Generative Model

formulated as density estimation problems:

- explicit density estimation: explicitly define and solve for $p_{model}(x)$, which means I can exactly know the probability of an input to be in the data distribution.
- implicit density estimation: learn model that can sample from $p_{model}(x)$ without explicitly defining it.

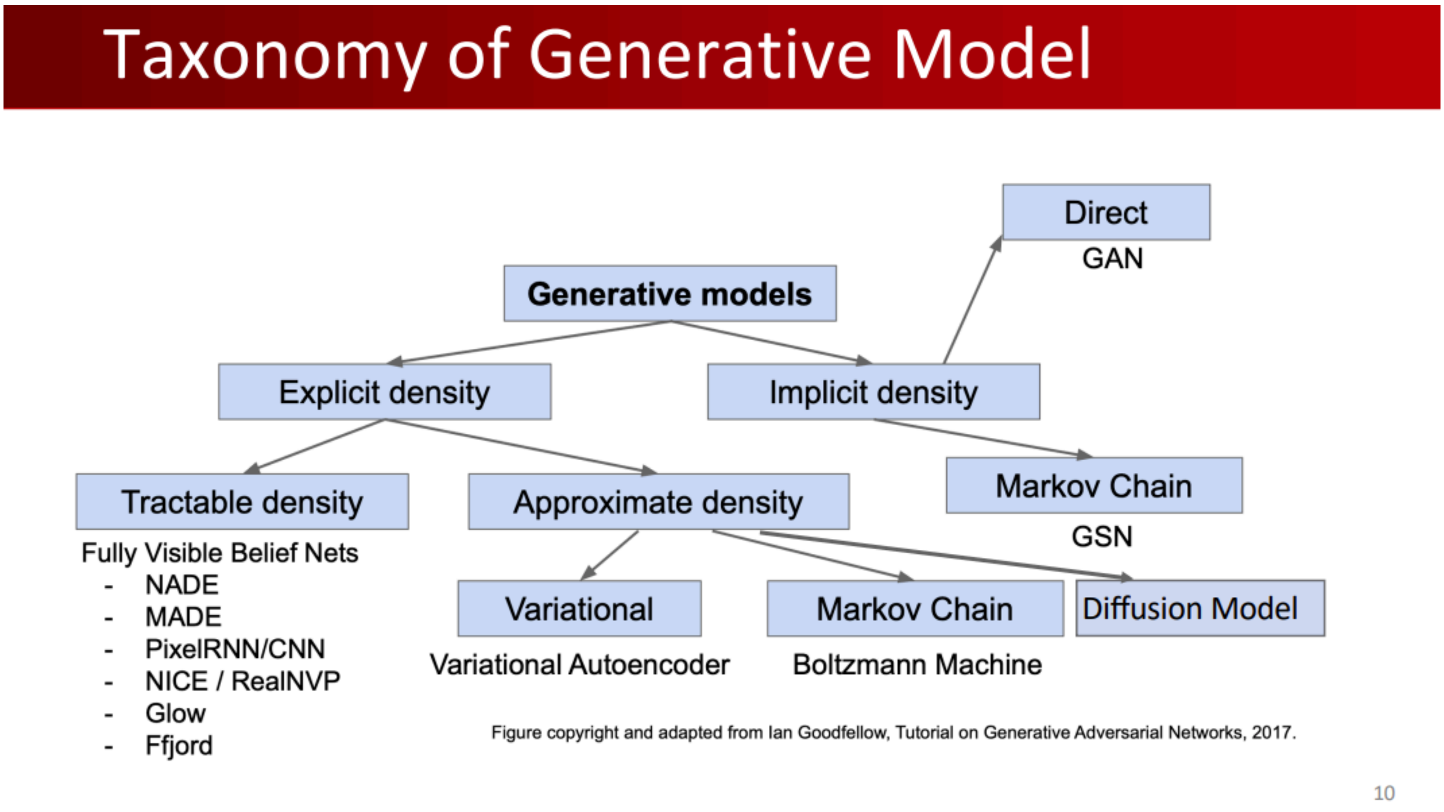


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Explicit Density -- tractable density

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

$$p(x) = \prod_{i=1}^{D} p(x_i | x_{<i})$$

Then maximize likelihood of training data.

several models for explicit density estimation: PixelRNN, PixelCNN
Pros:

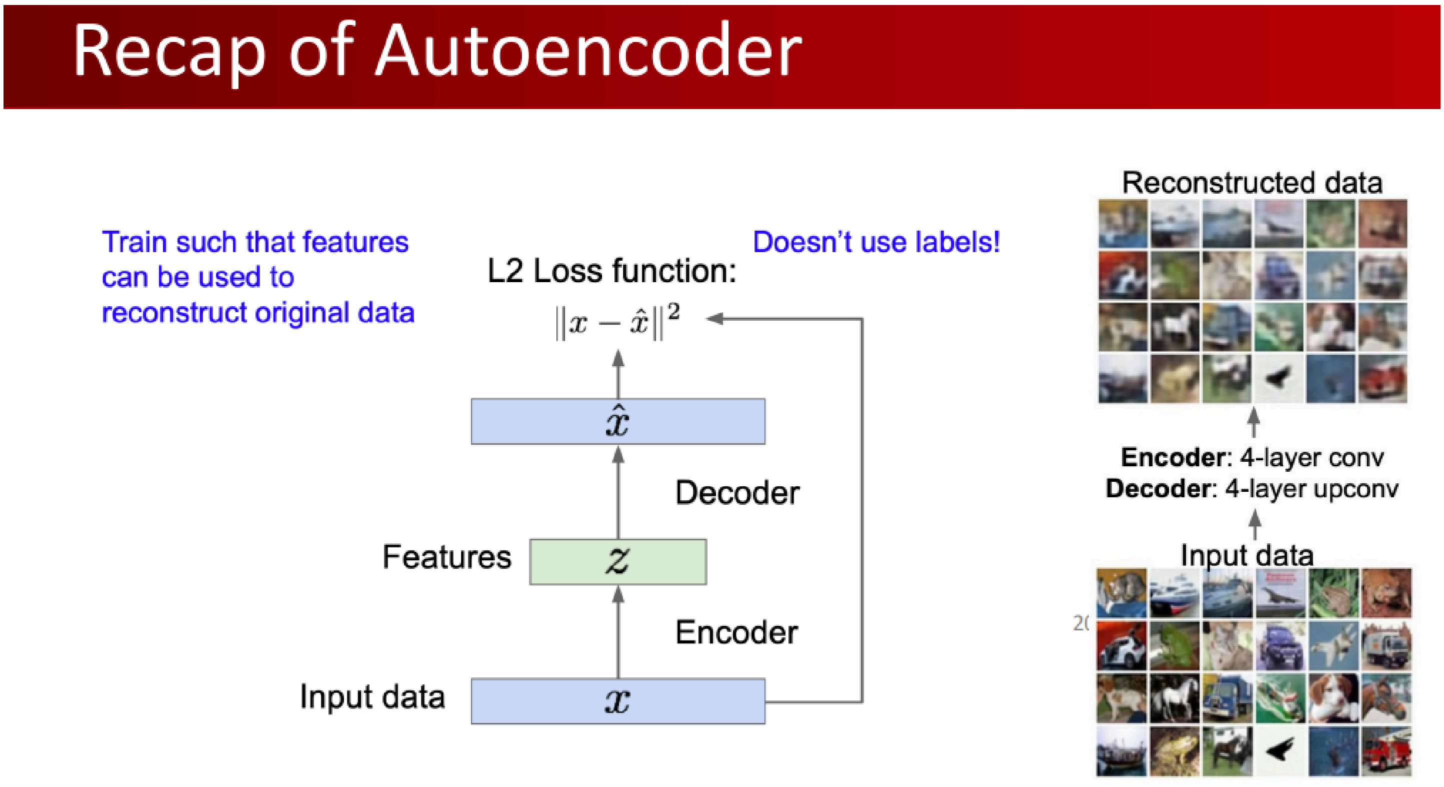
- can explicitly compute likelihood p(x)
- easy to optimize
- good samples

Cons:

- sequential generation -> slow

Explicity Density -- approximate density -- Variational Autoencoders (VAE)

Autoencoder



NOTE:

1. autoencoders can reconstruct data, and learn features to initialize a supervised model.
2. the features learned by an autoencoder captures factors of variation in training data.

Variational Autoencoder (VAE)

if we can sample from the feature space learned by autoencoder, we can generate random and valid images by simply decoding the latent variable. But the truth is don't know how to sample from the feature space.

Inorder to sample from the feature space, VAE made several assumptions:

1. the latent variable conforms to a standard normal distribution.
2. the decoder $p(x'|z)$ is a Gaussian, which predicts $\mu_{x'|z}$ and $\sigma_{x'|z}$
3. the encoder $p(z|x)$ is also a Guassian, which predicts $\mu_{z|x}$ and $\sigma_{z|x}$

Training of VAE:
First, the original target for decoder is to maximize the likelihood of true data, which is

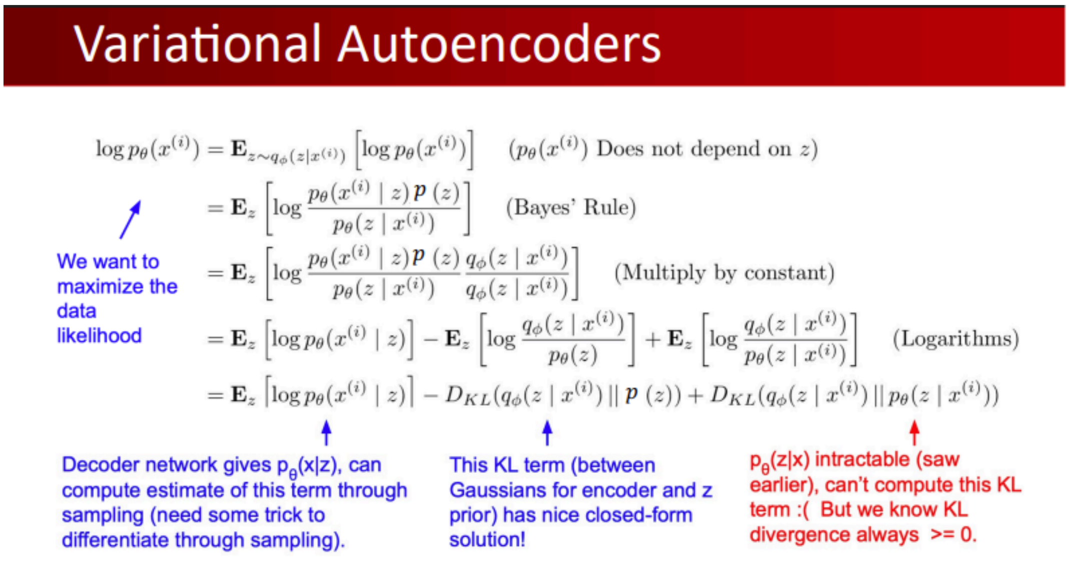
$$\int p(z)p_\theta(x|z)dz$$

Then we can use Monte Carlo sampling to approximate the integral. But the problem is the variance of Monte Carlo is pretty high(which mean the training process will be unstable)!

So let's try another way:

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

now the problem becomes what is $p_\theta(z|x)$? We don't know this distribution, but it looks exactly the same as the encoder $p(z|x)$ in autoencoder, we can learn another network to approximate this distribution. Still, we let the encoder learns a Gaussian.



Variational Autoencoders

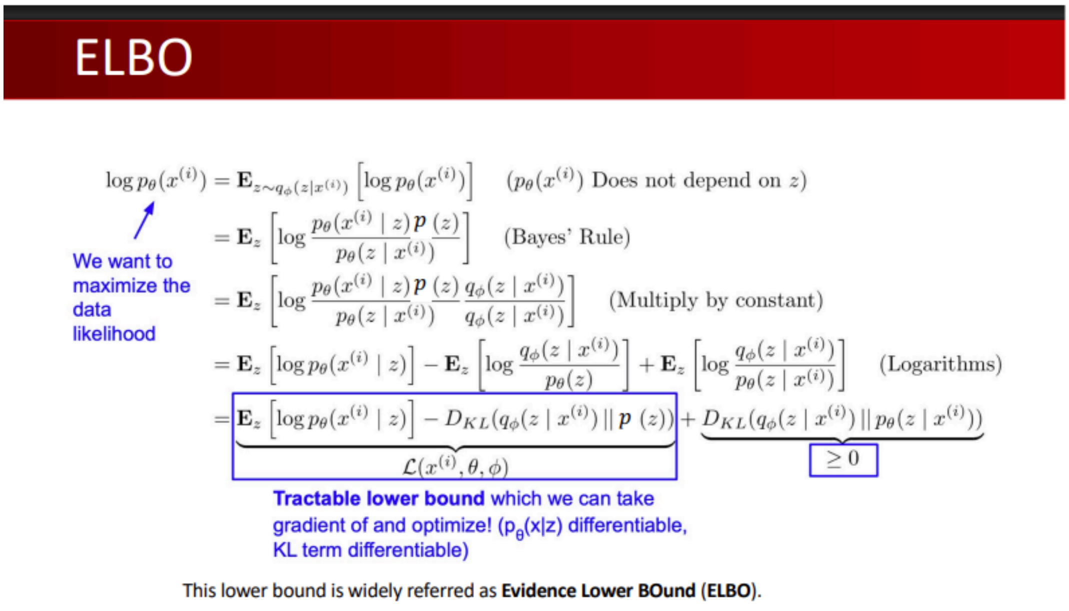
$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule})\\
&=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

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

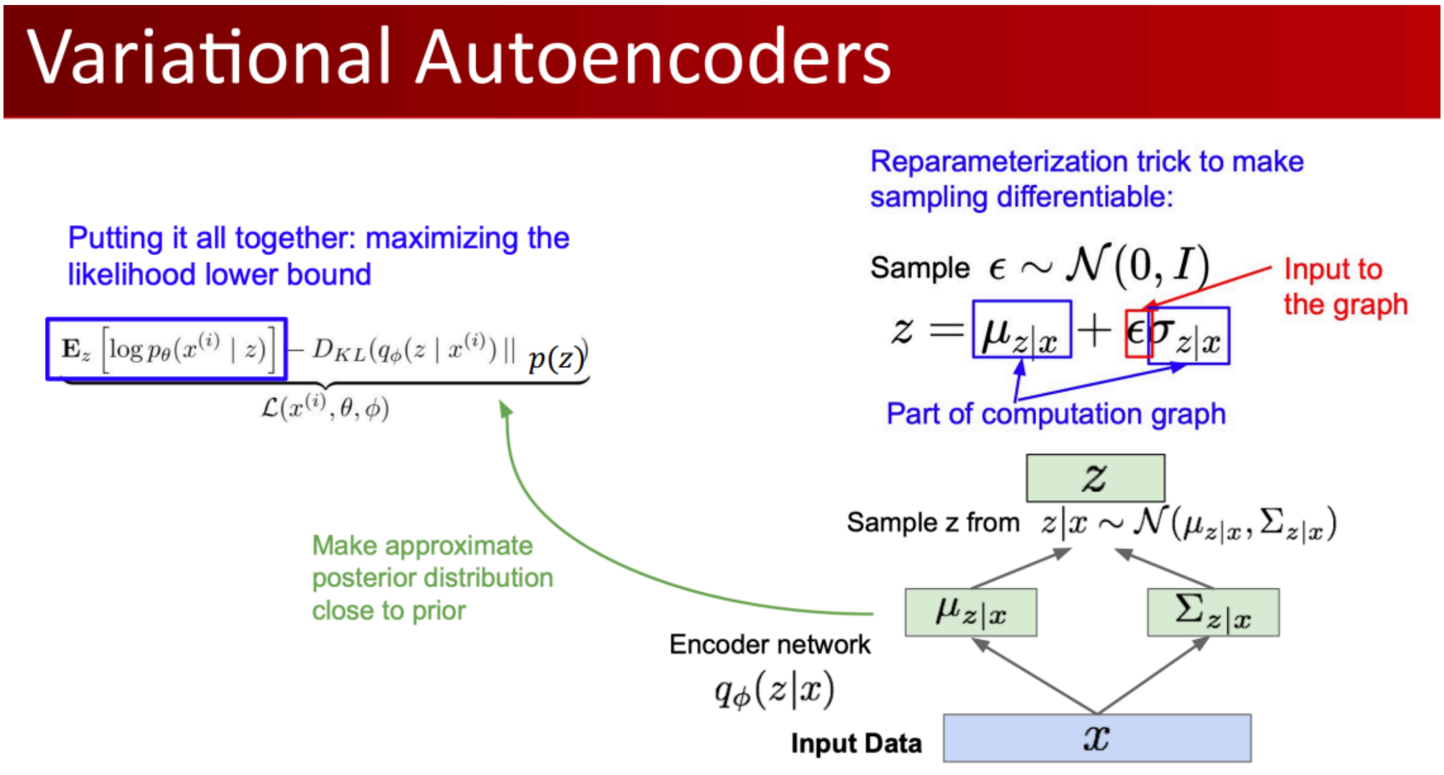
$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule})\\
&=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

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**).

So the last target we get is to maximize the ELBO. If we look at the fragments of ELBO, the first term is about reconstruct the input data as confident as possible, the second term is about make approximate posterior distribution close to prior(standard normal distribution). The first term still includes the expectation, so sampling is still needed, but the variance will be smaller because the distribution of z is estimated for this input image. The sampling process is reparametrized to be differentiable:


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 $\epsilon \sim \mathcal{N}(0, I)$
$z = \boxed{\mu_{z|x}} + \boxed{\epsilon}\boxed{\sigma_{z|x}}$
Input to the graph
Part of computation graph
z
Sample z from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$
Make approximate posterior distribution close to prior
$\mu_{z|x}$
$\Sigma_{z|x}$
Encoder network
$q_\phi(z|x)$
Input Data
x

However, one huge problem of VAE's target is that the two terms are actually conficting. The first term need the feature to be expressive enough to reconstruct the input data, but the second term expect that for any input, the output feature distribution of all be the same: standard normal distribution. So VAE is problematic. Also, because the decoder predicts a Guassian, the output is usually blurred.

After VAE is trained, the encoder is no longer used in inference. We simply sample $z\ N(0,1)$ and the decoder will predict the $\mu_{x|z}$ and $\Sigma_{x|z}$. Then we can sample new image as $x|z \sim N(\mu_{x|z}, \Sigma_{x|z})$.

To sum up:
Pros:

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

Cons:

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

Implicit Density -- Generative Adversarial Networks (GAN)

In GAN, we don't want the ability to model the density, we just want the ability to sample. But it's really hard to sample from the images' space (HxWxCx3, dimension is too high!) So the idea is to sample from a simple distribution, eg. random noise, and learn a transformation to map it to the image space. And instead of modeling a Guassian traget(network predict μ and σ for each pixel), we prefer a deterministic target, which is usually of higher quality(less blurry).

Now the question is how should we evaluate the quality of the generation network? We don't model the density, so we cannot use maximal likelihood of real images. So we instead use a discriminator network to evaluate the quality of the generated images, which seperates the real images from the fake 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:

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

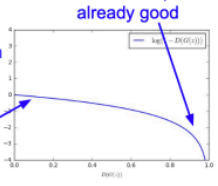
$$\min_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

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:

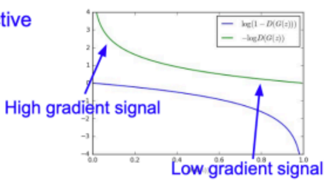
1. **Gradient ascent** on discriminator

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

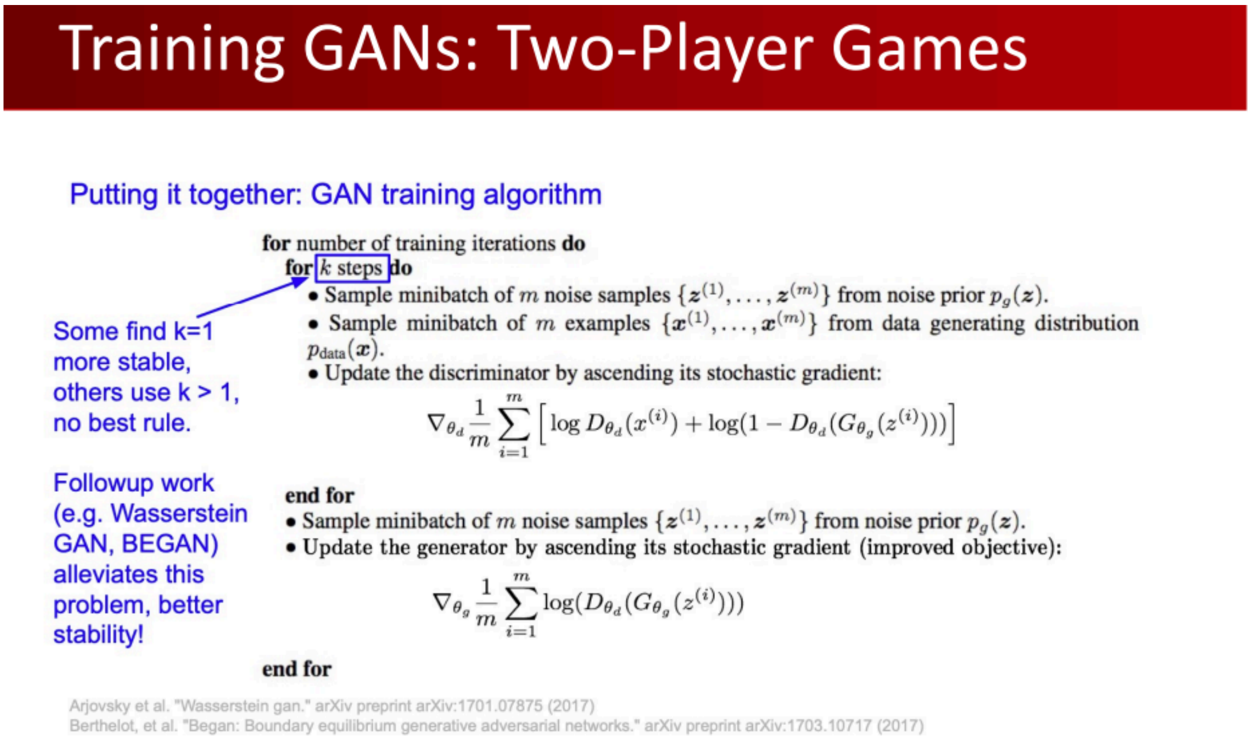
2. **Instead: Gradient ascent** on generator, different objective

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

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.



Algorithm:



Training GANs: Two-Player Games

Putting it together: GAN training algorithm

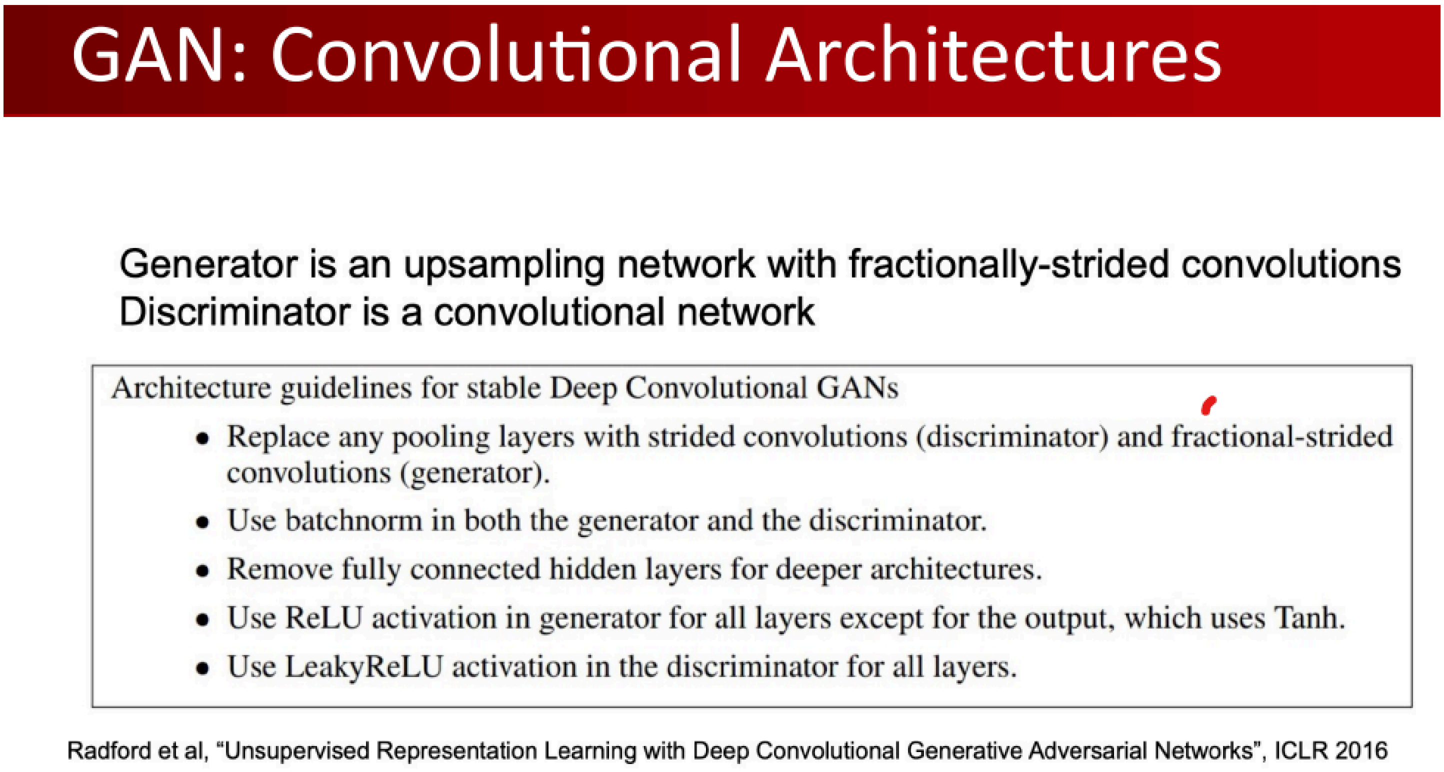
```
for number of training iterations do
  for $\boxed{k \text{ steps}}$ do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
    • 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 $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • 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
```

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!

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)

Special Case: Convolutional GAN



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

Evaluation of GAN:

1. Nearest neighbors: to detect overfittng, 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 presenta1on 1me (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: mode collapse means that the generator only generate a small set of pictures with high fidelity, and it obviously cannot cover the space of real data; while mode drop is kind of overfitting, it means the generator lacks imagination and can only generate samples it didn't see. 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.
4. Fréchet Incep1on 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. Lower FID means smaller distances between synthetic and real data distributions.

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

Special Attributes: the vanilla GAN(random noise -> image) has interpretable vector math, the input magically conforms to a linear attributes: glass man - no glass man + no glass women = glass women.

Sum up:
Pros:

- generate stat-of-the-art samples

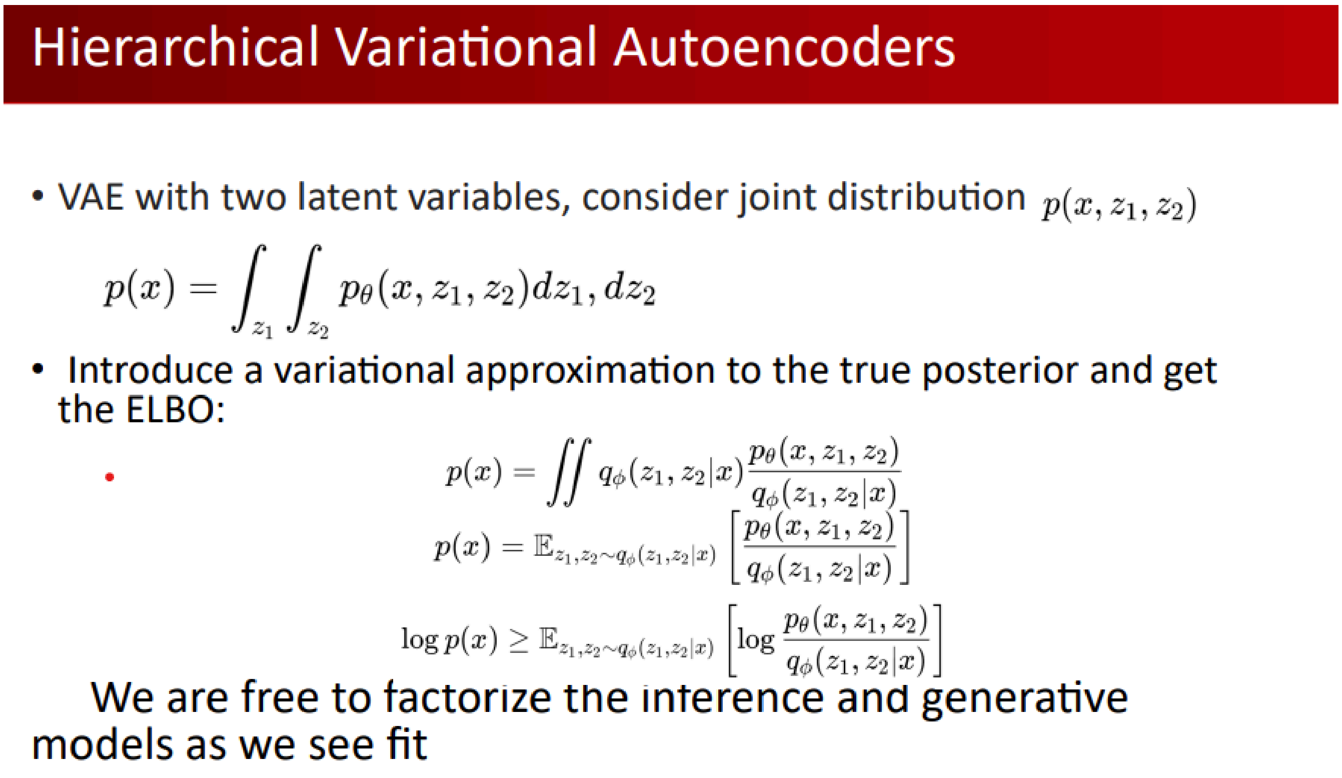
Cons:

- Tricker / More unstable to train
- Can't solve inference queries such as p(x), p(z|x) because we don't model the density

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

Explicity Density -- approximate density -- diffusion models

hierarchical VAE



Hierarchical Variational Autoencoders

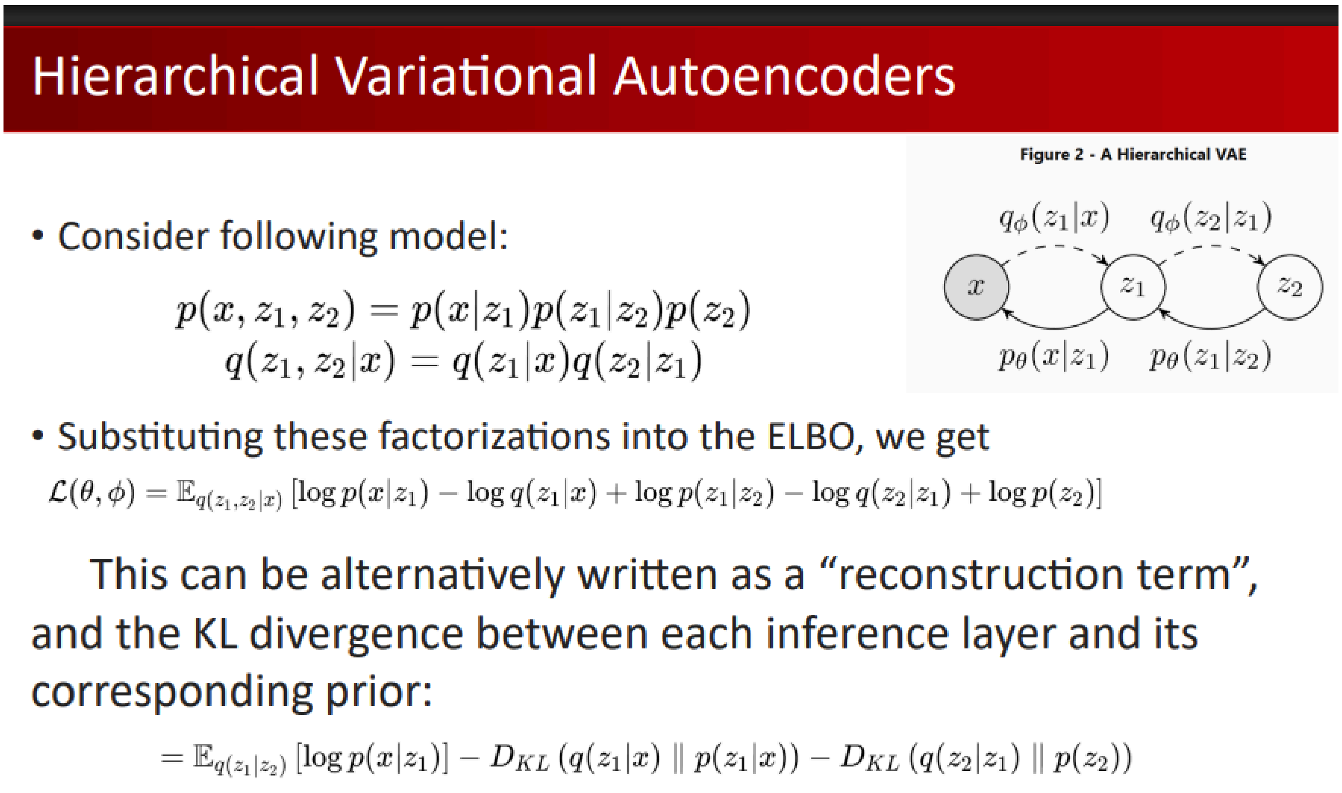
- VAE with two latent variables, consider joint distribution $p(x, z_1, z_2)$

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

- 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

- Consider following model:

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

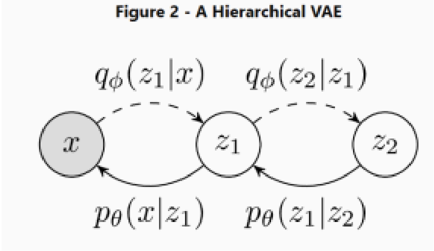


Figure 2 - A Hierarchical VAE

- Substituting these factorizations into the ELBO, we get

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

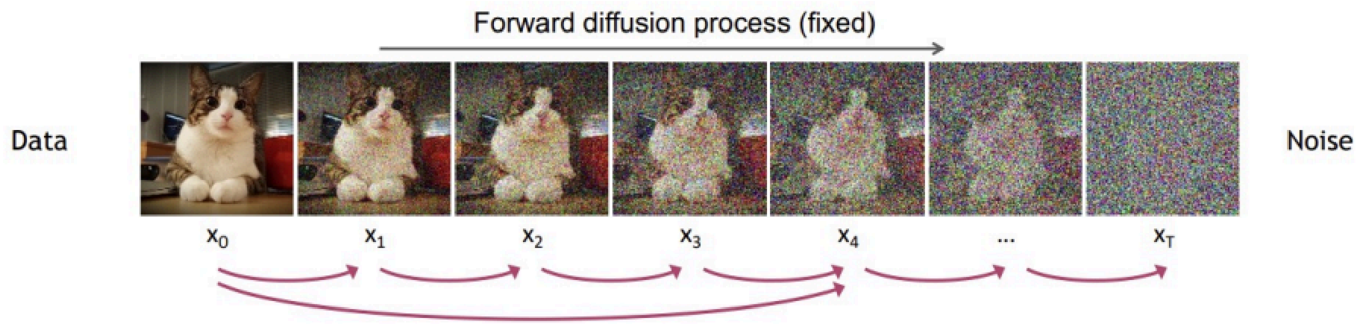
This can be alternatively written as a “reconstruction term”, and the KL divergence between each inference layer and its corresponding prior:

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

diffusion probabilistic models

diffusion model is like a specific realization of a hierachical VAE. where $q(x_t|x_{t-1}) = N(x_t; \sqrt{1-\beta_t}x_{t-1} + \beta_t I)$ and $q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$.

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 $\bar{\alpha}_T \rightarrow 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

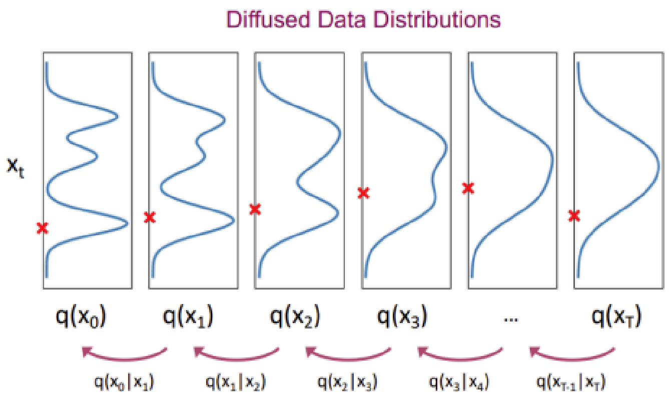
Generative Learning by Denoising

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

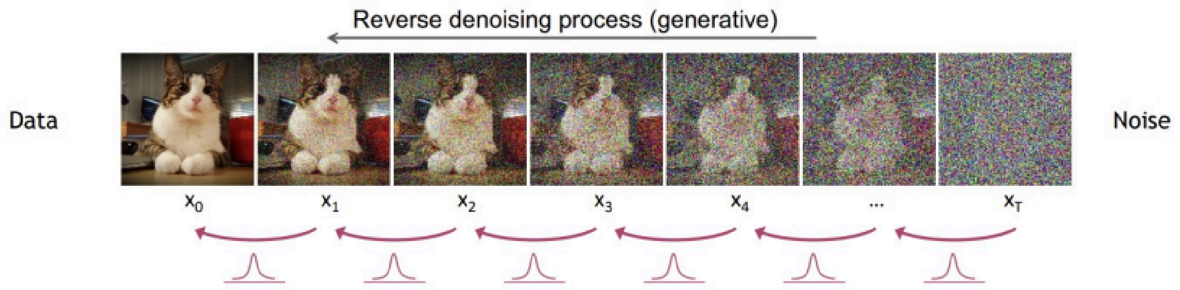
Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$



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:



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

Learning Denoising Model — Variational upper bound

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

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

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.

Io, 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

```
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boxed{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}, t) \right\|^2$
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:   $\mathbf{x}_{t-1} = \boxed{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)} + \sigma_t \mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

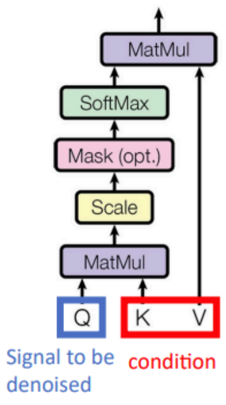
Conditional Diffusion Model

Almost the same as unconditional diffusion!

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

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 encoder)
- $\varphi_i(x_t)$ is the signal to be denoised