10-423/10-623 Generative AI

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Diffusion Models
+
Variational Inference

Matt Gormley

Lecture 8
Feb. 12, 2024

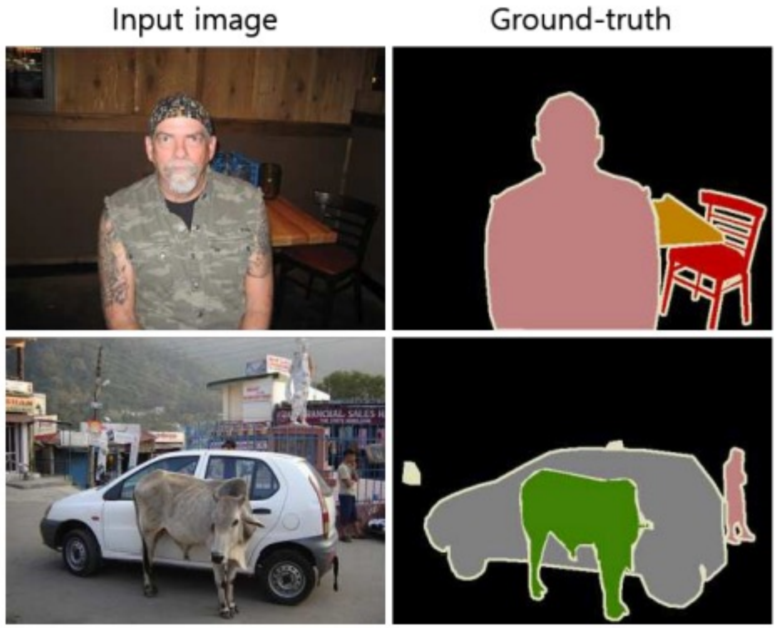
Reminders

- Homework 2: Generative Models of Images
  - Out: Thu, Feb 8
  - Due: Mon, Feb 19 at 11:59pm

U-NET

Semantic Segmentation

- Given an image, predict a label for every pixel in the image
- Not merely a classification problem, because there are strong correlations between pixel-specific labels



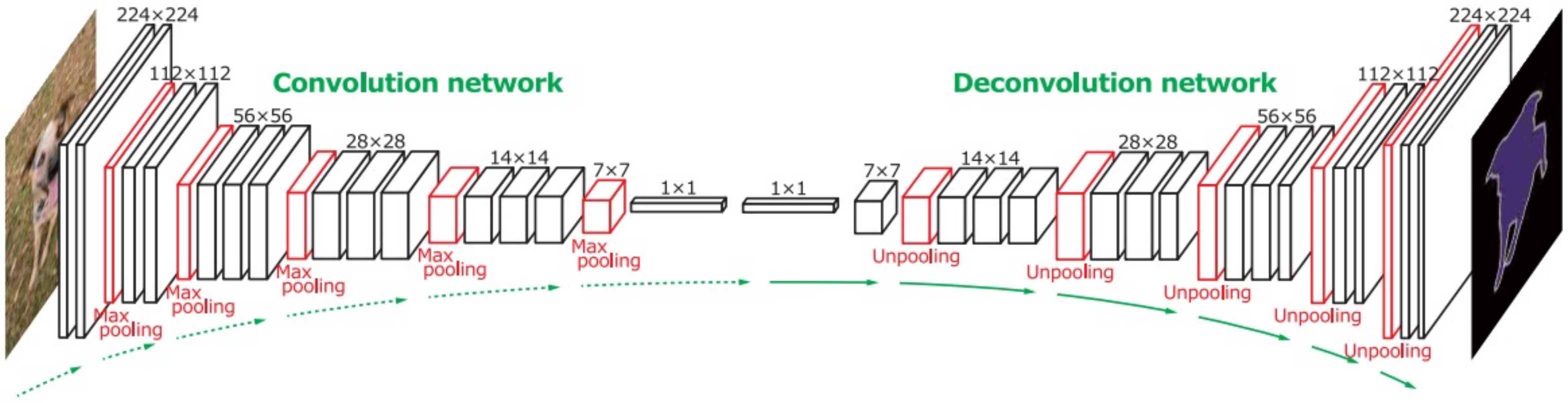


Figure from https://openaccess.thecvf.com/content_iccv_2015/papers/Noh_Learning_Deconvolution_Network_ICCV_2015_paper.pdf

Instance Segmentation

- Predict per-pixel labels as in semantic segmentation, but differentiate between different instances of the same label
- *Example*: if there are two people in the image, one person should be labeled **person-1** and one should be labeled **person-2**

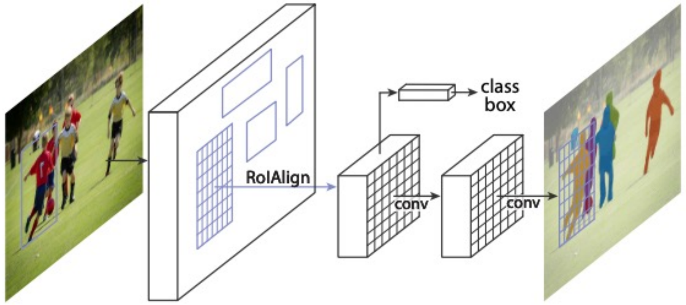
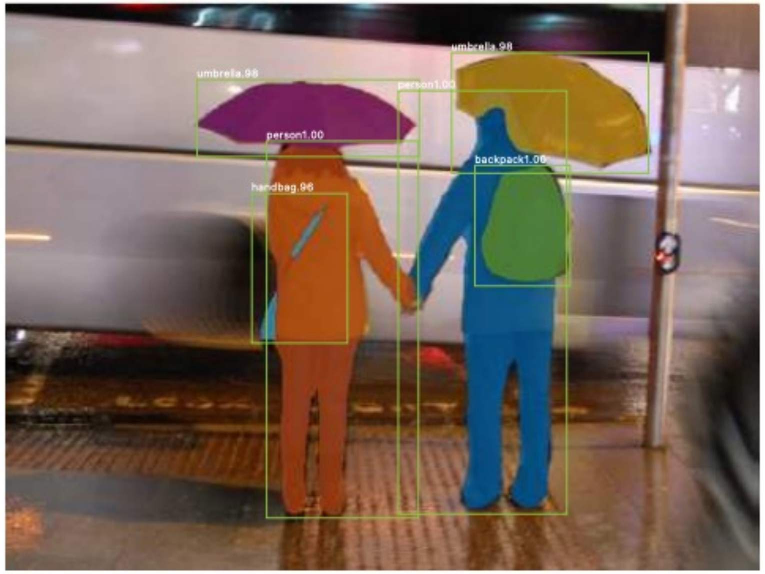


Figure 1. The **Mask R-CNN** framework for instance segmentation.



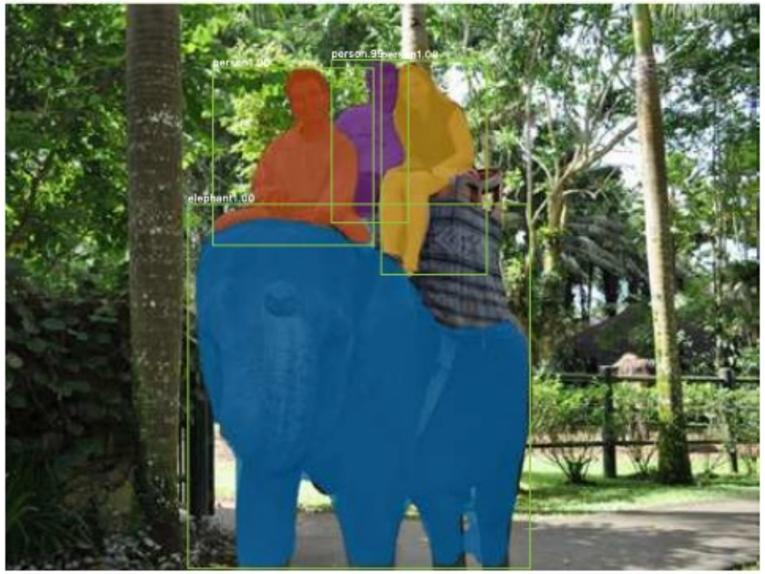


Figure from https://openaccess.thecvf.com/content_ICCV_2017/papers/He_Mask_R-CNN_ICCV_2017_paper.pdf

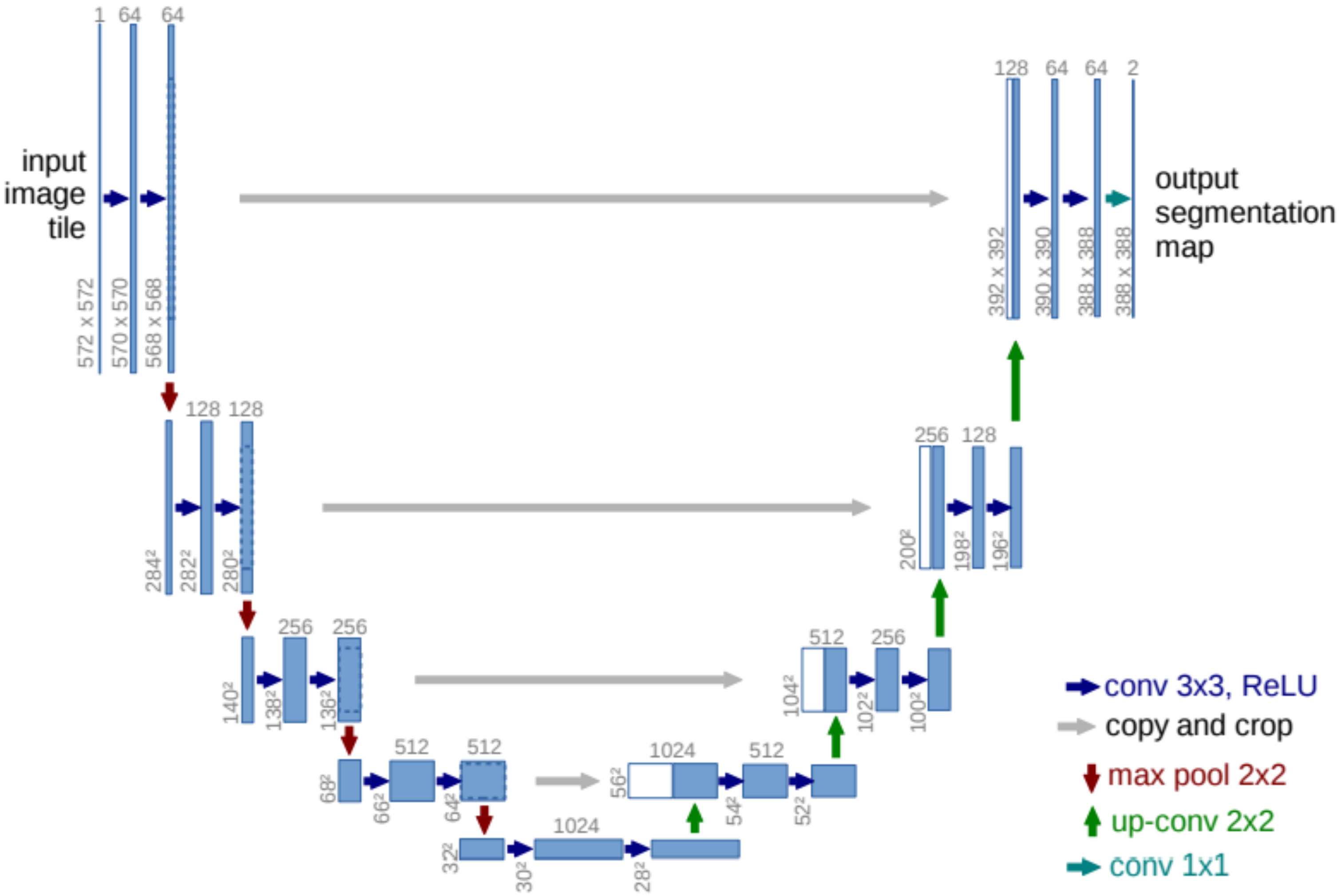
U-Net

Contracting path

- block consists of:
  - 3x3 convolution
  - 3x3 convolution
  - ReLU
  - max-pooling with stride of 2 (downsample)
- repeat the block N times, doubling number of channels

Expanding path

- block consists of:
  - 2x2 convolution (upsampling)
  - concatenation with contracting path features
  - 3x3 convolution
  - 3x3 convolution
  - ReLU
- repeat the block N times, halving the number of channels



U-Net

- Originally designed for applications to biomedical segmentation
- Key observation is that the output layer has the **same** dimensions as the input image (possibly with different number of channels)

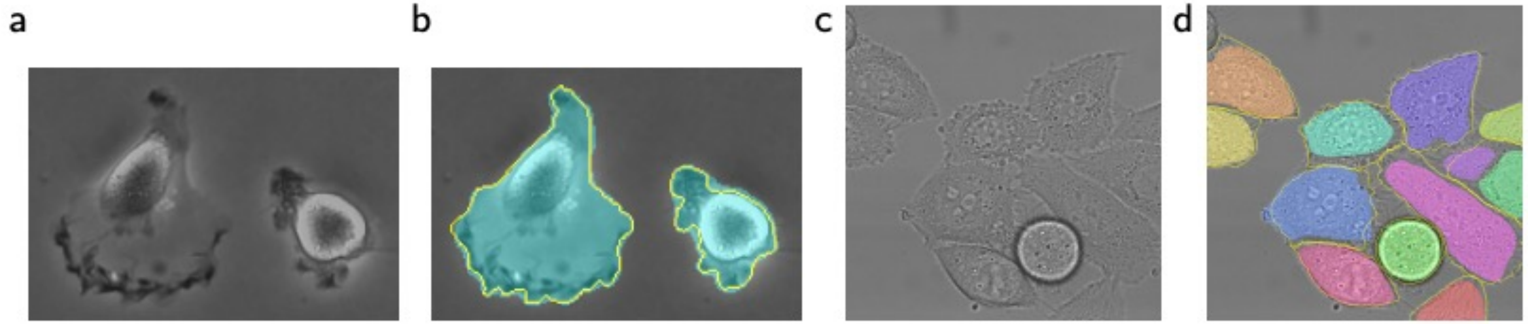


Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

UNSUPERVISED LEARNING

Unsupervised Learning

Assumptions:

1. our data comes from some distribution $q(\mathbf{x}_0)$
2. we choose a distribution $p_\theta(\mathbf{x}_0)$ for which sampling $x_0 \sim p_\theta(\mathbf{x}_0)$ is tractable

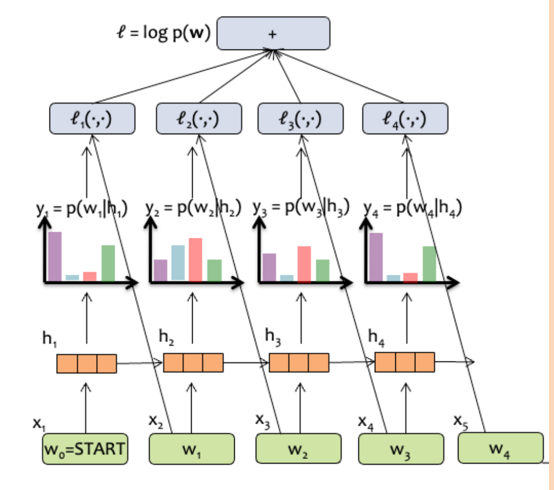
Goal: learn θ s.t. $p_\theta(\mathbf{x}_0) \approx q(\mathbf{x}_0)$

Unsupervised Learning

Assumptions:

1. our data comes from some distribution $q(\mathbf{x}_0)$
2. we choose a distribution $p_\theta(\mathbf{x}_0)$ for which sampling $x_0 \sim p_\theta(\mathbf{x}_0)$ is tractable

Goal: learn θ s.t. $p_\theta(\mathbf{x}_0) \approx q(\mathbf{x}_0)$



Example: autoregressive LMs

- true $q(\mathbf{x}_0)$ is the (human) process that produced text on the web
- choose $p_\theta(\mathbf{x}_0)$ to be an autoregressive language model
  - autoregressive structure means that $p(\mathbf{x}_t \mid \mathbf{x}_1, \ldots, \mathbf{x}_{t-1}) \sim \text{Categorical}(.)$ and ancestral sampling is exact/efficient
- learn by finding

  $$\theta \approx \text{argmax}_\theta \log(p_\theta(\mathbf{x}_0))$$

  using gradient based updates on

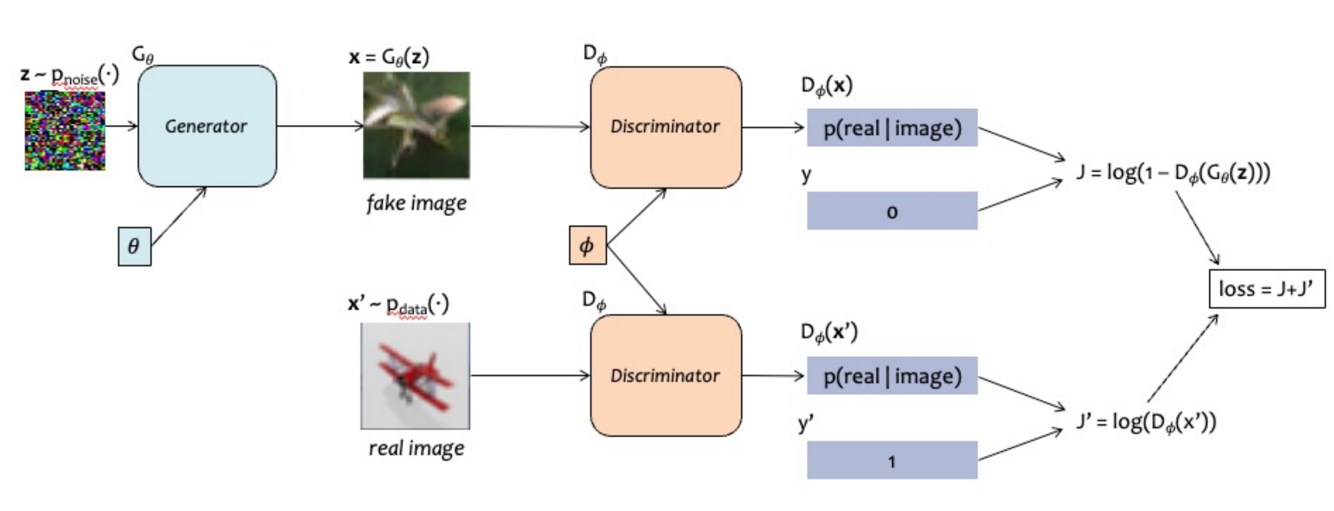
  $$\nabla_\theta \log(p_\theta(\mathbf{x}_0))$$

Unsupervised Learning

Assumptions:

1. our data comes from some distribution $q(\mathbf{x}_0)$
2. we choose a distribution $p_\theta(\mathbf{x}_0)$ for which sampling $x_0 \sim p_\theta(\mathbf{x}_0)$ is tractable

Goal: learn θ s.t. $p_\theta(\mathbf{x}_0) \approx q(\mathbf{x}_0)$



so optimize a minimax loss instead

Example: GANs

- true $q(\mathbf{x}_0)$ is distribution over photos taken and posted to Flikr
- choose $p_\theta(\mathbf{x}_0)$ to be an expressive model (e.g. noise fed into inverted CNN) that can generate images
  - sampling is typically easy: $\mathbf{z} \sim N(\mathbf{0}, \mathbf{I})$ and $\mathbf{x}_0 = f_\theta(\mathbf{z})$

learn by finding $\theta \approx \text{argmax}_\theta \log(p_\theta(\mathbf{x}_0))$?

- No! Because we can't even compute $\log(p_\theta(\mathbf{x}_0))$ or its gradient
- Why not? Because the integral is intractable even for a simple 1-hidden layer neural network with nonlinear activation

$$p(\mathbf{x}_0) = \int_{\mathbf{z}} p(\mathbf{x}_0 \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}$$

Unsupervised Learning

Assumptions:

1. our data comes from some distribution $q(\mathbf{x}_0)$
2. we choose a distribution $p_\theta(\mathbf{x}_0)$ for which sampling $x_0 \sim p_\theta(\mathbf{x}_0)$ is tractable

Goal: learn θ s.t. $p_\theta(\mathbf{x}_0) \approx q(\mathbf{x}_0)$

Example: Diffusion Models

- true $q(\mathbf{x}_0)$ is distribution over photos taken and posted to Flikr
- choose $p_\theta(\mathbf{x}_0)$ to be an expressive model (e.g. noise fed into inverted CNN) that can generate images
  - sampling is will be easy
- learn by finding $\theta \approx \text{argmax}_\theta \log(p_\theta(\mathbf{x}_0))$?
  - Sort of! We can't compute the gradient $\nabla_\theta \log(p_\theta(\mathbf{x}_0))$
  - So we instead optimize a variational lower bound (more on that later)

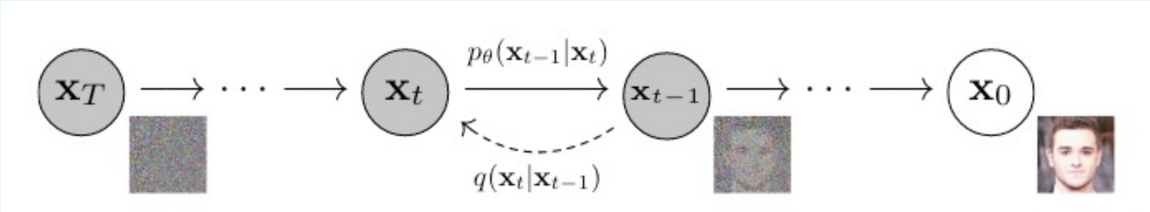


Figure from Ho et al. (2020)

Latent Variable Models

$p(x) = \int_z p(x|z)p(z)dz$

- For GANs, we assume that there are (unknown) **latent variables** which give rise to our observations
- The **noise vector z** are those latent variables
- After learning a GAN, we can **interpolate** between images in latent **z** space

Figure 4: Top rows: Interpolation between a series of 9 random points in Z show that the space learned has smooth transitions, with every image in the space plausibly looking like a bedroom. In the 6th row, you see a room without a window slowly transforming into a room with a giant window. In the 10th row, you see what appears to be a TV slowly being transformed into a window.

Figure from Radford et al. (2016)

DIFFUSION MODELS

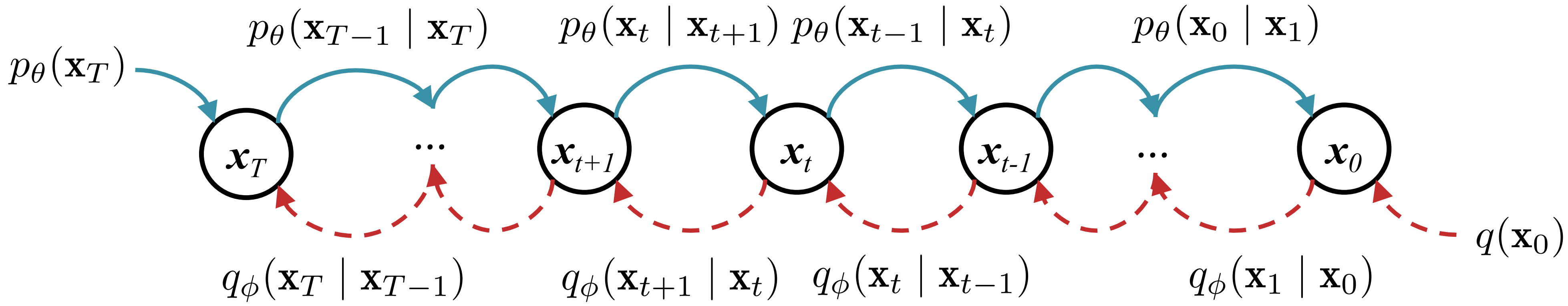
Diffusion Models

- Next we will consider (1) **diffusion mo**
variational autoencoders (VAEs)
  - Although VAEs came first, we're going to
models since they will receive more of ou
- The steps in defining these models is
  - Define a probability distribution involving
  - Use a variational lower bound as an obje
  - Learn the parameters of the probabi y
the objective function
- So what is a variational lower bound?

The standard presentation of diffusion models requires an understanding of variational inference. (we'll do that next time)

Today, we'll do an alternate presentation without variational inference!

Diffusion Model



Forward Process:

$$q_\phi(\mathbf{x}_{1:T}) = q(\mathbf{x}_0) \prod_{t=1}^{T} q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

(Learned) Reverse Process:

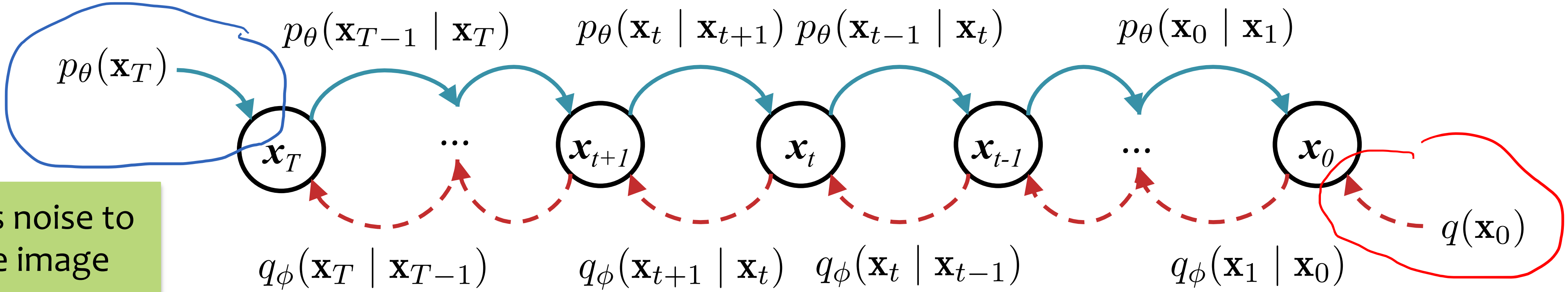
$$p_\theta(\mathbf{x}_{1:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

(Exact) Reverse Process:

$$q_\phi(\mathbf{x}_{1:T}) = q_\phi(\mathbf{x}_T) \prod_{t=1}^{T} q_\phi(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

The *exact* reverse process requires inference. And, even though $q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is simple, computing $q_\phi(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is intractable! Why? Because $q(\mathbf{x}_0)$ might be not-so-simple.

Diffusion Model



adds noise to the image

Forward Process:

if we could sample from this we'd be done

$$q_\phi(\mathbf{x}_{1:T}) = \boxed{q(\mathbf{x}_0)} \prod_{t=1}^{T} q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

(Learned) Reverse Process:

removes noise

$$p_\theta(\mathbf{x}_{1:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} \boxed{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}$$

goal is to learn this

(Exact) Reverse Process:

$$q_\phi(\mathbf{x}_{1:T}) = q_\phi(\mathbf{x}_T) \prod_{t=1}^{T} q_\phi(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

$= \int_{x_{1:T}} \quad dx_{1:T}$

The *exact* reverse process requires inference. And, even though $q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is simple, computing $q_\phi(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is intractable! Why? Because $q(\mathbf{x}_0)$ might be not-so-simple.

Diffusion Model

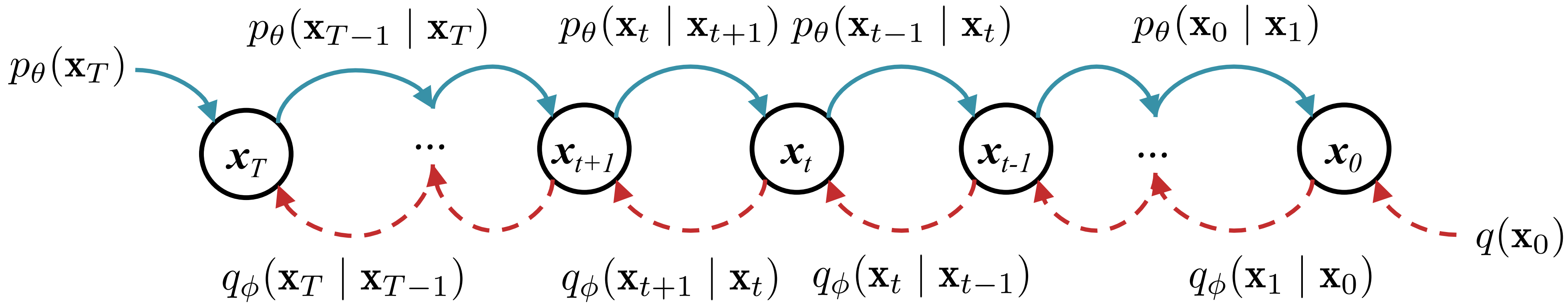
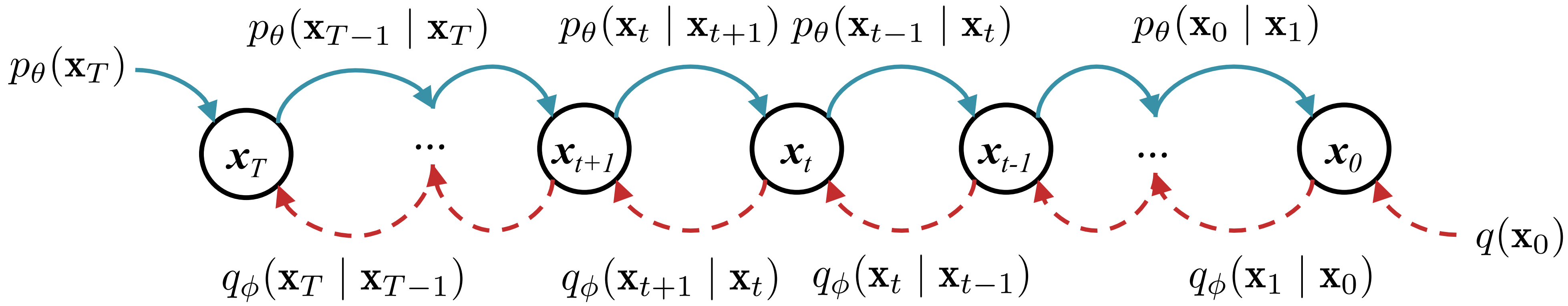


Figure from Ho et al. (2020)

Diffusion Model



Forward Process:

$$q_\phi(\mathbf{x}_{1:T}) = q(\mathbf{x}_0) \prod_{t=1}^{T} q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

Wait! Q: If q_ϕ is just adding noise, how can p_θ be interesting at all?

A: B/c $q(x_0)$ is not just a noise dist. and p_θ must capture that interesting variably

(Learned) Reverse Process:

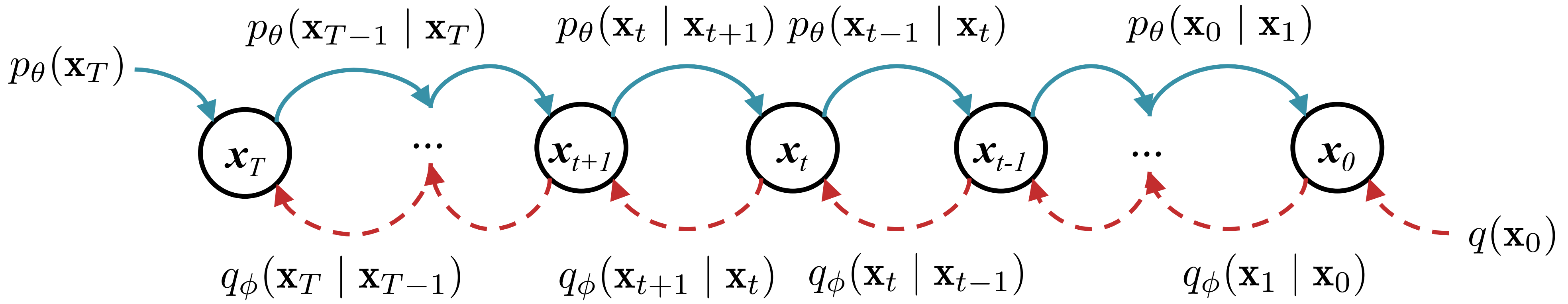
$$p_\theta(\mathbf{x}_{1:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

Wait! Q: But if $p_\theta(x_{t-1} | x_t)$ is Gaussian how can it learn a θ s.t. the $p_\theta(x_0) \approx q(x_0)$? Won't $p_\theta(x_0)$ be Gaussian too!?

A: No. In fact, a diffusion model of sufficiently long timespan T can capture any smooth target distribution

Diffusion Model Analogy

Denoising Diffusion Probabilistic Model (DDPM)



Forward Process:

$$q_\phi(\mathbf{x}_{1:T}) = q(\mathbf{x}_0) \prod_{t=1}^{T} q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

$$q(\mathbf{x}_0) = \text{data distribution}$$

$$q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \sim \mathcal{N}(\sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I})$$

(Learned) Reverse Process:

$$p_\theta(\mathbf{x}_{1:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

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

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$$

Denoising Diffusion Probabilistic Model (DDPM)

Noise schedule:

We choose α_t to follow a fixed schedule s.t. $q_\phi(\mathbf{x}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, just like $p_\theta(\mathbf{x}_T)$.

$x_{0:T}$

Forward Process:

$$q_\phi(\mathbf{x}_{1:T}) = q(\mathbf{x}_0) \prod_{t=1}^{T} q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

$x_{0:T}$

$$q(\mathbf{x}_0) = \text{data distribution}$$

$$q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \sim \mathcal{N}(\sqrt{\alpha_t}\mathbf{x}_{t-1}, (1 - \alpha_t)\mathbf{I})$$

(Learned) Reverse Process:

$$p_\theta(\mathbf{x}_{1:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

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

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$$

Gaussian (an aside)

Let $X \sim \mathcal{N}(\mu_x, \sigma_x^2)$ and $Y \sim \mathcal{N}(\mu_y, \sigma_y^2)$

1. Sum of two Gaussians is a Gaussian

$$X + Y \sim \mathcal{N}(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$$

2. Difference of two Gaussians is a Gaussian

$$X - Y \sim \mathcal{N}(\mu_x - \mu_y, \sigma_x^2 + \sigma_y^2)$$

3. Gaussian with a Gaussian mean has a Gaussian Conditional

$$Z \sim \mathcal{N}(\mu_z = X, \sigma_z^2) \Rightarrow P(Z \mid X) \sim \mathcal{N}(\cdot, \cdot)$$

4. But #3 does not hold if X is passed through a nonlinear function f

$$W \sim \mathcal{N}(\mu_z = f(X), \sigma_w^2) \nRightarrow P(W \mid X) \sim \mathcal{N}(\cdot, \cdot)$$

Properties of forward and *exact* reverse processes

Property #1:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$$

where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$

$\Rightarrow$ we can sample $\mathbf{x}_t$ from $\mathbf{x}_0$ at any timestep t efficiently in closed form

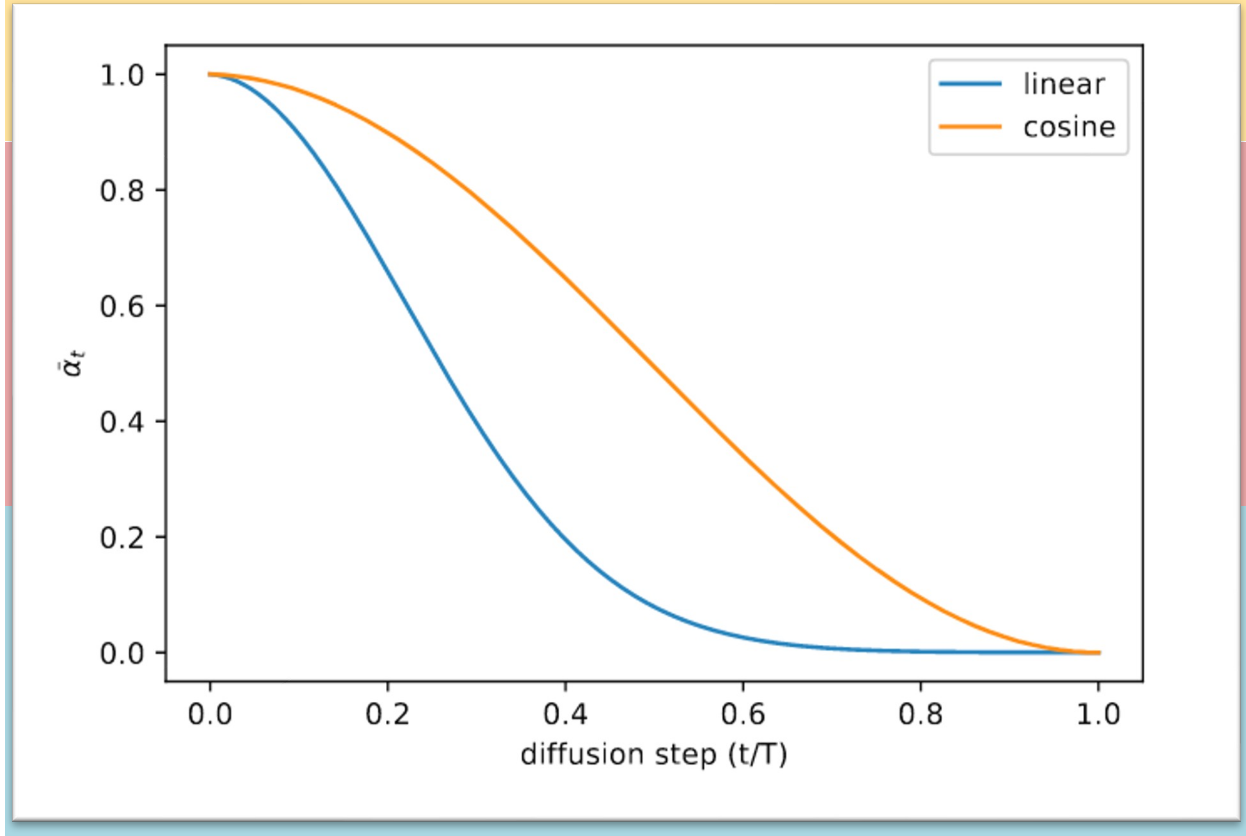
$\Rightarrow$ $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Denoising Diffusion Probabilistic Model (DDPM)

Noise schedule:

We choose α_t to follow a fixed schedule s.t. $q_\phi(\mathbf{x}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, just like $p_\theta(\mathbf{x}_T)$.

Q: What is $q(x_T | x_0)$? $S=1$ $q(x_T | x_0) \sim \mathcal{N}(\cancel{\mu} \approx 0, \Sigma \approx I)$



$$q(\mathbf{x}_0) = \text{data distribution}$$

$$q_\phi(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \sim \mathcal{N}(\sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I})$$

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

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Properties of forward and *exact* reverse processes

Property #1:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$

$\Rightarrow$ we can sample $\mathbf{x}_t$ from $\mathbf{x}_0$ at any timestep t efficiently in closed form

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

Property #2: Estimating $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is intractable because of its dependence on $q(\mathbf{x}_0)$. However, conditioning on $\mathbf{x}_0$ we can efficiently work with:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2\mathbf{I})$$

where $\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_t}(1-\alpha_t)}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}{1-\bar{\alpha}_t}\mathbf{x}_t$

$$= \alpha_t^{(0)}\mathbf{x}_0 + \alpha_t^{(t)}\mathbf{x}_t$$

$$\boxed{\sigma_t^2 = \frac{(1-\bar{\alpha}_{t-1})(1-\alpha_t)}{1-\bar{\alpha}_t}}$$

Parameterizing the *learned* reverse process

Recall: $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Parameterizing the *learned* reverse process

Recall: $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Idea #1: Rather than learn $\Sigma_\theta(\mathbf{x}_t, t)$ just use what we know about $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \sim \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2 \mathbf{I})$:

$$\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$$

Idea #2: Choose μ_θ based on $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$, i.e. we want $\mu_\theta(\mathbf{x}_t, t)$ to be close to $\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$. Here are three ways we could parameterize this:

Option A: Learn a network that approximates $\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$ directly from $\mathbf{x}_t$ and t:

$$\mu_\theta(\mathbf{x}_t, t) = \text{UNet}_\theta(\mathbf{x}_t, t)$$

where t is treated as an extra feature in UNet

Parameterizing the *learned* reverse process

Recall: $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Idea #1: Rather than learn $\Sigma_\theta(\mathbf{x}_t, t)$ just use what we know about $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \sim \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2 \mathbf{I})$:

$$\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$$

Idea #2: Choose μ_θ based on $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$, i.e. we want $\mu_\theta(\mathbf{x}_t, t)$ to be close to $\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$. Here are three ways we could parameterize this:

Option B: Learn a network that approximates the real $\mathbf{x}_0$ from only $\mathbf{x}_t$ and t:

$$\mu_\theta(\mathbf{x}_t, t) = \alpha_t^{(0)} \mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) + \alpha_t^{(t)} \mathbf{x}_t$$

where $\mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) = \text{UNet}_\theta(\mathbf{x}_t, t)$

Properties of forward and *exact* reverse processes

Property #1:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

$$\text{where } \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$

wrong! needs sqrt!

$\Rightarrow$ we can sample $\mathbf{x}_t$ from $\mathbf{x}_0$ at any timestep t efficiently in closed form

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

Property #2: Estimating $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is intractable because of its dependence on $q(\mathbf{x}_0)$. However, conditioning on $\mathbf{x}_0$ we can efficiently work with:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2\mathbf{I})$$

$$\begin{aligned}\text{where } \tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) &= \frac{\sqrt{\bar{\alpha}_t}(1-\alpha_t)}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}{1-\bar{\alpha}_t}\mathbf{x}_t \\ &= \alpha_t^{(0)}\mathbf{x}_0 + \alpha_t^{(t)}\mathbf{x}_t \\ \sigma_t^2 &= \frac{(1-\bar{\alpha}_{t-1})(1-\alpha_t)}{1-\bar{\alpha}_t}\end{aligned}$$

Property #3: Combining the two previous properties, we can obtain a different parameterization of $\tilde{\mu}_q$ which has been shown empirically to help in learning p_θ.

Rearranging $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}$ we have that:

typo

$$\mathbf{x}_0 = (\mathbf{x}_t + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}) / \sqrt{\bar{\alpha}_t}$$

Substituting this definition of $\mathbf{x}_0$ into property #2's definition of $\tilde{\mu}_q$ gives:

$$\begin{aligned}\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) &= \alpha_t^{(0)}\mathbf{x}_0 + \alpha_t^{(t)}\mathbf{x}_t \\ &= \alpha_t^{(0)}\left((\mathbf{x}_t + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}) / \sqrt{\bar{\alpha}_t}\right) + \alpha_t^{(t)}\mathbf{x}_t \\ &= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{(1-\alpha_t)}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\right)\end{aligned}$$

Parameterizing the *learned* reverse process

Recall: $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Idea #1: Rather than learn $\Sigma_\theta(\mathbf{x}_t, t)$ just use what we know about $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \sim \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2 \mathbf{I})$:

$$\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$$

Idea #2: Choose μ_θ based on $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$, i.e. we want $\mu_\theta(\mathbf{x}_t, t)$ to be close to $\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$. Here are three ways we could parameterize this:

Option C: Learn a network that approximates the $\boldsymbol{\epsilon}$ that gave rise to $\mathbf{x}_t$ from $\mathbf{x}_0$ in the forward process from $\mathbf{x}_t$ and t:

$$\mu_\theta(\mathbf{x}_t, t) = \alpha_t^{(0)} \mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) + \alpha_t^{(t)} \mathbf{x}_t$$

where $\mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) = (\mathbf{x}_{\cancel{0}\,t} + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)) / \sqrt{\bar{\alpha}_t}$

where $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) = \text{UNet}_\theta(\mathbf{x}_t, t)$

Learning the Reverse Process

Depending on which of the options for parameterization we pick, we get a different training algorithm.

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Algorithm 1 Training (Option A, all timesteps)

initialize θ

for $e \in \{1, \dots, E\}$ **do**

$\quad$ $\boxed{\textbf{for } x_0 \in \mathcal{D} \textbf{ do}}$

$\quad\quad$ **for** $t \in \{1, \dots, T\}$ **do**

$\quad\quad\quad$ $\cancel{t \sim \text{Uniform}(1, \dots, T)}$

$\quad\quad\quad$ $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$\quad\quad\quad$ $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$

$\quad\quad\quad$ $\tilde{\mu}_q \leftarrow \alpha_t^{(0)}\mathbf{x}_0 + \alpha_t^{(t)}\mathbf{x}_t$

$\quad\quad\quad$ $\ell_t(\theta) \leftarrow \|\tilde{\mu}_q - \mu_\theta(\mathbf{x}_t, t)\|^2$

$\quad\quad$ $\theta \leftarrow \theta - \nabla_\theta \sum_{t=1}^{T} \ell_t(\theta)$

in practice, this is batched

Learning the Reverse Process

Depending on which of the options for parameterization we pick, we get a different training algorithm.

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Algorithm 1 Training (Option A)

initialize θ
for $e \in \{1, \ldots, E\}$ **do**
  $\boxed{\textbf{for } x_0 \in \mathcal{D} \textbf{ do}}$ batched
    $t \sim \text{Uniform}(1, \ldots, T)$
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$
    $\tilde{\mu}_q \leftarrow \alpha_t^{(0)} \mathbf{x}_0 + \alpha_t^{(t)} \mathbf{x}_t$
    $\ell_t(\theta) \leftarrow \|\tilde{\mu}_q - \mu_\theta(\mathbf{x}_t, t)\|^2$
    $\theta \leftarrow \theta - \nabla_\theta \ell_t(\theta)$

Q: why use just one value of t?

A: the gradients for any x_0 across different t are correlated

Learning the Reverse Process

Depending on which of the options for parameterization we pick, we get a different training algorithm.

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Algorithm 1 Training (Option B)

```
initialize θ
for e ∈ {1, ..., E} do
    for x_0 ∈ D do
        t ~ Uniform(1, ..., T)
        ϵ ~ N(0, I)
        x_t ← sqrt(ᾱ_t) x_0 + sqrt(1 − ᾱ_t) ϵ
        ℓ_t(θ) ← ||x_0 − x_θ^(0)(x_t, t)||^2
        θ ← θ − ∇_θ ℓ_t(θ)
```

$$
\begin{aligned}
&\text{initialize } \theta\\
&\textbf{for } e \in \{1, \ldots, E\} \textbf{ do}\\
&\quad \textbf{for } x_0 \in \mathcal{D} \textbf{ do}\\
&\qquad t \sim \text{Uniform}(1, \ldots, T)\\
&\qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\\
&\qquad \mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}\\
&\qquad \ell_t(\theta) \leftarrow \|\mathbf{x}_0 - \mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t)\|^2\\
&\qquad \theta \leftarrow \theta - \nabla_\theta \ell_t(\theta)
\end{aligned}
$$

Learning the Reverse Process

Depending on which of the options for parameterization we pick, we get a different training algorithm.

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

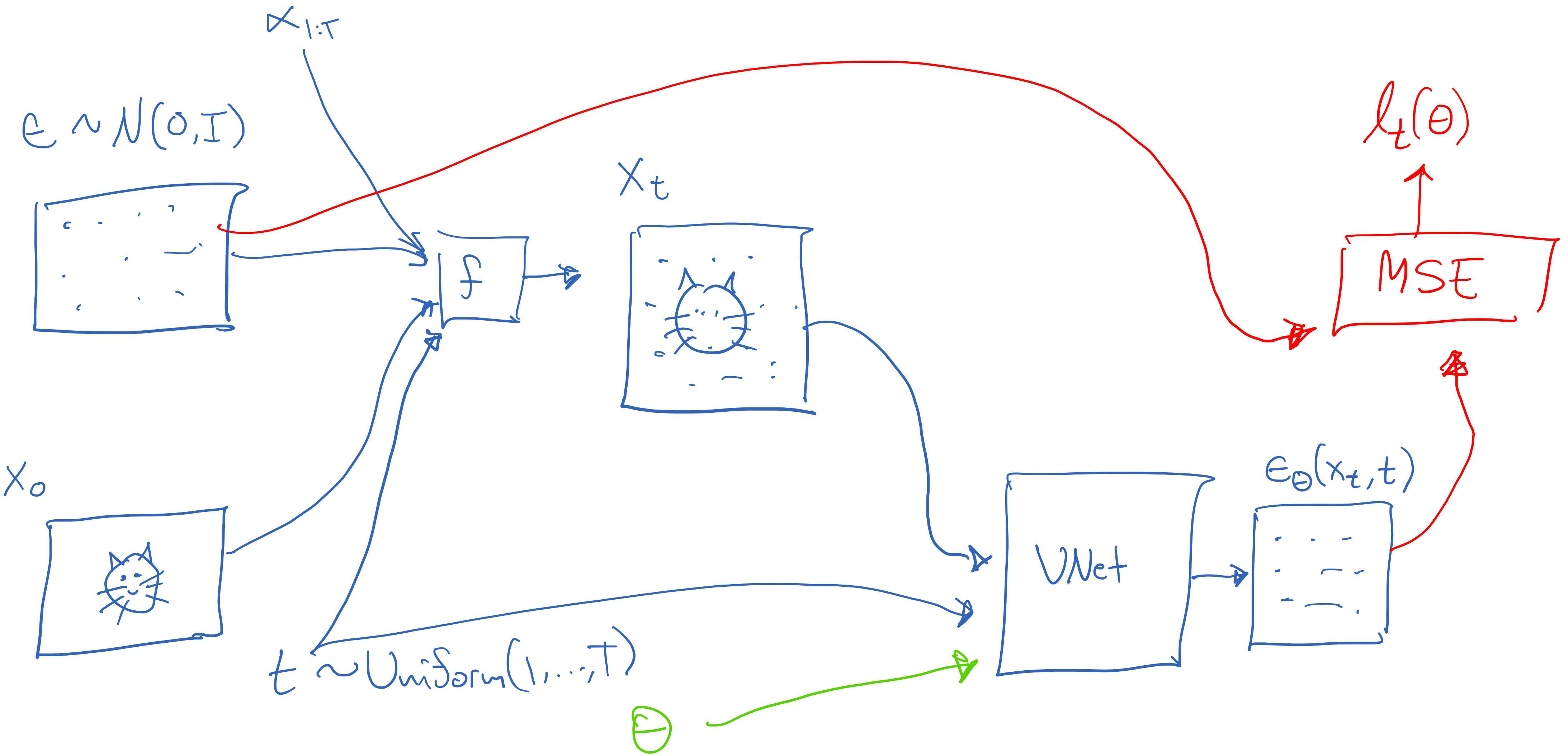
Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Algorithm 1 Training (Option C)

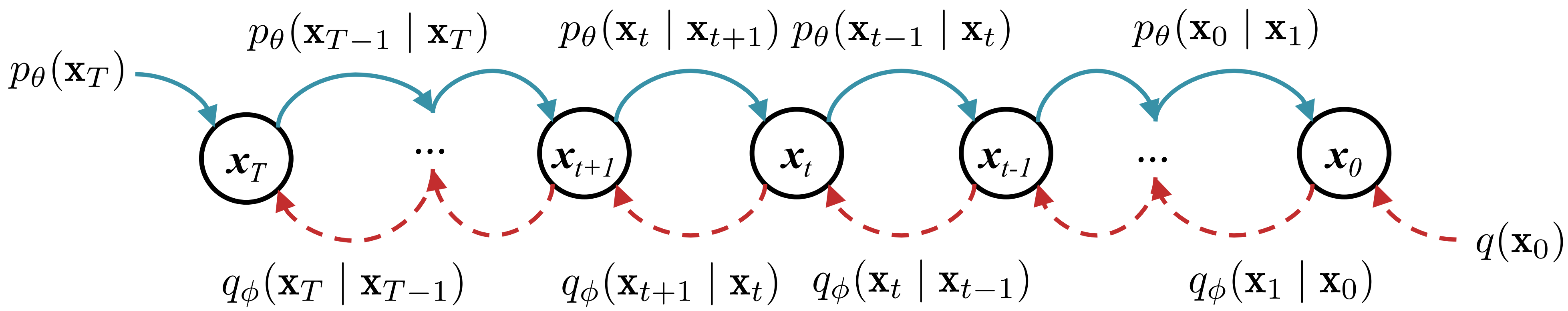
initialize θ
for $e \in \{1, \ldots, E\}$ **do**
    for $x_0 \in \mathcal{D}$ **do**
        $t \sim \text{Uniform}(1, \ldots, T)$
        $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
        $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$
        $\ell_t(\theta) \leftarrow \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2$
        $\theta \leftarrow \theta - \nabla_\theta \ell_t(\theta)$

Option C is the best empirically

Training (Computation Graph)



Sampling from the *learned* reverse process

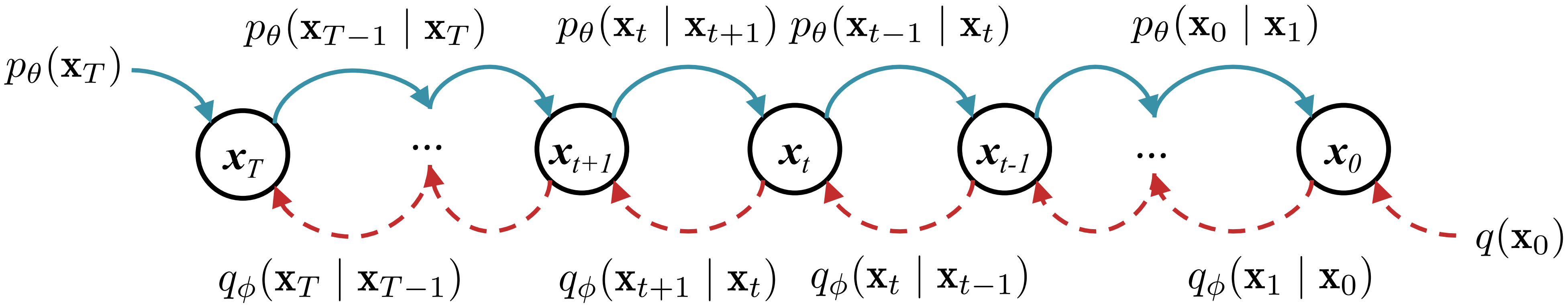


Algorithm 1 Sampling

T, ..., 1

$\mathbf{x}_T \sim p_\theta(\mathbf{x}_T)$
for $t \in \cancel{\boxed{\{1, \ldots, T\}}}$ **do**
  $\mathbf{x}_{t-1} \sim p(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$
return $\mathbf{x}_0$

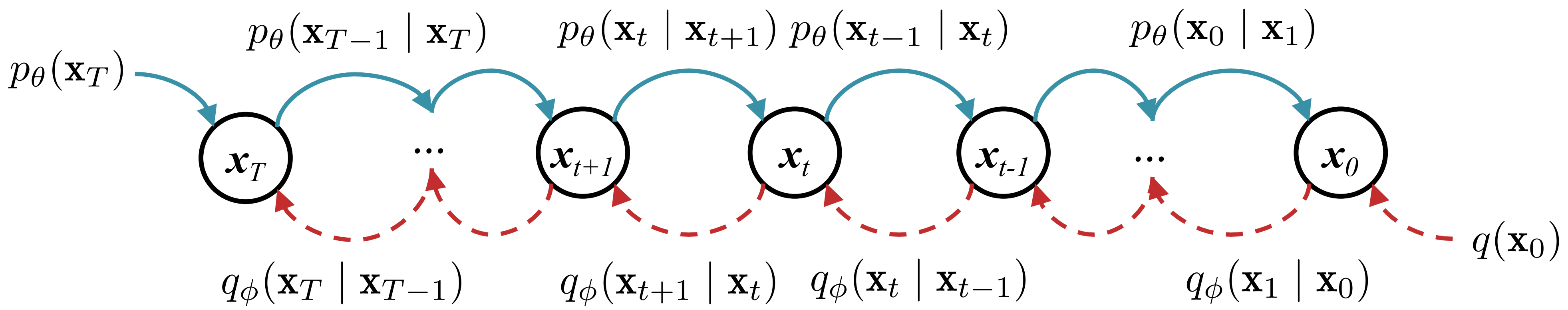
Sampling from the *learned* reverse process



Algorithm 1 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t \in \{1, \dots, T\}$ **do**
3: $\quad \mathbf{x}_{t-1} \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$ $\Rightarrow X_{t-1} = \mu_\theta(x_t, t) + \Sigma_\theta(x_t, t)\,\epsilon$
where $\epsilon \sim \mathcal{N}(0, I)$
4: **return** $\mathbf{x}_0$

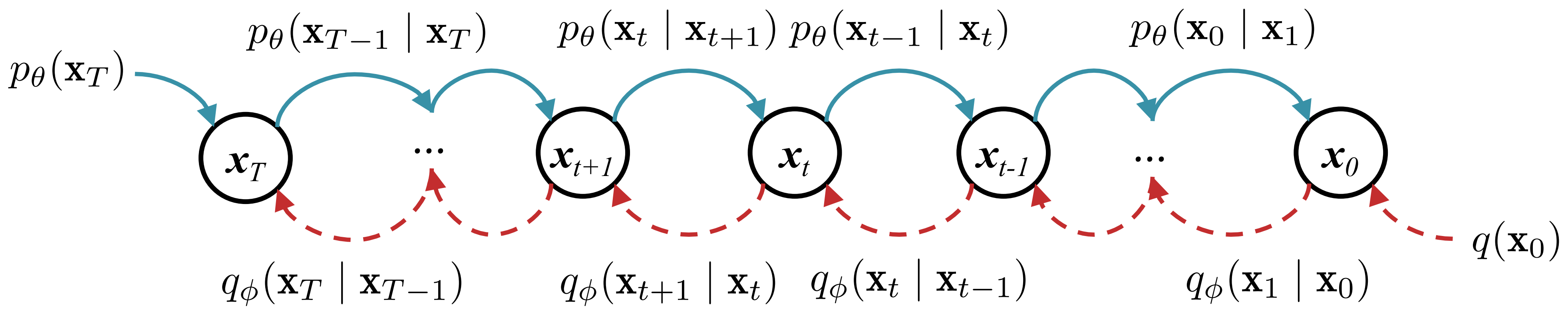
Sampling from the *learned* reverse process



Algorithm 1 Sampling (Option A)

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t \in \{1, \ldots, T\}$ **do**
3: $\quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4: $\quad \mathbf{x}_{t-1} \leftarrow \mu_\theta(\mathbf{x}_t, t) + \sigma_t^2 \boldsymbol{\epsilon}$
5: **return** $\mathbf{x}_0$

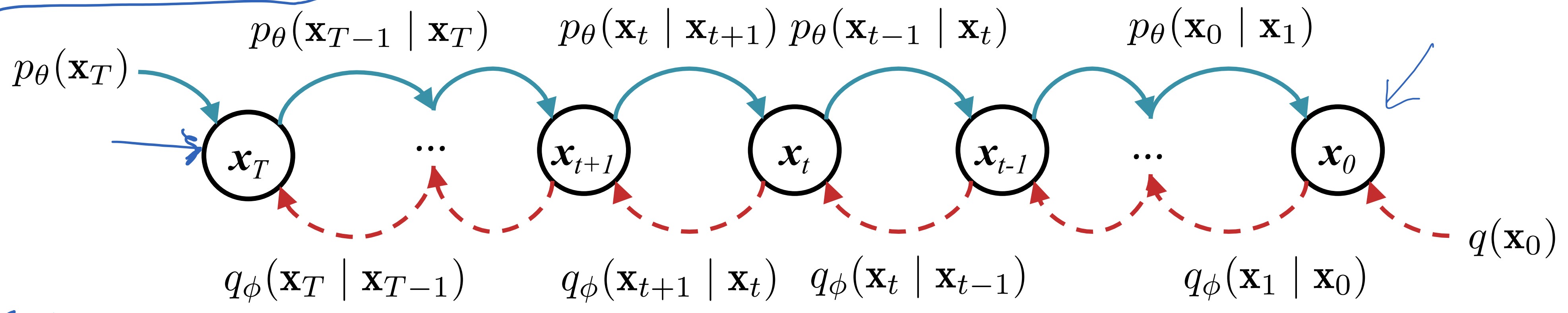
Sampling from the *learned* reverse process



Algorithm 1 Sampling (Option B)

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t \in \{1, \ldots, T\}$ **do**
3: $\quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4: $\quad \hat{\boldsymbol{\mu}}_t \leftarrow \alpha_t^{(0)} \mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) + \alpha_t^{(t)} \mathbf{x}_t$
5: $\quad \mathbf{x}_{t-1} \leftarrow \hat{\boldsymbol{\mu}}_t + \sigma_t^2 \boldsymbol{\epsilon}$
6: **return** $\mathbf{x}_0$

Sampling from the *learned* reverse process



Algorithm 1 Sampling (Option C)

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t \in \{1, \ldots, T\}$ **do**
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\hat{\mathbf{x}}_0 \leftarrow (\mathbf{x}_t + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)) / \sqrt{\bar{\alpha}_t}$
  $\hat{\boldsymbol{\mu}}_t \leftarrow \alpha_t^{(0)} \hat{\mathbf{x}}_0 + \alpha_t^{(t)} \mathbf{x}_t$
  $\mathbf{x}_{t-1} \leftarrow \hat{\boldsymbol{\mu}}_t + \sigma_t^2 \boldsymbol{\epsilon}$
return $\mathbf{x}_0$

$$p_\theta(x_0) = \int_{x_{1:T}} p(x_0, x_{1:T})\, dx_{1:T}$$

Monte Carlo estimate

$x_T \sim p(x_T)$

$\vdots$

$x_3 \sim p(x_3 \mid x_4)$

$x_2 \sim p(x_2 \mid x_3$

Unsupervised Learning

Assumptions:

1. our data comes from some distribution $q(\mathbf{x}_0)$
2. we choose a distribution $p_\theta(\mathbf{x}_0)$ for which sampling $x_0 \sim p_\theta(\mathbf{x}_0)$ is tractable

Goal: learn θ s.t. $p_\theta(\mathbf{x}_0) \approx q(\mathbf{x}_0)$

Example: Diffusion Models

- true $q(\mathbf{x}_0)$ is distribution over photos taken and posted to Flikr
- choose $p_\theta(\mathbf{x}_0)$ to be an expressive model (e.g. noise fed into inverted CNN) that can generate images
  - sampling is will be easy
- learn by finding $\theta \approx \text{argmax}_\theta \log(p_\theta(\mathbf{x}_0))$?
  - Sort of! We can't compute the gradient $\nabla_\theta \log(p_\theta(\mathbf{x}_0))$
  - So we instead optimize a variational lower bound (more on that later)

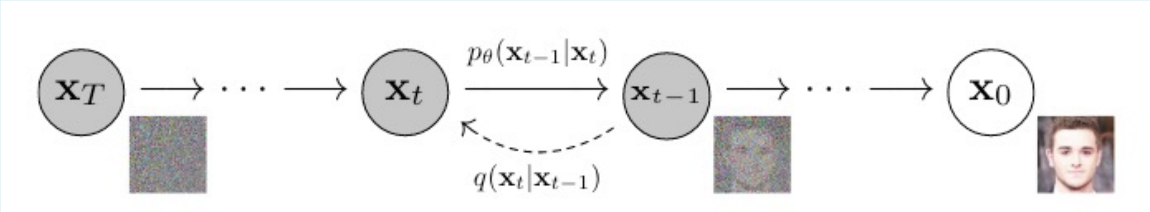


Figure from Ho et al. (2020)

DDPM Objective Function

$$\begin{tikzcd} x_t \arrow[r, bend left, "p_\theta(x_{t-1}|x_t)"] & x_{t-1} \arrow[l, bend left, "q(x_t|x_{t-1})"] \end{tikzcd}$$

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \overbrace{\mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right]}^{\text{ELBO}} = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] =: L$$

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

constant
wrt θ

can fold into
L_{t-1}

This KL divergence term L_{t-1} wants the two conditional distributions to be as close as possible.

Equations from Ho et al. (2020)

KL DIVERGENCE

KL Divergence

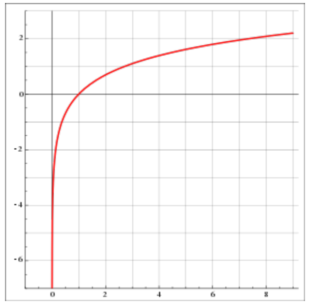
- *Definition*: for two distributions q(x) and p(x) over x ∈ $\mathcal{X}$, the **KL Divergence** is:

$$\text{KL}(q||p) = E_{q(x)}\left[\log \frac{q(x)}{p(x)}\right] = \begin{cases} \sum_x q(x) \log \frac{q(x)}{p(x)} \\ \int_x q(x) \log \frac{q(x)}{p(x)} dx \end{cases}$$

- *Properties*:
  - KL(q || p) measures the **proximity** of two distributions q and p
  - KL is **not** symmetric: KL(q || p) ≠ KL(p || q)
  - KL is minimized when q(x) = p(x) for all x ∈ $\mathcal{X}$

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

KL Divergence



Understanding the Behavior of KL as an objective function

Example 1: Keeping all else constant, consider the effect of a particular x’ on KL(q || p)

x’	q(x’)	p(x’)	q(x’) log(q(x’)/p(x’))	effect on KL(q || p)
1	0.9	0.9	0	no increase
2	0.9	0.1	1.97	big increase
3	0.1	0.9	-0.21	little decrease
4	0.1	0.1	0	little decrease

KL **does** insist on good approximations for values that have **high** probability in q

KL **does not** insist on good approximations for values that have **low** probability in q

Example 2: Which q distribution minimizes KL(q || p)?

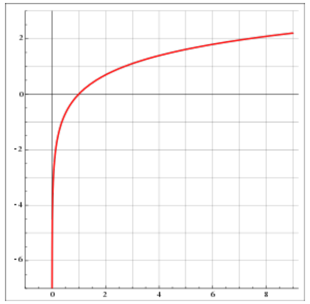
$$\mathbf{p} = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix} \quad \mathbf{q}^{(1)} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \mathbf{q}^{(2)} = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix} \quad \mathbf{q}^{(3)} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

Q: If we’re minimizing KL, why not return $q^{(3)}$?

A: Because it’s not a distribution!

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

KL Divergence

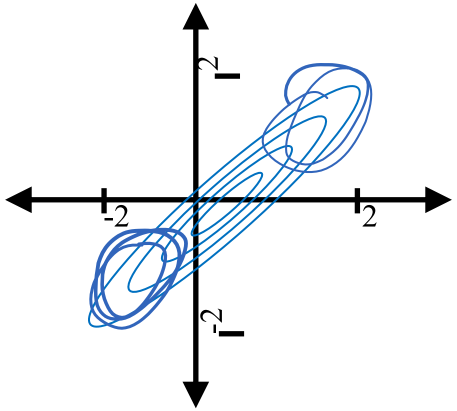


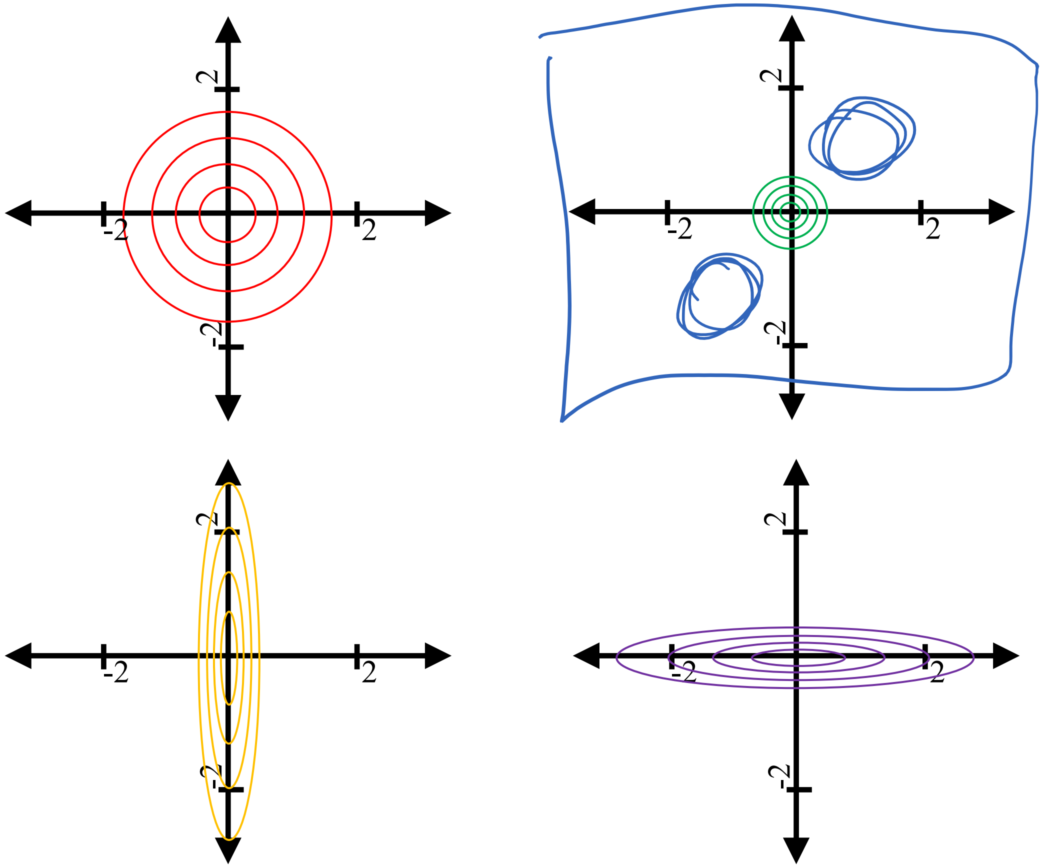
Understanding the Behavior of KL as an objective function

Example 3: Which q distribution minimizes KL(q || p)?

$$p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu} = [0,0]^T, \boldsymbol{\Sigma})$$

$$q(x_1, x_2) = \mathcal{N}_1(x_1 \mid \mu_1, \sigma_1^2)\mathcal{N}_2(x_2 \mid \mu_2, \sigma_2^2)$$





VARIATIONAL DIFFUSION MODELS AND VARITIONAL AUTOENCODERS (VAES)

Diffusion Models

- Next we will consider (1) **diffusion models** and (2) **variational autoencoders (VAEs)**
  - Although VAEs came first, we're going to dive into diffusion models since they will receive more of our attention
- The steps in defining these models is roughly:
  - Define a probability distribution involving Gaussian noise
  - Use a variational lower bound as an objective function
  - Learn the parameters of the probability distribution by optimizing the objective function
- So what is a variational lower bound?