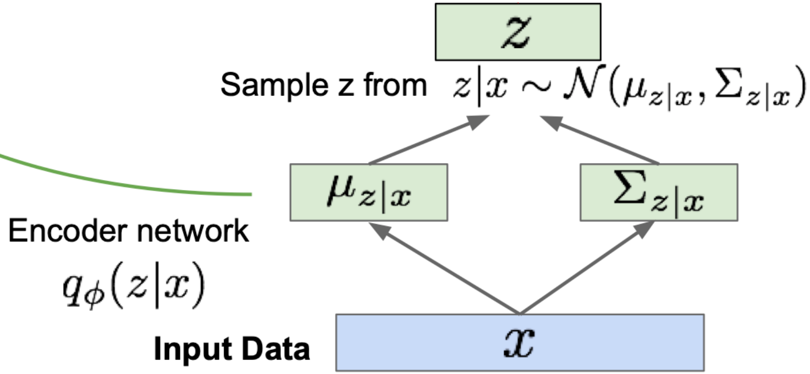
Reparameterization trick to make sampling differentiable:

Sample $\epsilon \sim \mathcal{N}(0, I)$

$$z = \boxed{\mu_{z|x}} + \boxed{\epsilon}\boxed{\sigma_{z|x}}$$

Input to the graph

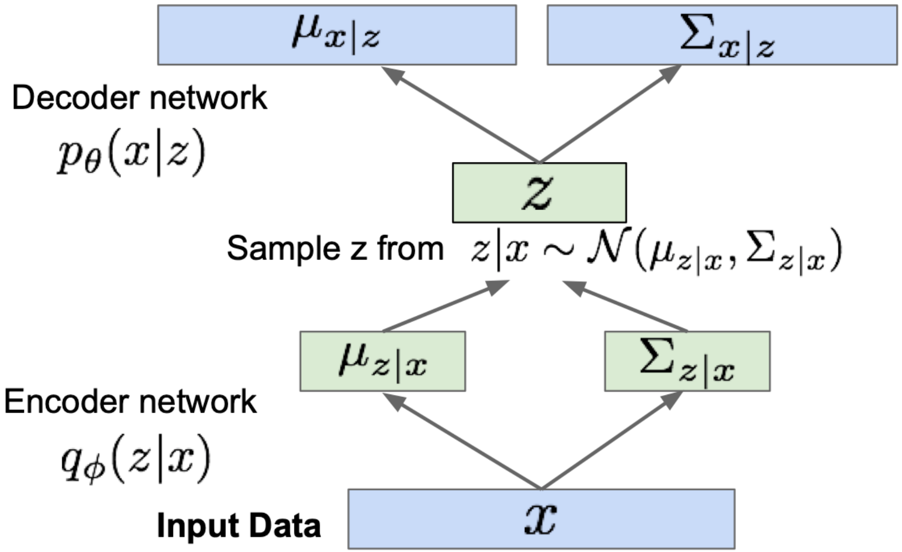
Part of computation graph



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

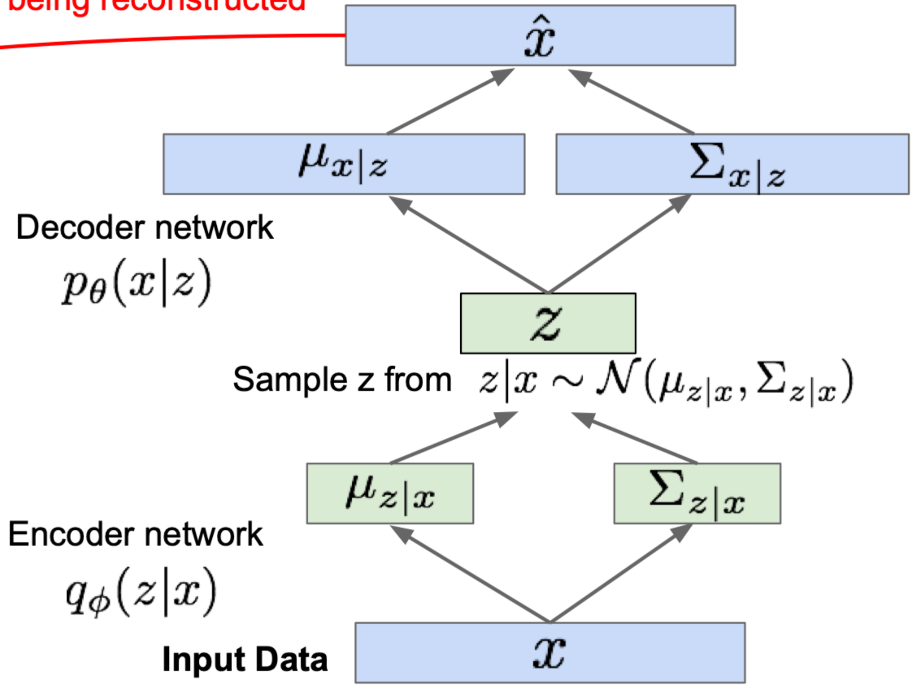


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

Maximize likelihood of original input being reconstructed

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$



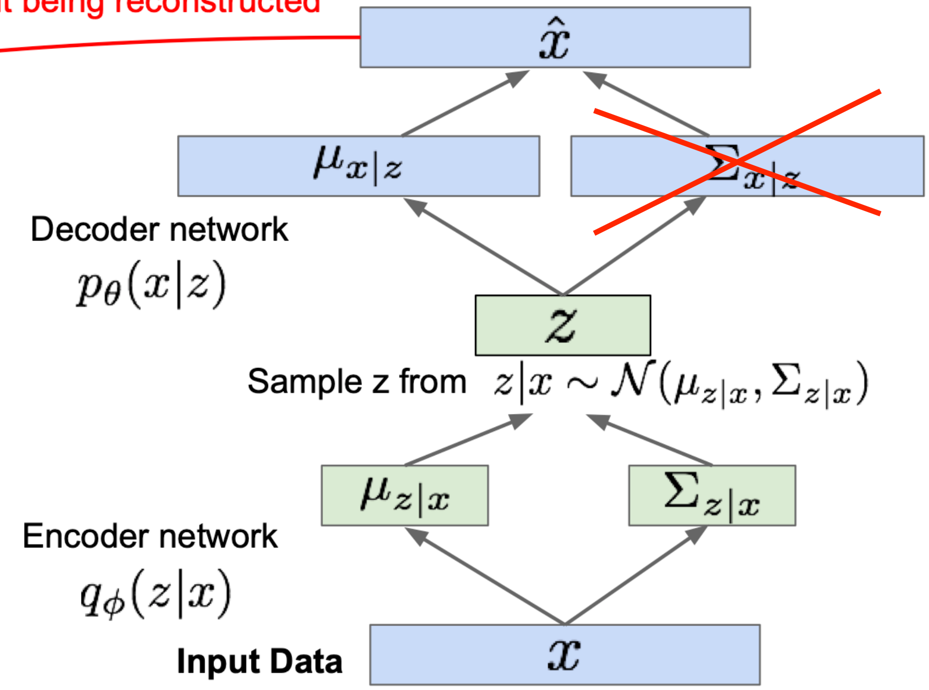
Note that this term is still intractable, which we need to use Monte Carlo estimation, which simply removes E_z.

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\boxed{\log p_\theta(x^{(i)} \mid z)} - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Maximize likelihood of original input being reconstructed



In practice, some implementations simply set $\Sigma_{x|z} = I$, then this reconstruction term becomes MSE loss:

$$||x^{(i)} - \mu_{x|z}||^2$$

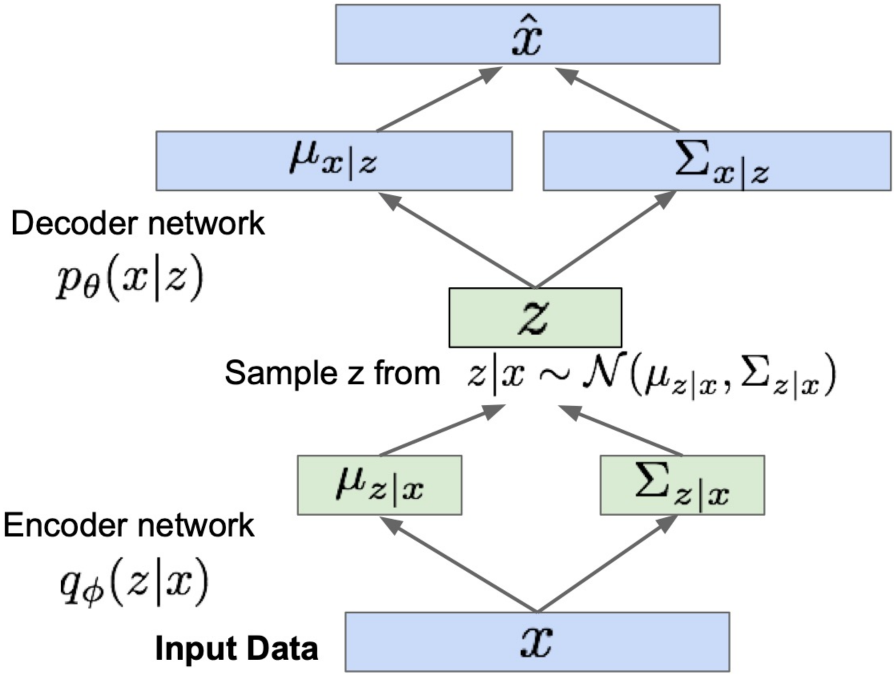
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\boxed{\underbrace{\log p_\theta(x^{(i)} \mid z) \;\; - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}}$$

For every minibatch of input data: compute this forward pass, and then backprop!

Why called variational?



Variation and Functional

- Functional: mappings from a set of functions to the real numbers, where the independent variable is a function.

- Variations δ: small changes in functions and functionals, to find maxima and minima (collectively called extrema) of functionals.

Variation and Functional

- Functional: mappings from a set of functions to the real numbers, where the independent variable is a function.
- Variations δ: small changes in functions and functionals, to find maxima and minima (collectively called extrema) of functionals.
- Example: Find the shortest curve to connect two points, A and B, in a 2D plane.
  - Independent variable: the function of the curve $f(x, y)$
  - Functional l: $f(x, y) \to \mathbb{R}$ (length $l = \int_A^B f(x, y) ds$)

Why Called Variational?

- Training VAE can be seen as solving a **variational problem** (to obtain the extrema of the functional):
  - $ELBO$ is a functional of $q(z|x)$ and $p(x|z)$.
  - The variational problem to solve: $\hat{p}, \hat{q} = argmax_{p,q} ELBO$
  - Solving this particular problem is called **variational inference**.
  - This is a kind of approximate inference, since it can't give your the true data log probability $\log p(x)$.
  - Instead, it gives your the lower bound of $\log p(x)$, that is ELBO.

Variational Autoencoders

- In reality, we use θ, ϕ to parameterize p, q:
  - The set of functions (or called **variational family**): $q_\phi(z|x), p_\theta(z|x)$
  - Problem to solve: $\hat{\phi}, \hat{\theta} = argmax_{\phi,\theta} ELBO$
    - Becomes a known problem: to obtain the maximum of a function (not a functional anymore!)
    - We can use gradient descent on ϕ and θ.

Variational Inference

- From the perspective of classic statistical learning, this approximate inference is done via solving a variational problem (another mainstream method is MCMC sampling), so it is called variational.

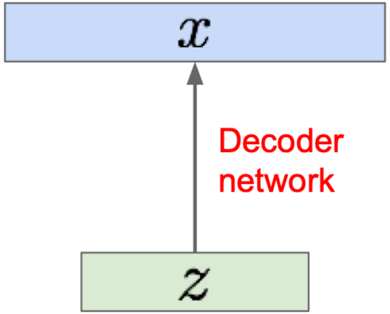
- From a modern perspective, training VAE is essentially the same with training other neural networks.
    - All neural network trainings can be seen as solving variational problems!
    - Functional: your loss function
    - Independent variable: your neural network function
    - However nobody calls it in this way any more

Variational Autoencoders: Generating Data

Our assumption about data generation process

Sample from true conditional

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



Sample from true prior

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

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data

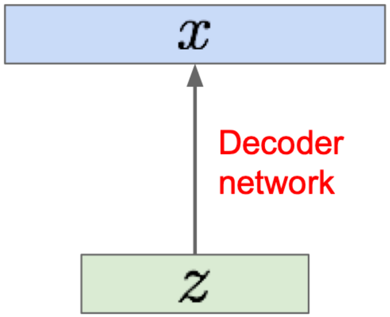
Our assumption about data generation process

Sample from true conditional

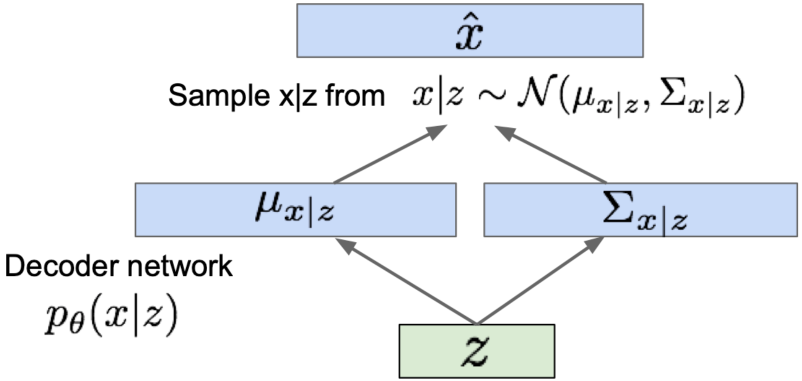
$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

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



Now given a trained VAE:
use decoder network & sample z from prior!

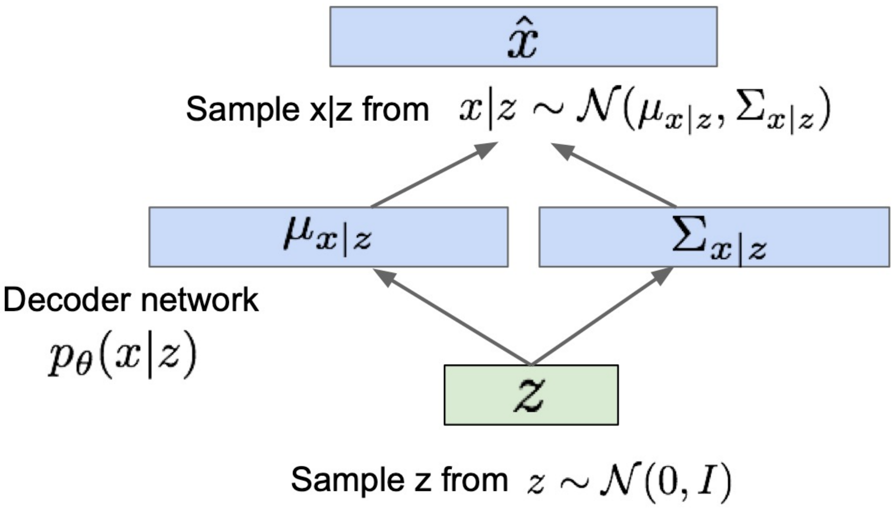


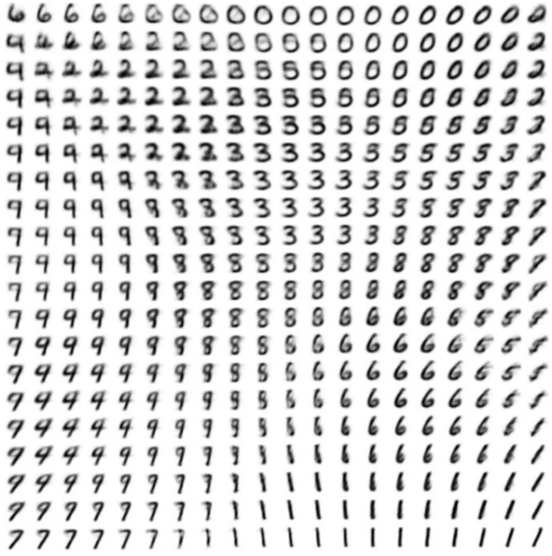
Sample z from $z \sim \mathcal{N}(0, I)$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data

Use decoder network. Now sample z from prior!

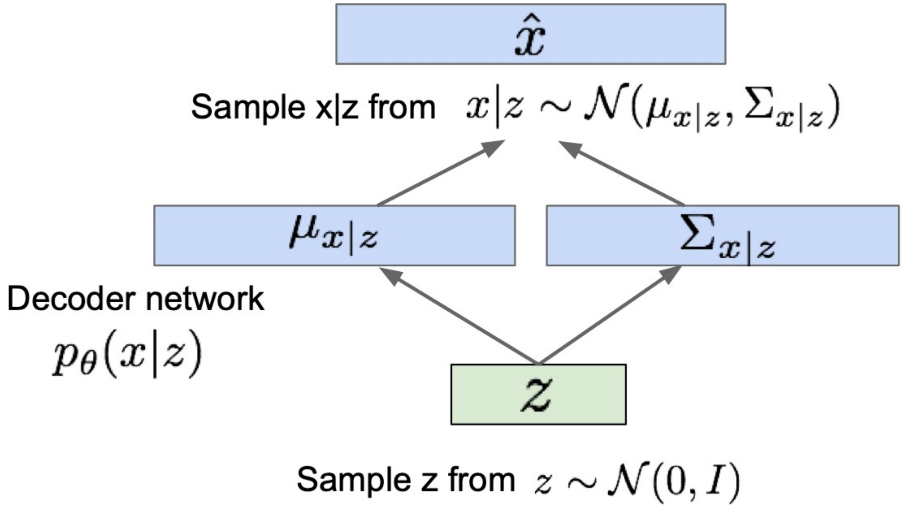




Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

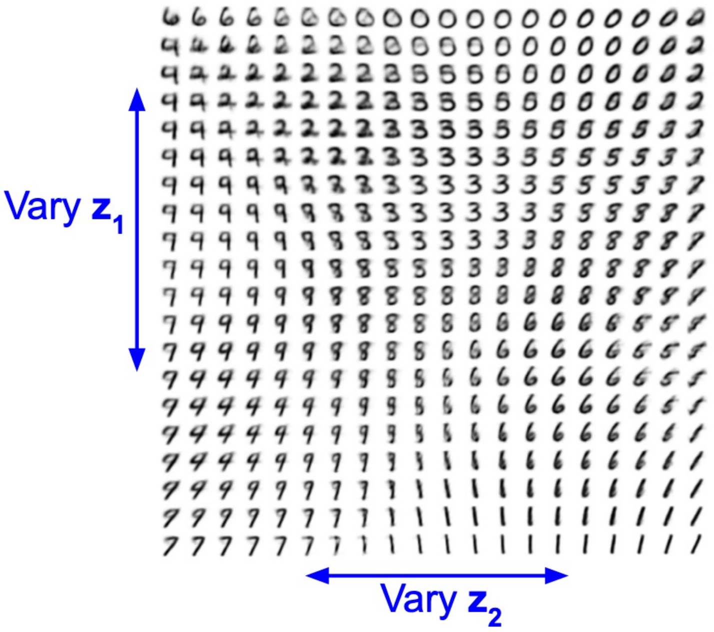
Variational Autoencoders: Generating Data

Use decoder network. Now sample z from prior!



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Data manifold for 2-d **z**



Variational Autoencoders: Generating Data

Diagonal prior on **z**
=> independent
latent variables

Different
dimensions of **z**
encode
interpretable factors
of variation

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data

Diagonal prior on **z**
=> independent
latent variables

Different
dimensions of **z**
encode
interpretable factors
of variation

Also good feature representation that can be computed using $q_\phi(z|x)$!

Degree of smile

Vary $\mathbf{z}_1$

Vary $\mathbf{z}_2$

Head pose

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data

32x32 CIFAR-10

Labeled Faces in the Wild

Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data
Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Interpretable latent space.
- Allows inference of $q(z|x)$, can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.

Generative Adversarial Networks (GAN)

Some slides are borrowed from Stanford CS231N.

Taxonomy of Generative Models

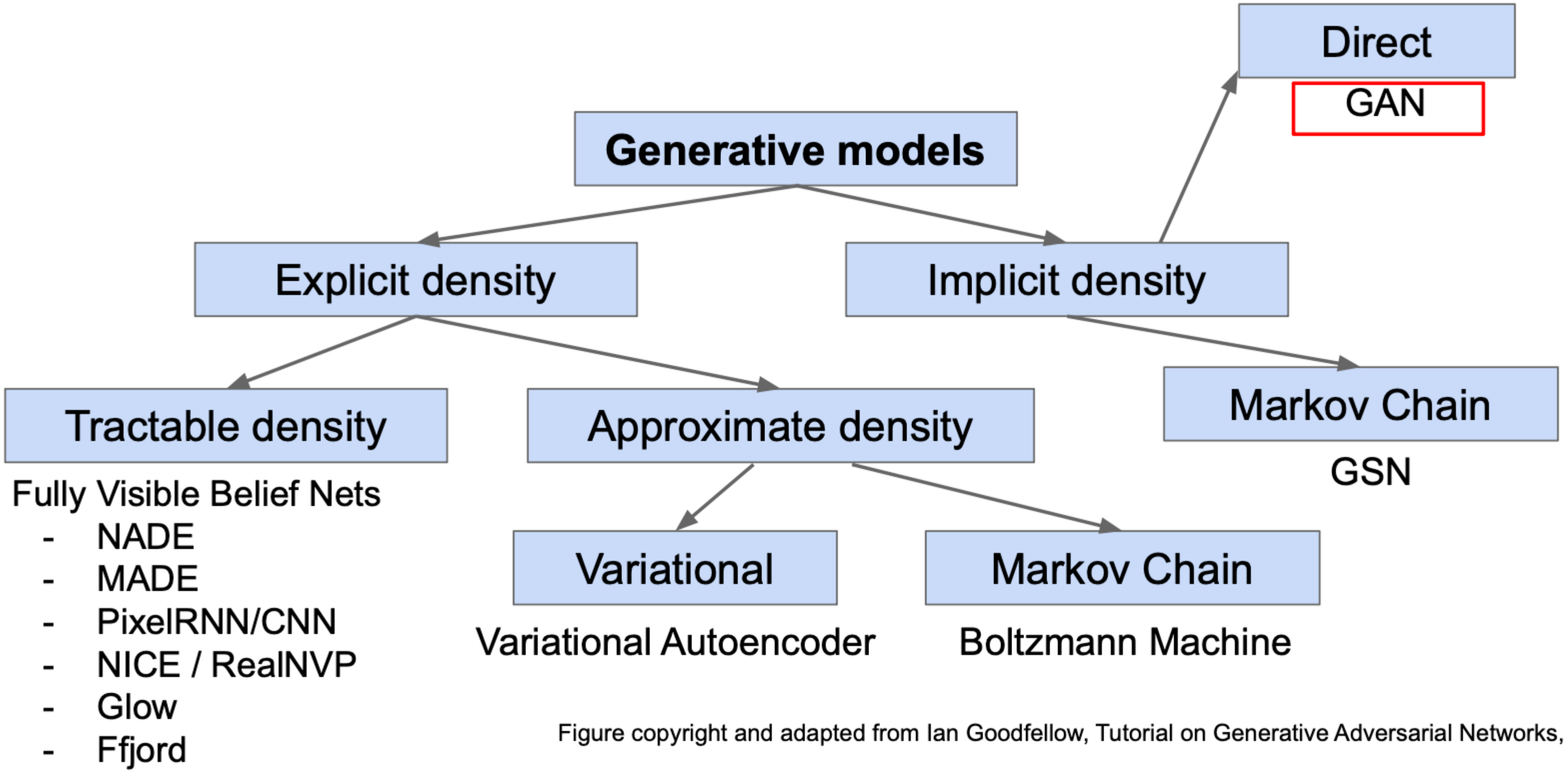


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Motivation

So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

VAEs define intractable density function with latent **z**:

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

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: not modeling any explicit density function!

Generative Adversarial Networks

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

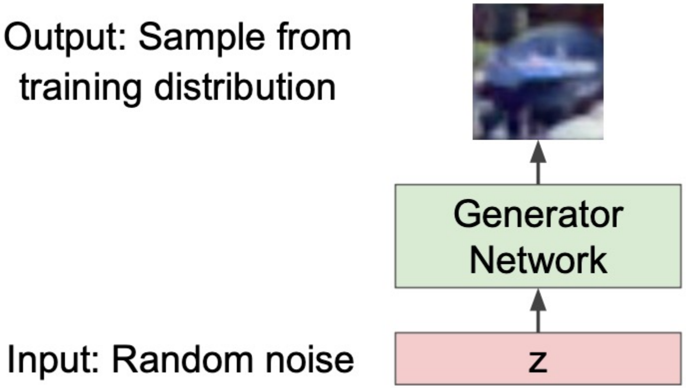
Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Generative Adversarial Networks

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images



What loss function should we use?

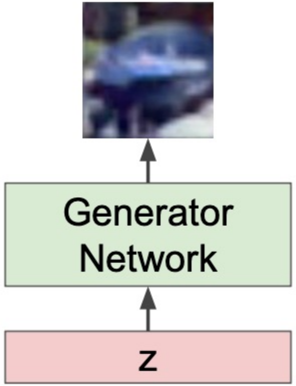
Generative Adversarial Networks

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution



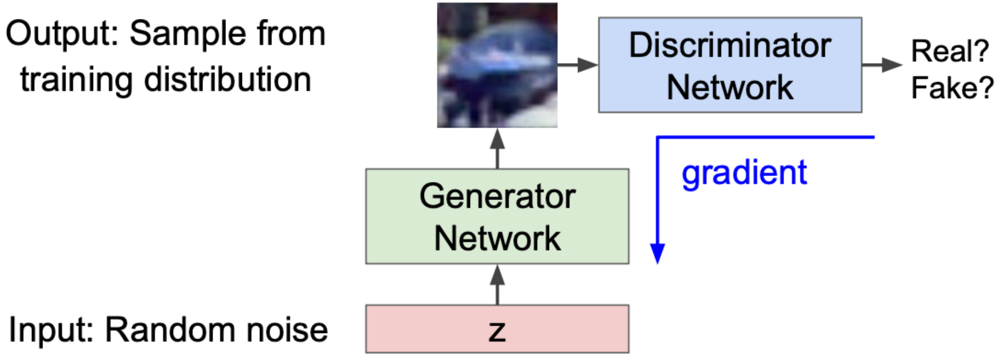
Input: Random noise

Objective: generated images should look "real"

Generative Adversarial Networks

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

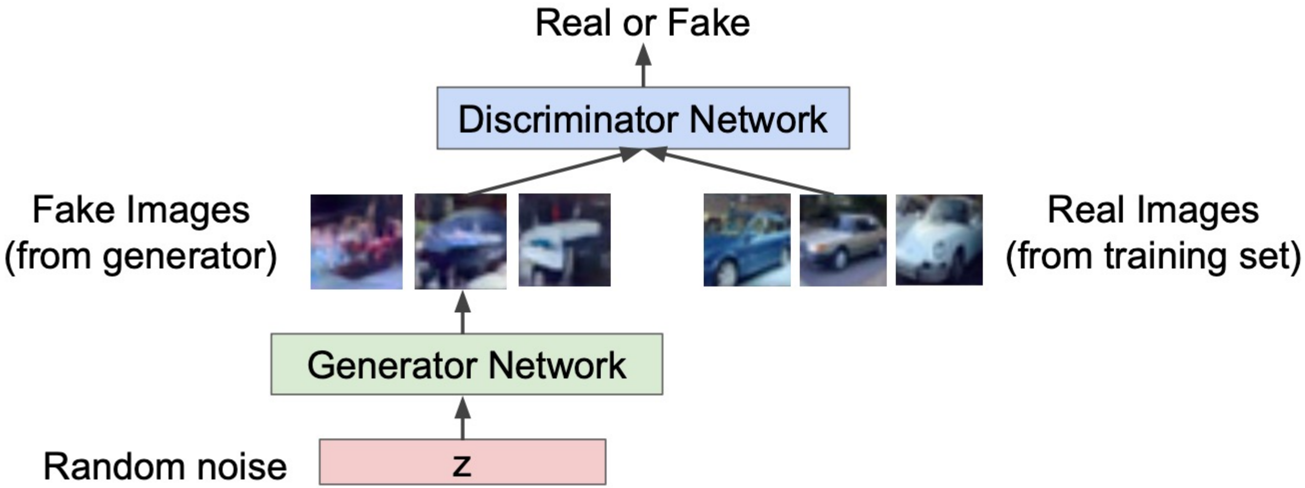
Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.



Solution: Use a discriminator network to tell whether the generate image is within data distribution ("real") or not

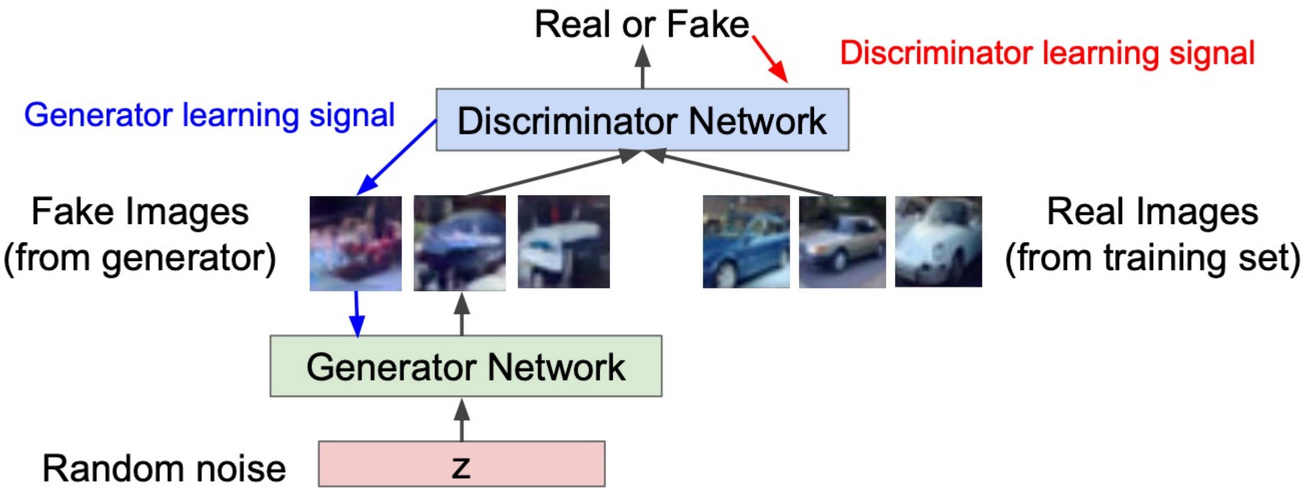
Training GANs: Two-Player Games

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



Training GANs: Two-Player Games

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



Training GANs: Two-Player Games

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

Train jointly in **minimax game**

Minimax objective function:

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

Generator objective

Discriminator objective

Training GANs: Two-Player Games

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

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

Training GANs: Two-Player Games

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

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

Training GANs: Two-Player Games

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

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- 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 Games

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

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

Training GANs: Two-Player Games

Minimax objective function:

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

Alternate between:

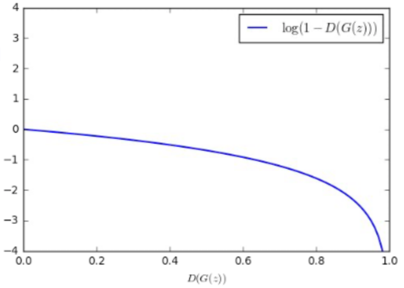
1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

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

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).



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

Training GANs: Two-Player Games

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

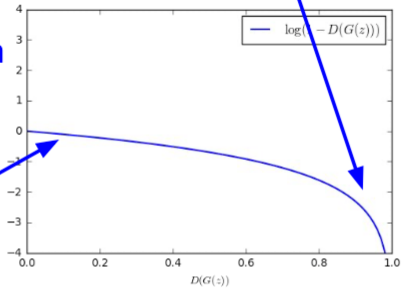
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

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

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



Training GANs: Two-Player Games

Minimax objective function:

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

Alternate between:

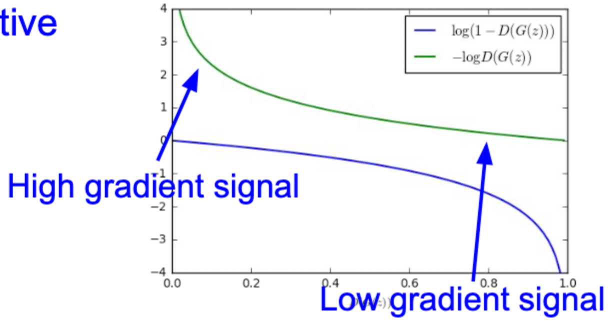
1. **Gradient ascent** on discriminator

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

2. **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.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Training GANs: Two-Player Games

Putting it together: GAN training algorithm

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:

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

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

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

end for

Training GANs: Two-Player Games

Putting it together: GAN training algorithm

Some find k=1 more stable, others use k > 1, no best rule.

Followup work (e.g. Wasserstein GAN, BEGAN) alleviates this problem, better stability!

for number of training iterations **do**

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

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

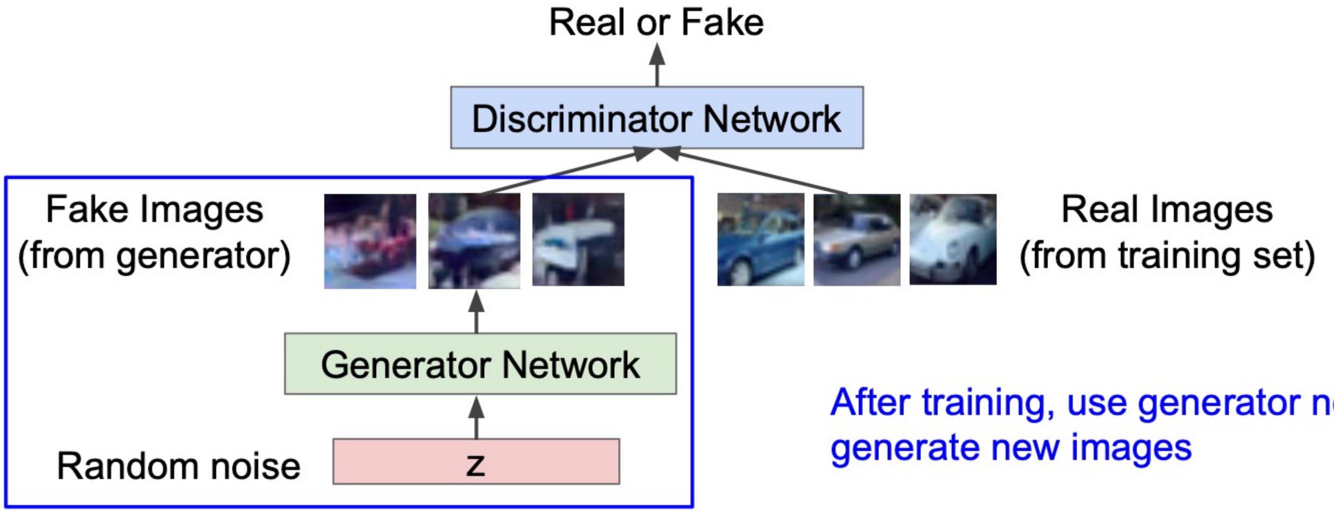
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)
Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

Training GANs: Two-Player Games

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



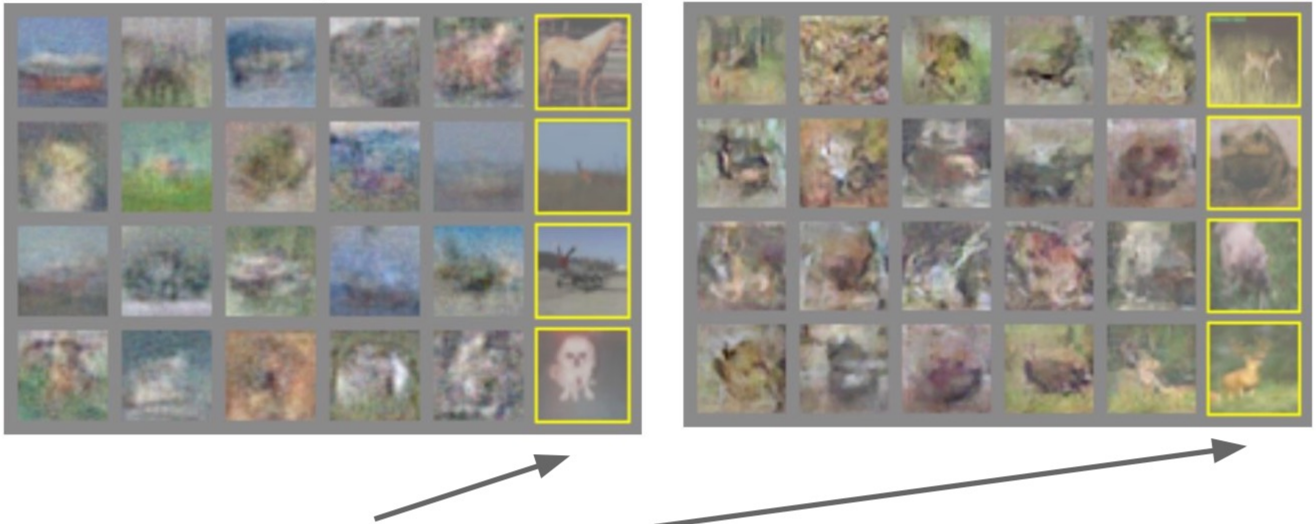
After training, use generator network to generate new images

Generated Samples by GAN

Nearest neighbor from training set

Generated Samples by GAN

Generated samples (CIFAR-10)



Nearest neighbor from training set

GAN: Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

Samples from Convolutional GAN

Samples from the model look much better!

Radford et al,
ICLR 2016

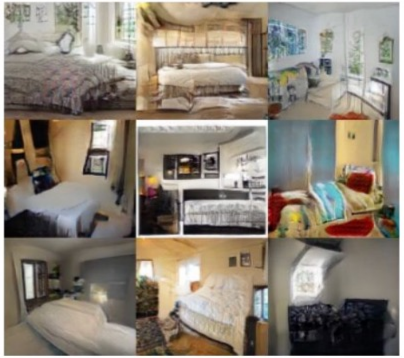
Samples from Convolutional GAN

Interpolating between random points in laten space

Radford et al, ICLR 2016

2017: Explosion of GANs

Better training and generation



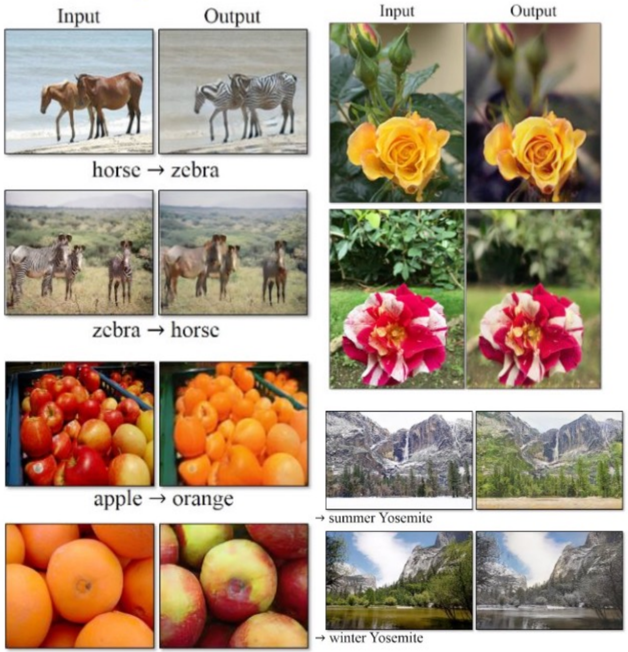
LSGAN, Zhu 2017.

Wasserstein GAN, Arjovsky 2017. Improved Wasserstein GAN, Gulrajani 2017.

Progressive GAN, Karras 2018.

2017: Explosion of GANs

Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis

Reed et al. 2017.

Many GAN applications

Pix2pix. Isola 2017. Many examples at https://phillipi.github.io/pix2pix/

2019: BigGAN

Brock et al., 2019

StyleGAN Series

StyleGAN v1

StyleGAN v2

StyleGAN v3

Evaluation Metric

- There is no objective function used when training GAN generator models, meaning models must be evaluated using the quality of the generated synthetic images.

- Manual inspection of generated images is a good starting point when getting started.

- **Quantitative measures**, such as the inception score and the Frechet inception distance, can be combined with **qualitative assessment** to

Qualitative GAN Generator Evaluation

1. **Nearest neighbors**: to detect overfitting, generated samples are shown next to their nearest neighbors in the training set

2. **User study**: in these experiments, participants are asked to distinguish generated samples from real images in a short presentation time (e.g. 100 ms), i.e. real v.s fake; or, participants are asked to rank models in terms of the fidelity of their generated images

3. **Mode drop and mode collapse**: Over datasets with known modes (e.g. a GMM or a labeled dataset), modes are computed as by measuring the distances of generated data to mode centers

Quantitative Measurement: FID

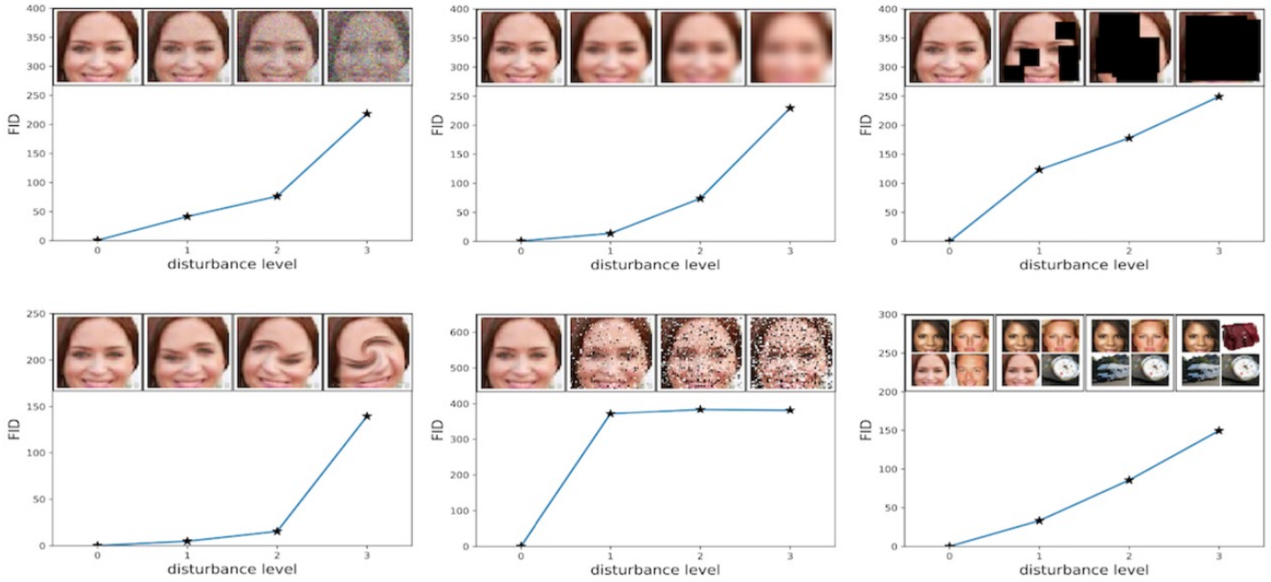
Fréchet Inception Distance (FID)

- FID embeds a set of generated samples into a feature space given by a specific layer of Inception Net (or any CNN).
- Viewing the embedding layer as a continuous multivariate Gaussian, the mean and covariance are estimated for both the generated data and the real data.
- The Fréchet distance between these two Gaussians (a.k.a Wasserstein-2 distance) is then used to quantify the quality of generated samples

$$FID(r,g) = ||\mu_r - \mu_g||_2^2 + Tr\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}\right)$$

Lower FID means smaller distances between synthetic and real data distributions.

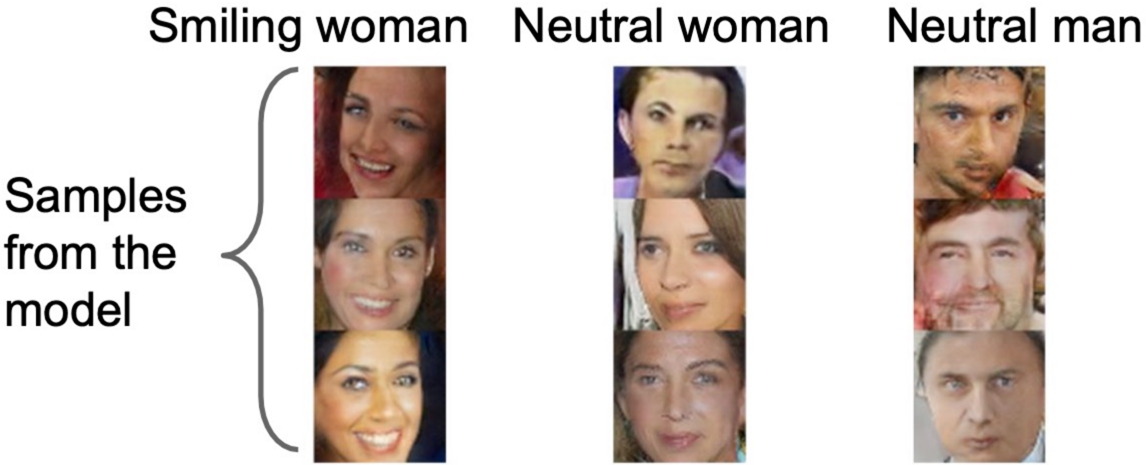
FID Measurement



FID measure is sensitive to image distortions. From upper left to lower right: Gaussian noise, Gaussian blur, implanted black rectangles, swirled images, salt and pepper noise, and CelebA dataset contaminated by ImageNet images.

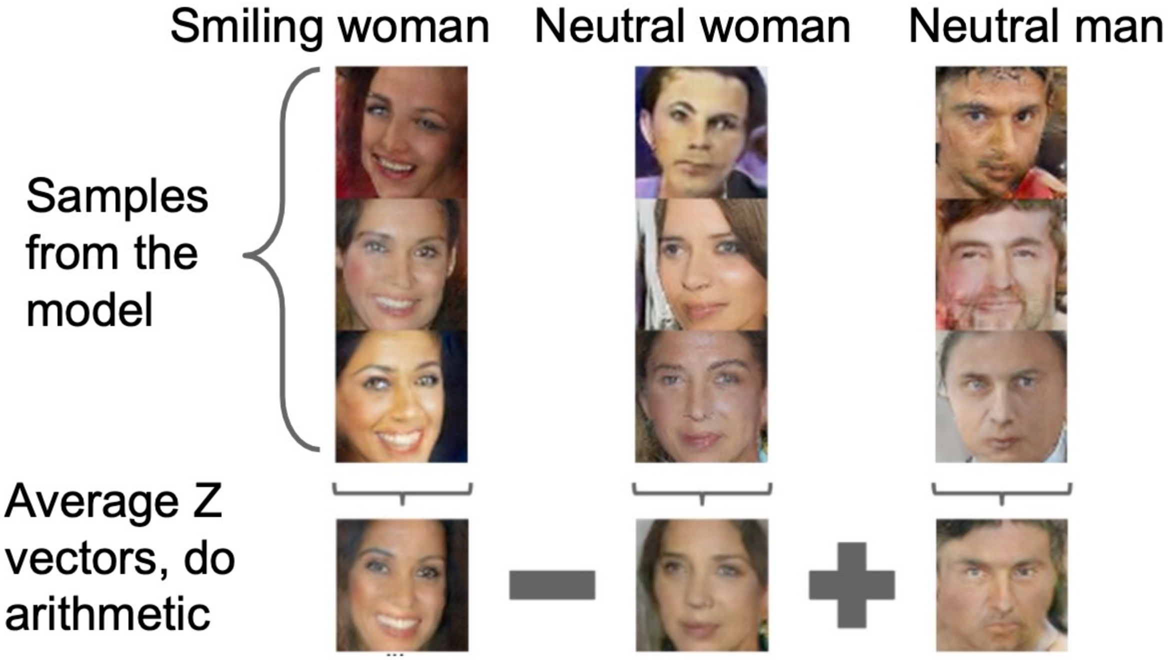
GAN: Interpretable Vector Math

Radford et al, ICLR 2016



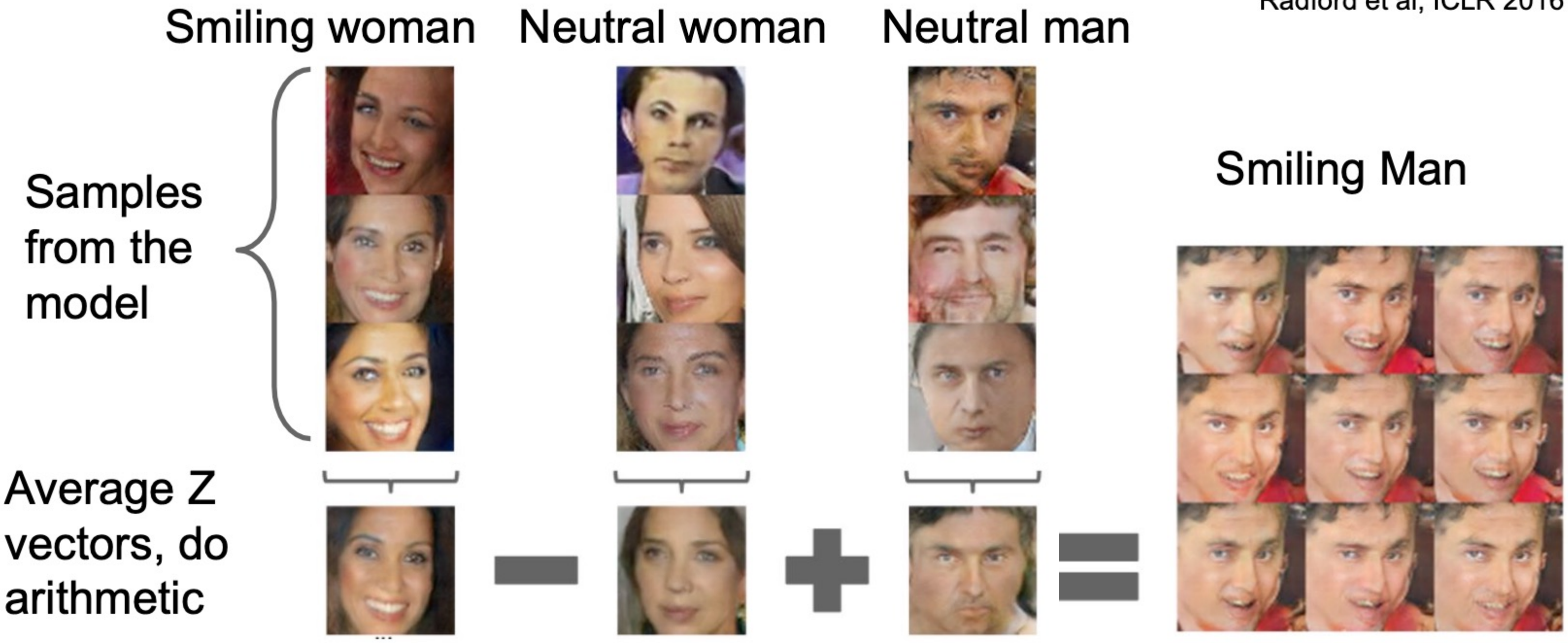
GAN: Interpretable Vector Math

Radford et al, ICLR 2016



GAN: Interpretable Vector Math

Radford et al, ICLR 2016



GAN: Interpretable Vector Math

Radford et al, ICLR 2016

The GAN Zoo

- 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

https://github.com/hindupuravinash/the-gan-zoo

Resources

My recommended reading list:

- WGAN
- WGAN-gp
- GAN landscape
- Progressive Growing GAN
- StyleGAN

Courses:

Stanford CS236: Deep Generative Models

Summary: GAN

Don't 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)

Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

VAE vs. GAN

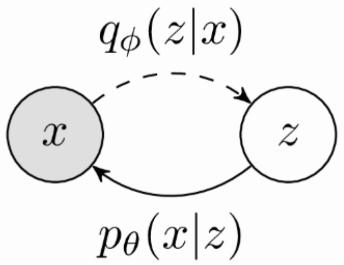
- VAE
  - Blurry
  - Full coverage of the data
  - Support approximate inference

- GAN
  - More realistic
  - Only penalize fake and therefore can suffer from mode collapse
  - Can’t infer probability

Diffusion Model

Variational Autoencoders

Figure 1 - Graphical Model for VAE



- Generative model and goal:

$$p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z) \qquad \theta^* = \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{x\sim\hat{p}(x), z\sim p(z)} [\log p_\theta(x|z;\theta)]$$

x: data; z: latent variable

- VAE solves it by sampling z from a new distribution $q_\phi(z|x)$

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

$$\log p(x) = \log \mathbb{E}_{z\sim q_\phi(z|x)} \left[\frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} \right]$$

$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} \right]$$

The right-hand side is the evidence lower-bound(ELBO) $\mathcal{L}(\theta, \phi)$

Hierarchical Variational Autoencoders

- VAE with two latent variables, consider joint distribution $p(x, z_1, z_2)$

$$p(x) = \int_{z_1} \int_{z_2} p_\theta(x, z_1, z_2) dz_1, dz_2$$

- Introduce a variational approximation to the true posterior and get the ELBO:

$$p(x) = \iint q_\phi(z_1, z_2|x) \frac{p_\theta(x, z_1, z_2)}{q_\phi(z_1, z_2|x)}$$

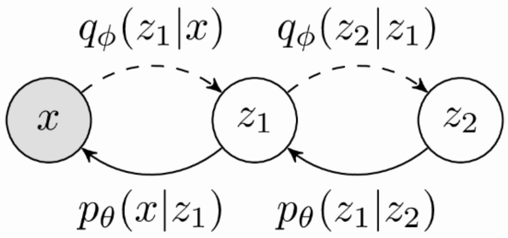
$$p(x) = \mathbb{E}_{z_1, z_2 \sim q_\phi(z_1, z_2|x)} \left[\frac{p_\theta(x, z_1, z_2)}{q_\phi(z_1, z_2|x)} \right]$$

$$\log p(x) \geq \mathbb{E}_{z_1, z_2 \sim q_\phi(z_1, z_2|x)} \left[\log \frac{p_\theta(x, z_1, z_2)}{q_\phi(z_1, z_2|x)} \right]$$

We are free to factorize the inference and generative models as we see fit

Hierarchical Variational Autoencoders

Figure 2 - A Hierarchical VAE



- Consider following model:

$$p(x, z_1, z_2) = p(x|z_1)p(z_1|z_2)p(z_2)$$
$$q(z_1, z_2|x) = q(z_1|x)q(z_2|z_1)$$

- Substituting these factorizations into the ELBO, we get

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{q(z_1,z_2|x)}\left[\log p(x|z_1) - \log q(z_1|x) + \log p(z_1|z_2) - \log q(z_2|z_1) + \log p(z_2)\right]$$

This can be alternatively written as a “reconstruction term”, and tl … d its corresponding prior:

$$= \mathbb{E}_{q(z_1|z_2)}\left[\log p(x|z_1)\right] - D_{KL}\left(q(z_1|x) \parallel p(z_1|x)\right) - D_{KL}\left(q(z_2|z_1) \parallel p(z_2)\right)$$

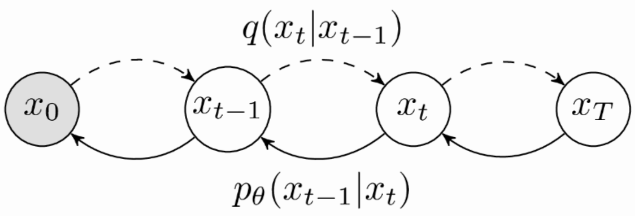
Diffusion Probabilistic Models

- Consider the following model with a sequence of ***T*** variables:

  $x_0 \sim p(x)$: observed data

  $x_{1:T}$: latent variables

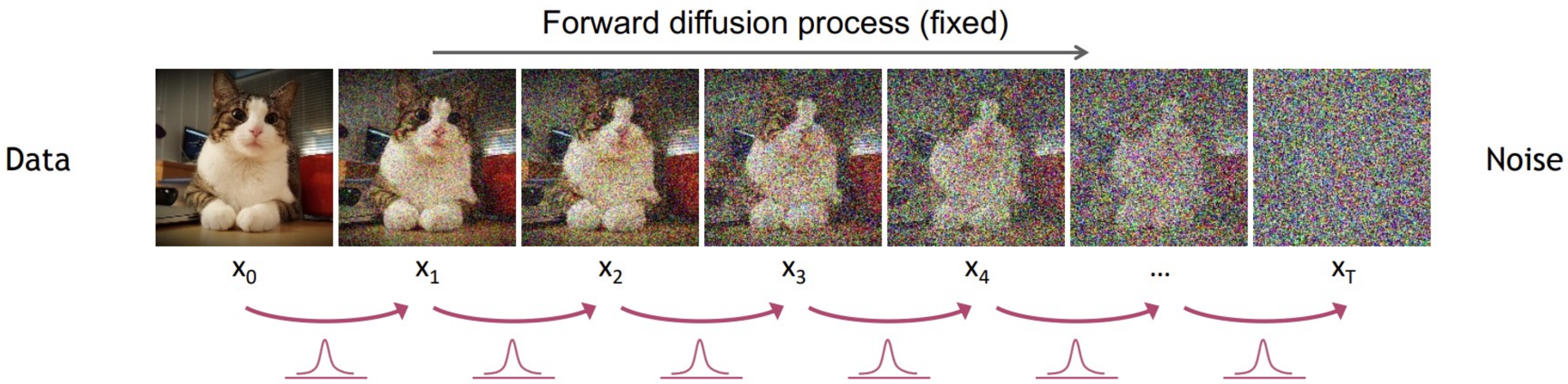
Figure 3 - Diffusion Probabilistic Model



In fact, we can think of diffusion models as a specific realization of a hierarchical VAE. What sets them apart is a unique inference model, which contains **no learnable parameters** and is constructed so that the final latent distribution $q(x_T)$ converges to a standard gaussian

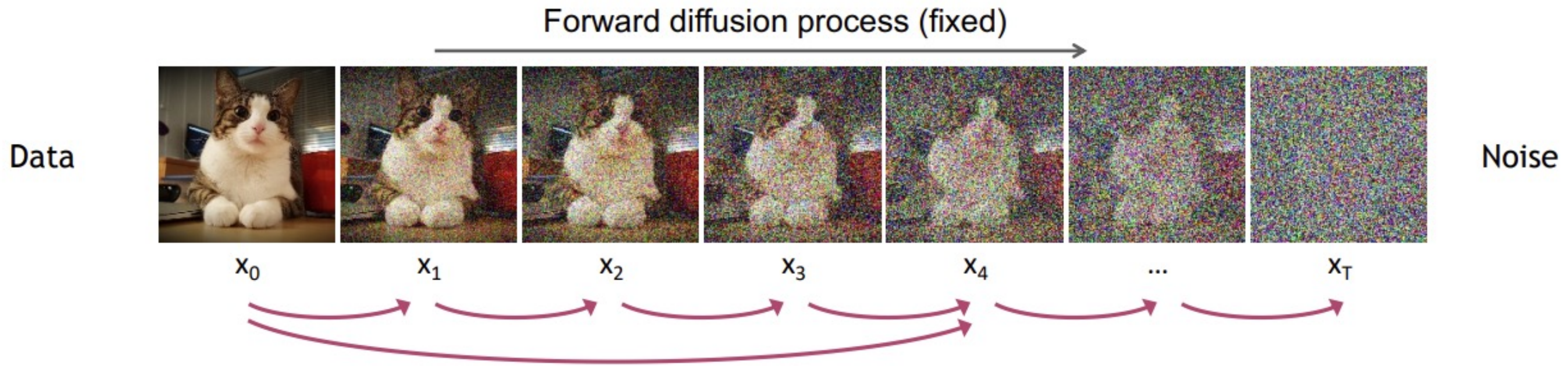
$$q(x_t|x_{t-1}) = \mathcal{N}(x_T; x_{t-1}\sqrt{1-\beta_t}, \beta_t I)$$

Forward Diffusion Process



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

Forward Diffusion Process — Diffusion Kernel



Define: $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ we have: $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$ (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 $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

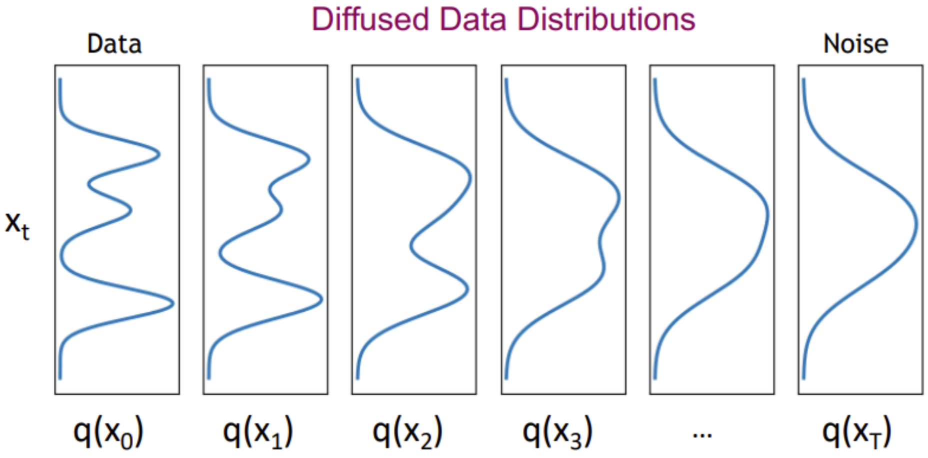
Forward Diffusion Process

What happens to a distribution in the forward diffusion?

So far, we discussed the diffusion kernel $q(\mathbf{x}_t|\mathbf{x}_0)$ but what about $q(\mathbf{x}_t)$?

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

The diffusion kernel is Gaussian convolution.



We can sample $\mathbf{x}_t \sim q(\mathbf{x}_t)$ by first sampling $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ and then sampling $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$ (i.e., ancestral sampling).

Generative Learning by Denoising

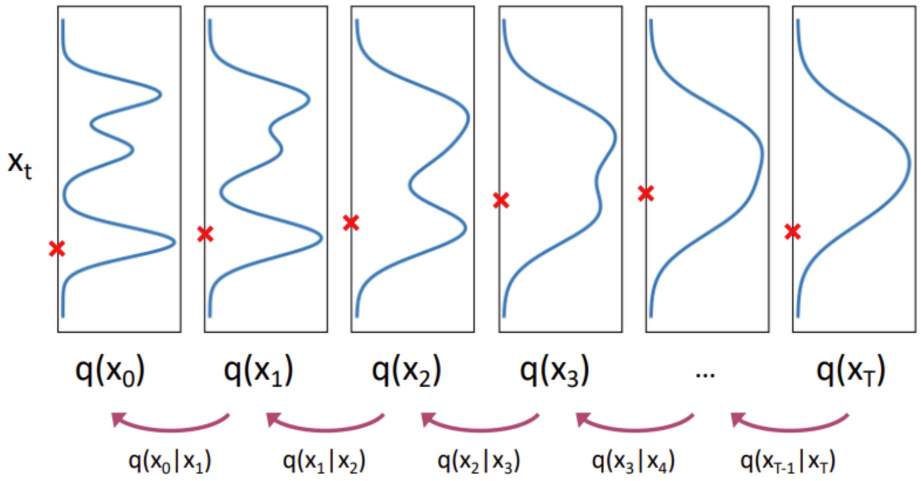
Recall, that the diffusion parameters are designed such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Generation:

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

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

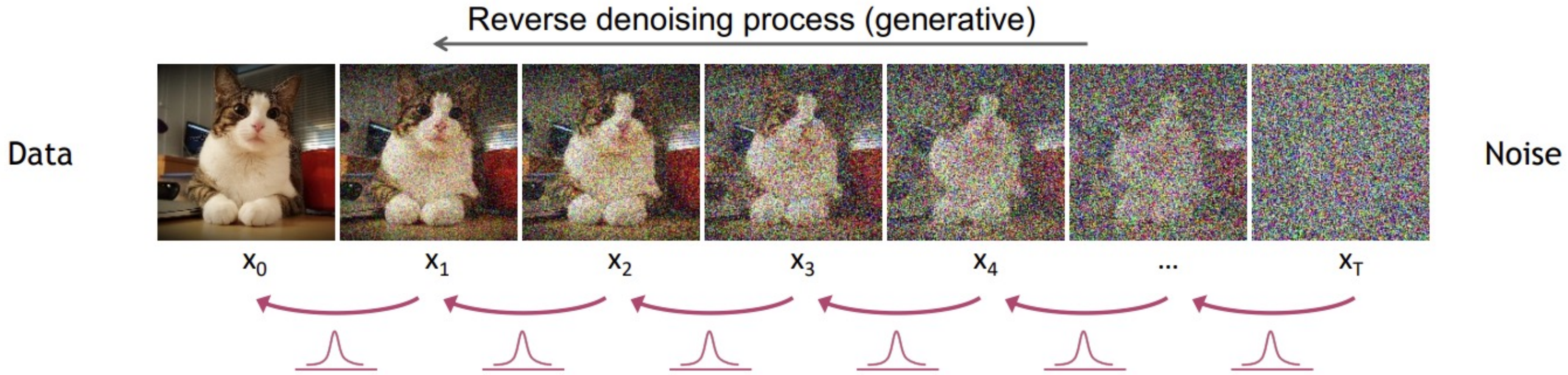
Diffused Data Distributions



Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$? Yes, we can use a Normal distribution if β_t is small in each forward diffusion step.

Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

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

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

Learning Denoising Model — Variational upper bound

For training, we can form variational upper bound that is commonly used for training variational autoencoders:

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

Recall that $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$. Ho et al. NeurIPS 2020 parameterized the mean of denoising model via:

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

Using a few simple arithmetic operations, we can write down the variational objective as:

$$L = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), t \sim \mathcal{U}\{1,T\}, \epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})}\left[\lambda_t ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon, t)||^2\right]$$

Ho et al. NeurIPS 2020 observe that simply setting λ_t to 1 for all *t* works best in practice.

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in neural information processing systems, 33, 6840-6851.

Training and Sample Generation

Algorithm 1 Training

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

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \ldots, 1$ **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 $\mathbf{x}_0$

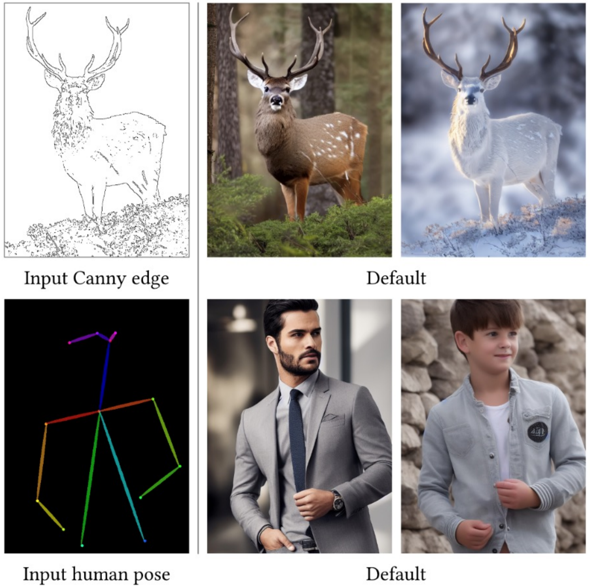
Conditional Diffusion Model

Conditional generation is of great importance!

Text-to-image

a teddy bear on a skateboard in times square

Spatial control



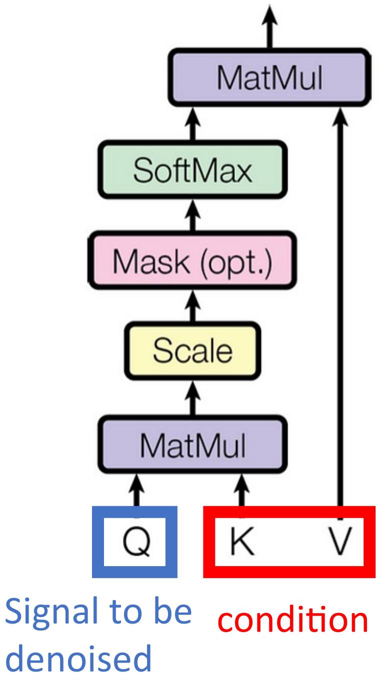
Conditional Diffusion Model

Almost the same as unconditional diffusion!

	Unconditional	Conditional
Forward	$q(x_1, \dots, x_T \| x_0) := \prod_{t=1} q(x_t \| x_{t-1})$	$q(x_1, \dots, x_T \| x_0, y) := \prod_{t=1} q(x_t \| x_{t-1}, y)$
Model	$\epsilon_\theta(x_t, t)$	$\epsilon_\theta(x_t, y, t)$
Loss	$\mathbb{E}_{t \sim U(0,T), x_t \sim p(x_t)}[\| \epsilon - \epsilon_\theta(x_t, t) \|^2]$	$\mathbb{E}_{x_t, y \sim p(x_t, y), t \sim U(0,T)}[\| \epsilon - \epsilon_\theta(x_t, y, t) \|^2]$

Condition Schemas

1. **Cross attention**

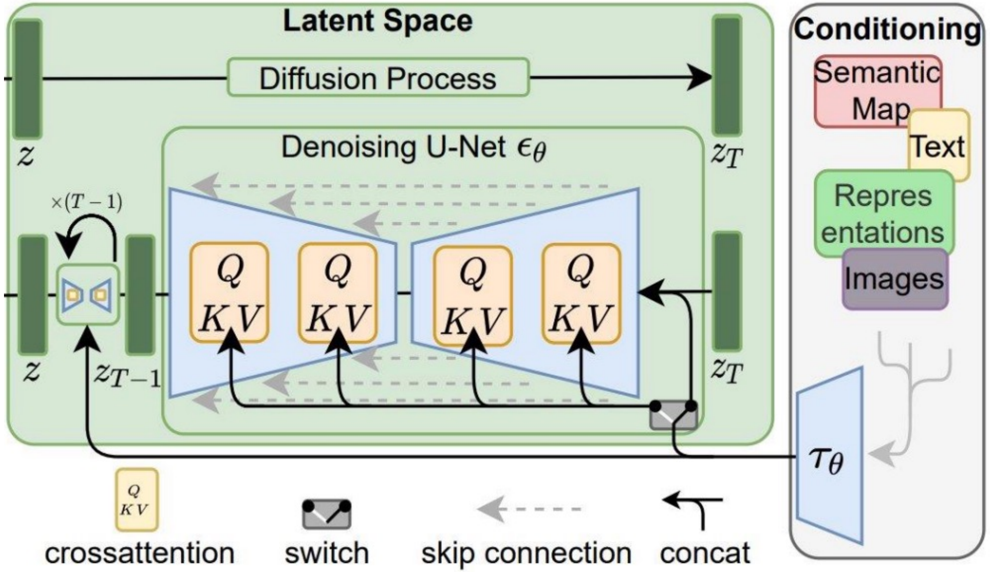


$$Q = W_Q \cdot \varphi_i(x_t), K = W_K \cdot \tau_\theta(y),\ V = W_V \cdot \tau_\theta(y)$$

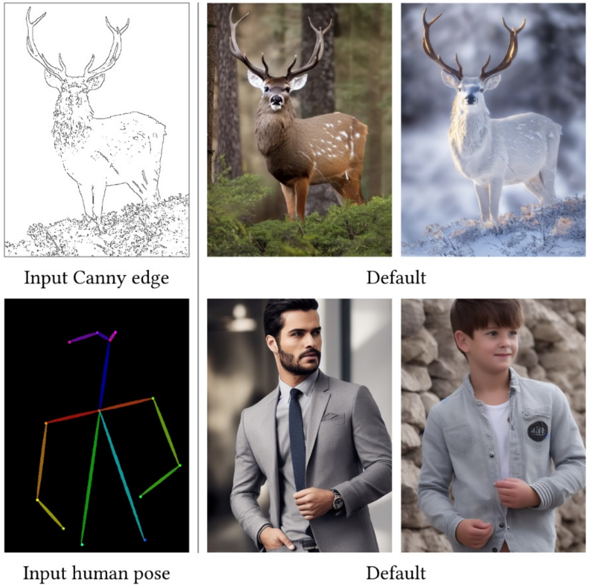
- y is the condition signal (e.g. text)
- τ_θ is a domain specific encoder (e.g. text en
- $\varphi_i(x_t)$ is the signal to be denoised

Condition Schemas

1. **Cross attention**



Stable Diffusion



ControlNet

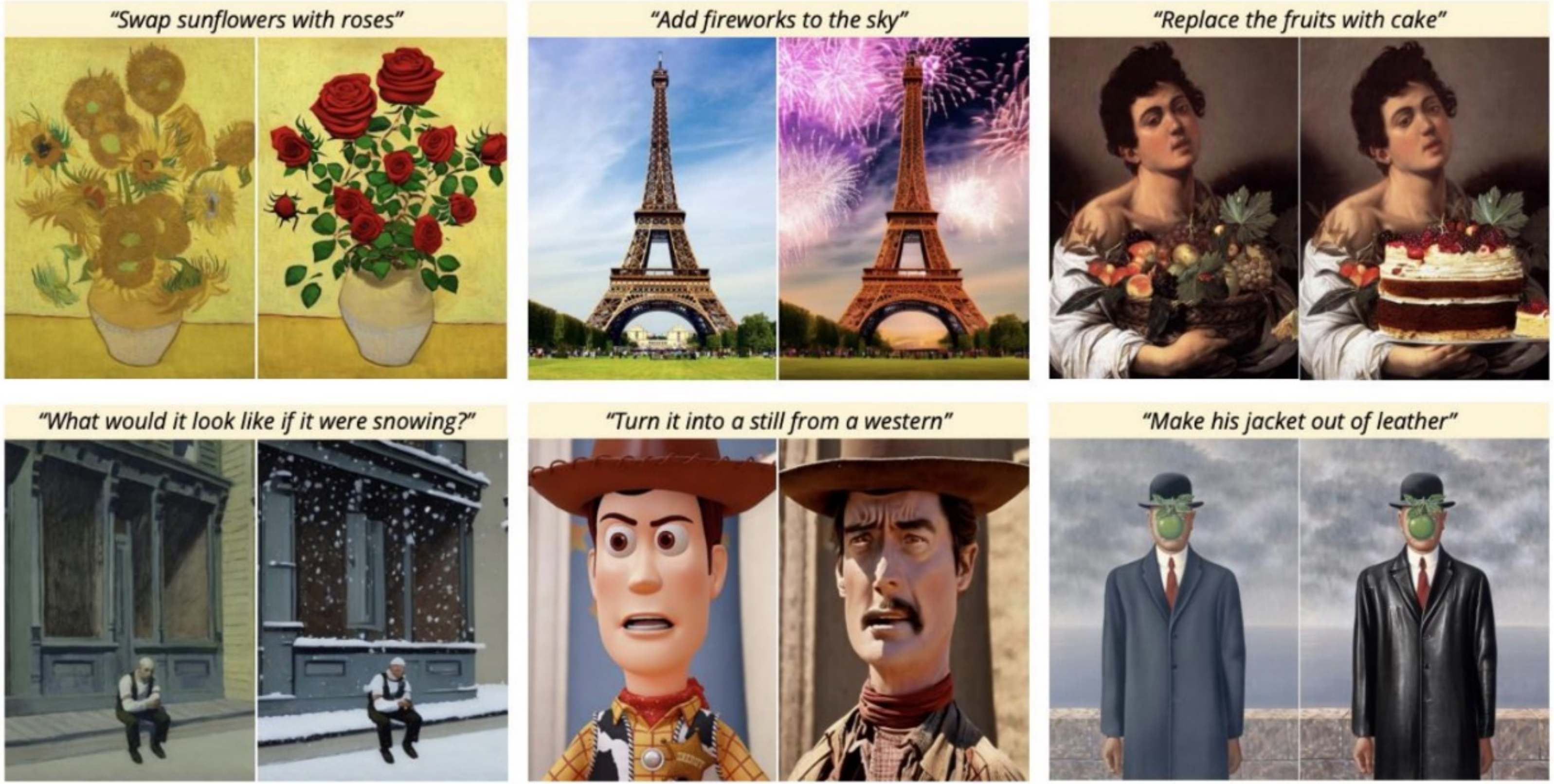
Condition Schemas

1. Cross attention
2. Simple concatenation (image to image tasks)
3. Conditional normalization
4. …

More applications: Text-to-Video

A stylish woman walks down a Tokyo street filled with warm glowing neon and animated city signage. She wears a black leather jacket, a long red dress, and black boots, and carries a black purse. She wears sunglasses and red lipstick. She walks confidently and casually. The street is damp and reflective, creating a mirror effect of the colorful lights. Many pedestrians walk about.

More applications: Image Editing



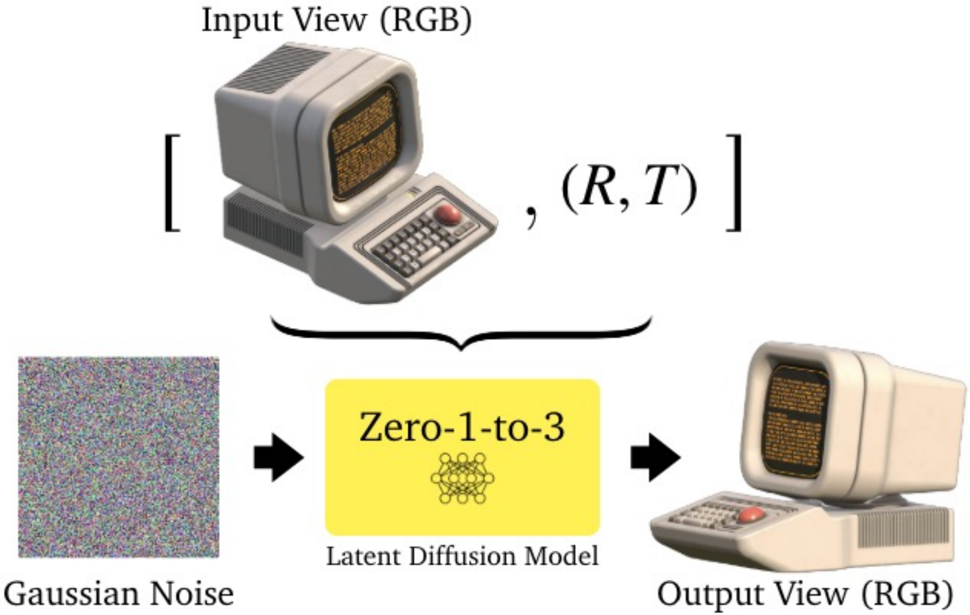
More applications: Personalization

Generated images

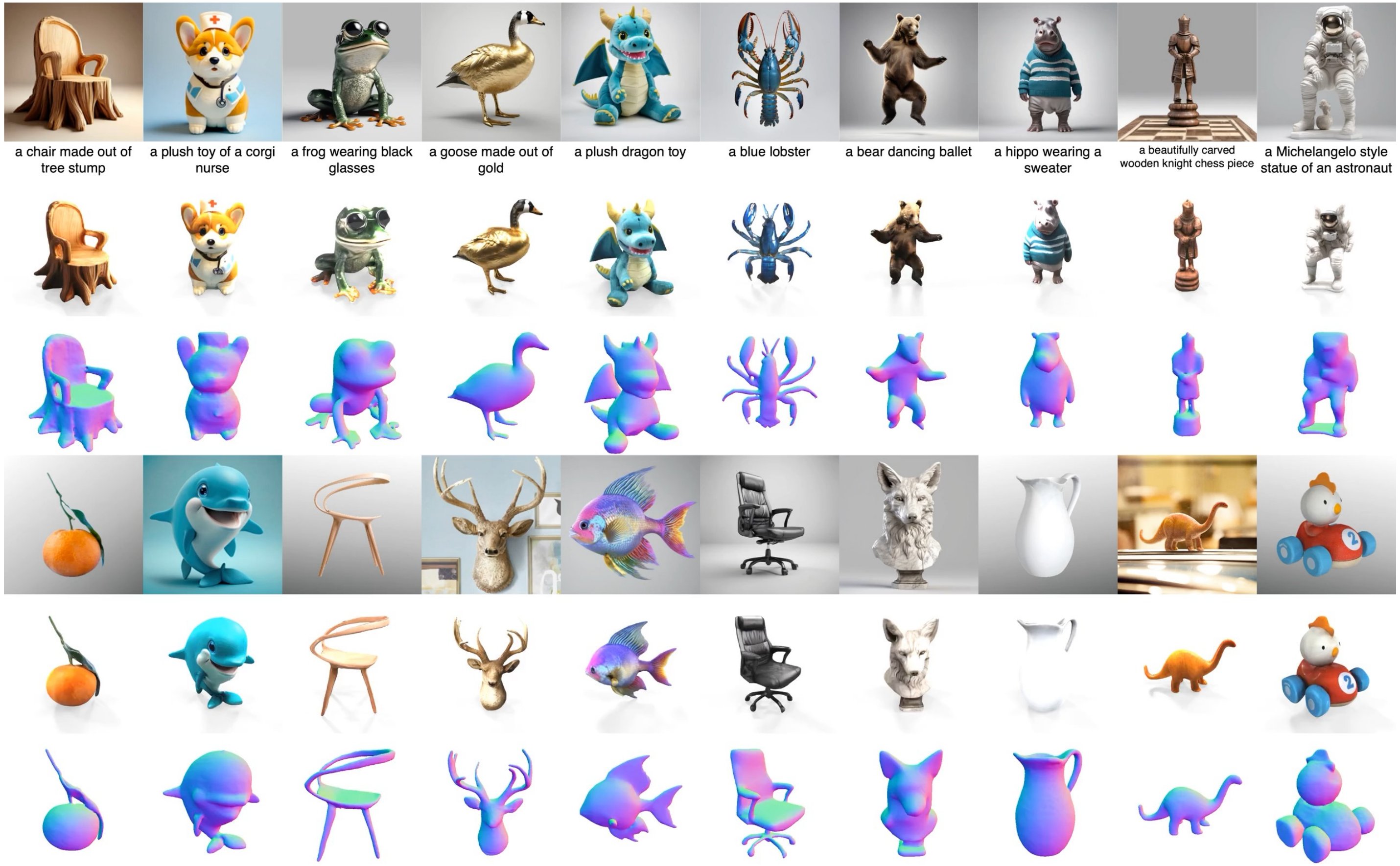
Ruiz et al., "DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation", CVPR 2023

More applications: Text-to-3D

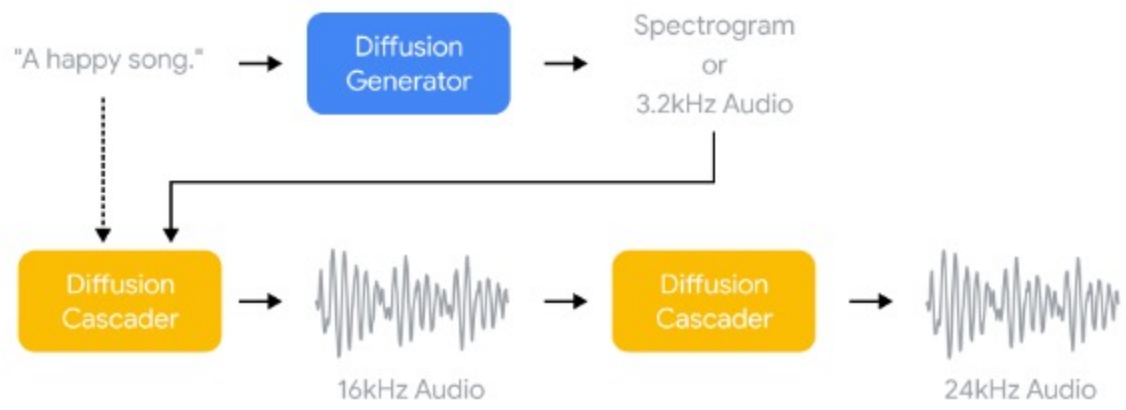
Finetune stable diffusion to **stimulate its awareness of camera poses**



More applications: Text-to-3D



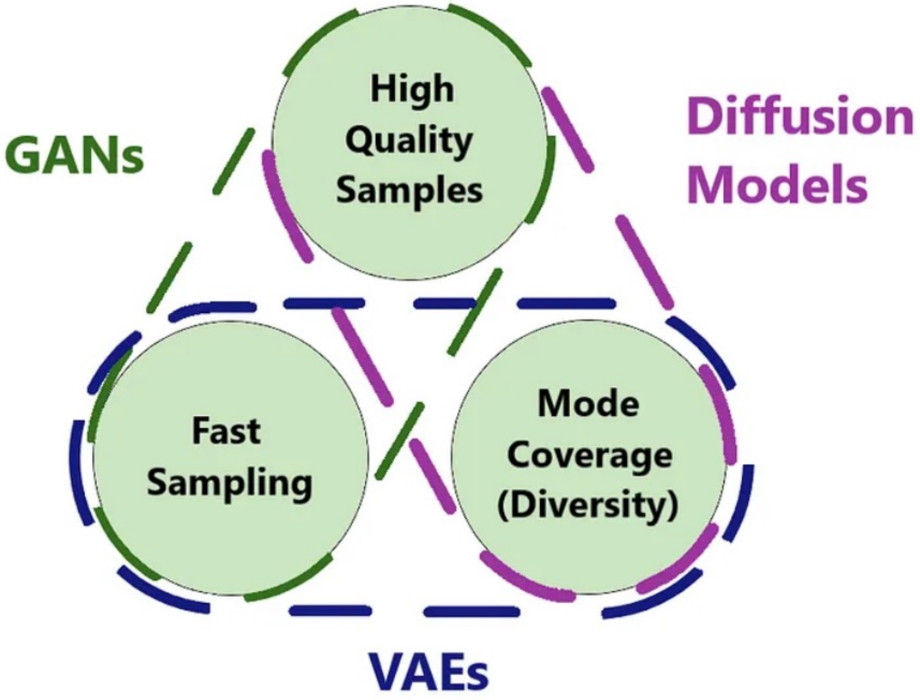
More applications: Text-to-Music



Smooth soft R&B song with tender vocals, romantic piano and groovy, funky bass.

Bright and groovy song featuring the piano that sounds like an opening theme for a comedy series.

Summary



GANs
Diffusion
Models
High
Quality
Samples
Fast
Sampling
Mode
Coverage
(Diversity)
VAEs

Summary of Computer Vision

- Compared to human vision, computer vision deals with the following tasks:
  - visual data acquisition (similar to human eyes but comes with many more choices)
  - image processing and feature extraction (mostly low-level)
  - analyze local structures and then 3D reconstruct the original scene (from mid-level to high-level)
  - understanding (mostly high-level)
  - generation (beyond the scope of human vision system)

Summary of Computer Vision

- Compared to human vision, computer vision deals with the following tasks:
  - visual data acquisition (similar to human eyes but comes with many more choices)
  - image processing and feature extraction (mostly low-level)
  - analyze local structures and then 3D reconstruct the original scene (from mid-level to high-level)
  - understanding (mostly high-level)
  - generation (beyond the scope of human vision system)
  - and further serving embodied agents to make decisions and take actions.

Please take Introduction to Embodied AI!

Thank You!