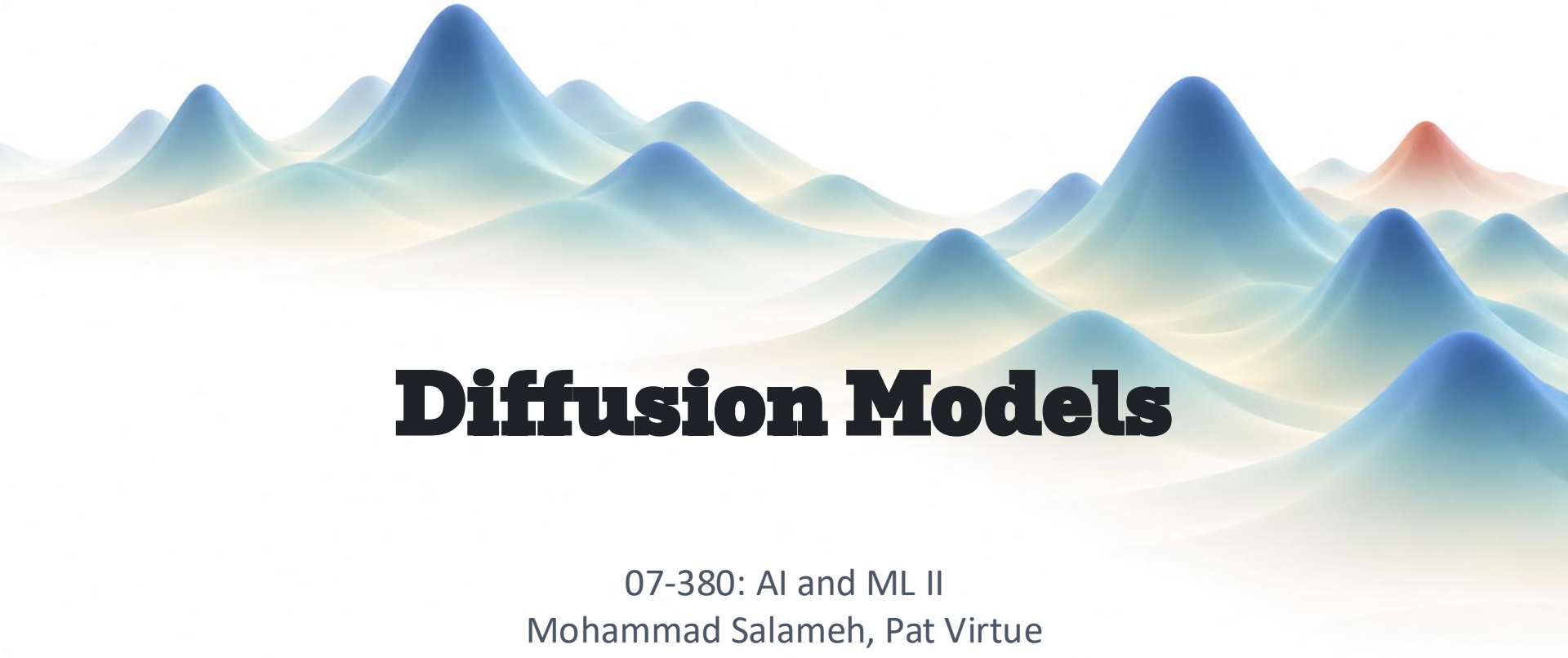


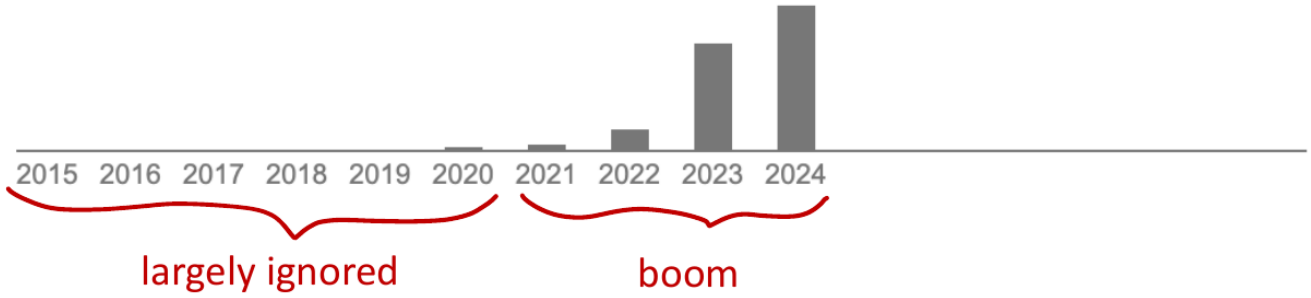
Diffusion Models

07-380: AI and ML II
Mohammad Salameh, Pat Virtue

Where Diffusion Models Began (2015)

Deep unsupervised learning using nonequilibrium thermodynamics

Authors	Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, Surya Ganguli
Publication date	2015/3/12
Journal	International Conference on Machine Learning
Total citations	Cited by 5630



Slide credit: Bilal Taha

Generative Modeling Objective

We observe samples from an unknown data distribution:

$$x_0 \sim p_{\text{data}}(x_0)$$

We choose a model $p_\theta(x_0)$ for which sampling is tractable, then learn

$$p_\theta(x_0) \approx p_{\text{data}}(x_0)$$

If the likelihood is tractable, maximum likelihood uses

$$\theta^* = \arg\max_\theta \ \mathbb{E}_{x_0 \sim p_{\text{data}}} [\log p_\theta(x_0)]$$

For latent-variable models,

$$p_\theta(x_0) = \int p_\theta(x_0, z)\, dz$$

and this marginalization is often intractable. VAEs and diffusion models therefore optimize a variational lower bound on the log likelihood.

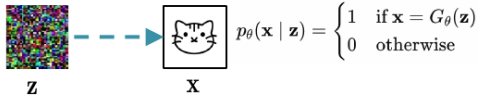
Slide credit: Bilal Taha

GAN → VAE → Diffusion

GANs

$p_\theta(\mathbf{z}) \sim \mathcal{N}(0, \mathbf{I})$ $\quad p_\theta(\mathbf{x} \mid \mathbf{z}) =$ point mass on $G_\theta(\mathbf{z})$

Model:



MLE?: $\hat{\theta} = \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)})$ NOPE!

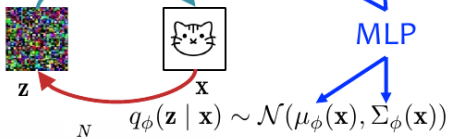
$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{x} \mid \mathbf{z})\, p_\theta(\mathbf{z})\, d\mathbf{z}$

Learn: minimax game

VAEs

$p_\theta(\mathbf{z}) \sim \mathcal{N}(0, \mathbf{I})$ $\quad p_\theta(\mathbf{x} \mid \mathbf{z}) \sim \mathcal{N}(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z}))$

Model:



$h = \tanh(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1)$
$\mu_\theta(\mathbf{z}) = \mathbf{W}_2 h + \mathbf{b}_2$
$\Sigma_\theta(\mathbf{z}) = (\mathbf{W}_3 h + \mathbf{b}_3)\mathbf{I}$

MLE?: $\hat{\theta} = \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)})$ NOPE!

$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{x} \mid \mathbf{z})\, p_\theta(\mathbf{z})\, d\mathbf{z}$

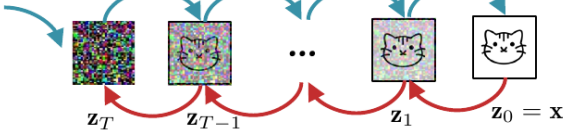
Learn: $\theta, \phi = \arg\min_{\theta,\phi} \mathrm{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p_\theta(\mathbf{z} \mid \mathbf{x}))$

where $p_\theta(\mathbf{z} \mid \mathbf{x})$ is true posterior of $p_\theta(\mathbf{x} \mid \mathbf{z})\, p_\theta(\mathbf{z})$

Diffusion

$p_\theta(\mathbf{z}) \sim \mathcal{N}(0, \mathbf{I})$ $\quad p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{z}_t), \Sigma_\theta(\mathbf{z}_t))$

Model:



Learn: $\theta = \arg\min_\theta \mathrm{KL}(q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}) \,\|\, p_\theta(\mathbf{z}_{1:T} \mid \mathbf{x}))$, $\quad \phi$ chosen ahead of time

where $p_\theta(\mathbf{z}_{1:T} \mid \mathbf{x})$ is true posterior of $p_\theta(\mathbf{x} \mid \mathbf{z}_1) \cdots p_\theta(\mathbf{z}_{T-1} \mid \mathbf{z}_T)\, p_\theta(\mathbf{z}_T)$

Slide credit: CMU MLD, Matt Gormley

Diffusion Model Analogy

A sculptor removes material until the figure appears.



Slide credit: Mohammad Salameh

Diffusion Model Analogy

A sculptor removes material until the figure appears.



A diffusion model removes a little noise at each step until an image appears.



Slide credit: Mohammad Salameh. Image credit: Bilal Taha

Diffusion as a Latent-Variable Model

The observed variable is the clean sample x_0. The noisy states $x_1, \dots, x_T$ are latent variables during generation.

Fixed forward process

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

Learned generative process

$$p_\theta(x_{0:T}) = \underbrace{p(x_T)}_{\mathcal{N}(0,I)} \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$$

The forward chain supplies tractable latent variables for training. The reverse chain supplies ancestral sampling from x_T to x_0

Slide credit: Bilal Taha

Diffusion Models

1 **Forward process**

add noise to data

2 **Reverse process**

learn to denoise

3 **Training objective**

from Hierarchical VAE to L2 loss

4 **Noise Conditional Network**

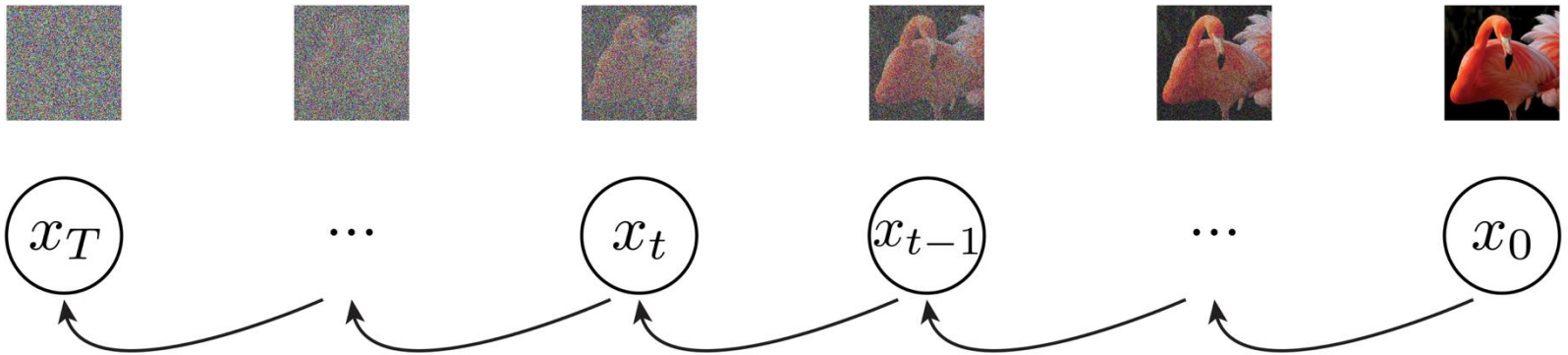
represent distributions

Slide credit: Bilal Taha

… in a nutshell

noise

data



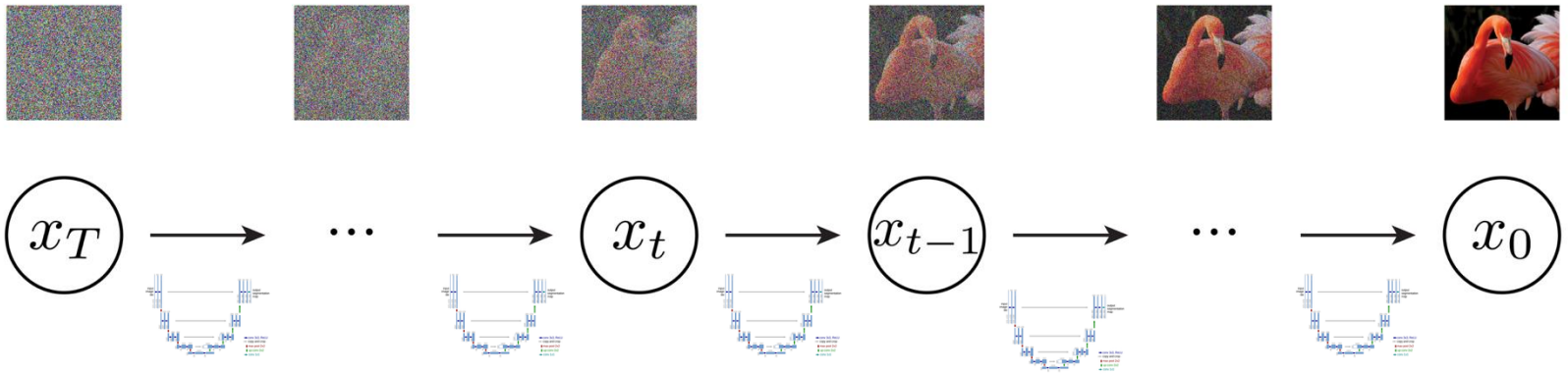
Forward process: add noise

Slide credit: Bilal Taha

… in a nutshell

noise

data



Reverse process: denoise

Slide credit: Bilal Taha. Image credit: Ronneberger et al. (2015)

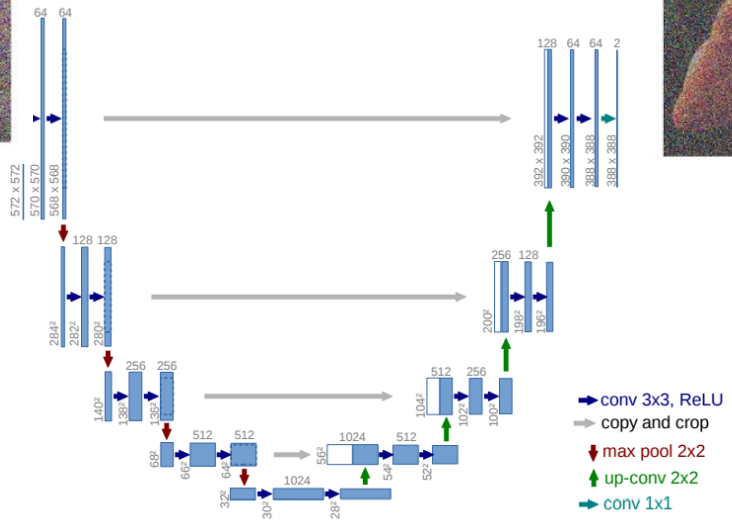
… in a nutshell

noise

data

Input noisy image







output a slightly less noisy image

x_T … x_0

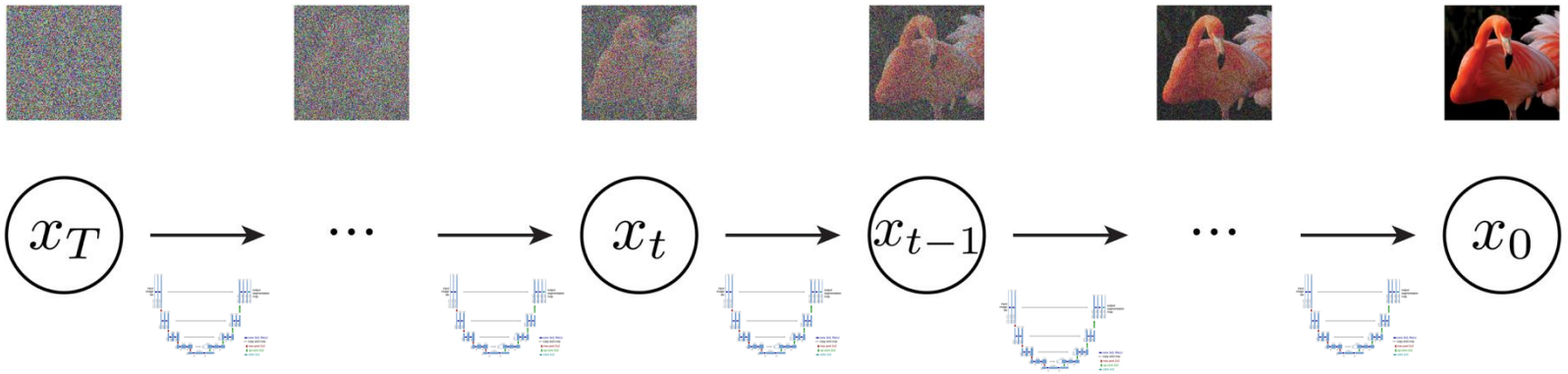
Reverse process: denoise

Slide credit: Bilal Taha. Image credit: Ronneberger et al. (2015)

… in a nutshell

noise

data

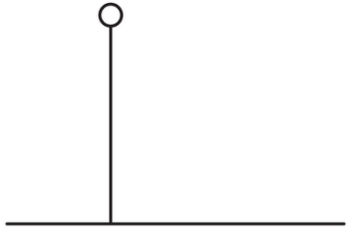


Reverse process: denoise

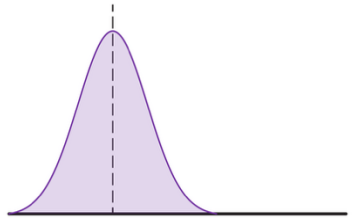
Slide credit: Bilal Taha. Image credit: Ronneberger et al. (2015)

What is noise?

- Adding Gaussian noise ⇔ sampling $x \sim \mathcal{N}(x \mid x_0, \sigma)$



$p(x) = \delta(x - x_0)$



$p(x) = \mathcal{N}(x \mid x_0, \sigma)$

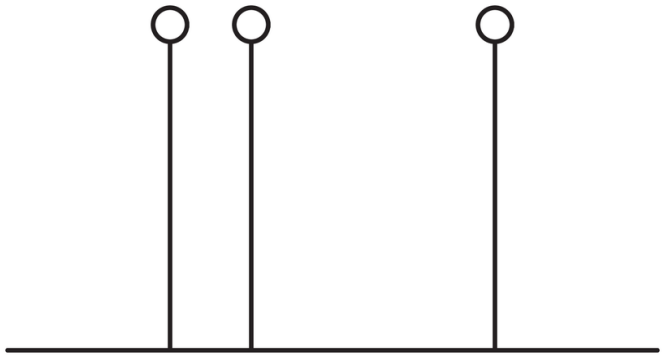




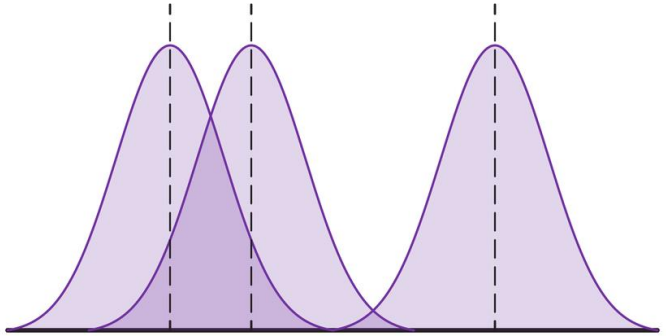
Slide credit: Bilal Taha

What is noise?

- Adding Gaussian noise ⇔ sampling $x \sim \mathcal{N}(x \mid x_0, \sigma)$



$p_{\text{data}}(x)$



$p_{\text{data}}(x) * \mathcal{N}(x \mid 0, \sigma)$

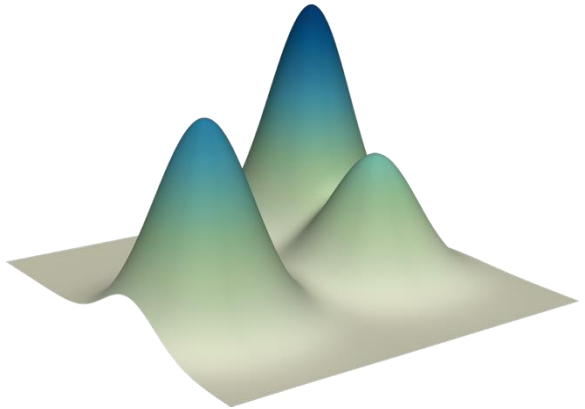
$*$ denotes a convolution (of the pdf)

Mathematically, the noisy distribution is p_{data} **convolved** with a Gaussian: adding noise to samples blurs the density

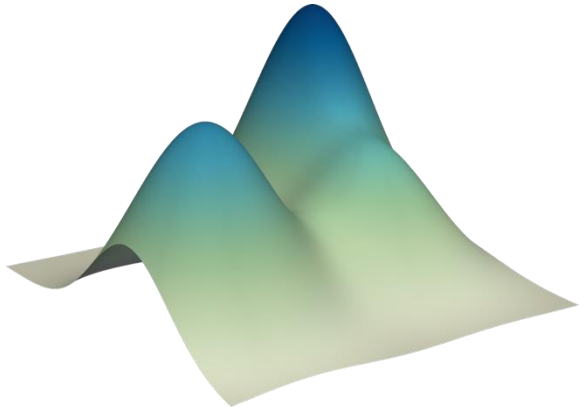
Slide credit: Bilal Taha

What is noise?

- Adding Gaussian noise ⇔ sampling $x \sim \mathcal{N}(x \mid x_0, \sigma)$



$p_{\text{data}}(x)$

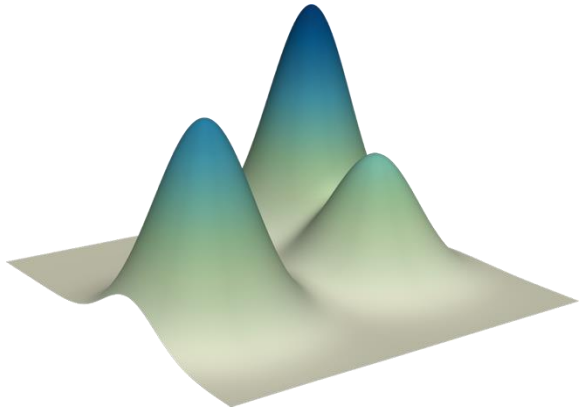


$p_{\text{data}}(x) * \mathcal{N}(x \mid 0, \sigma)$

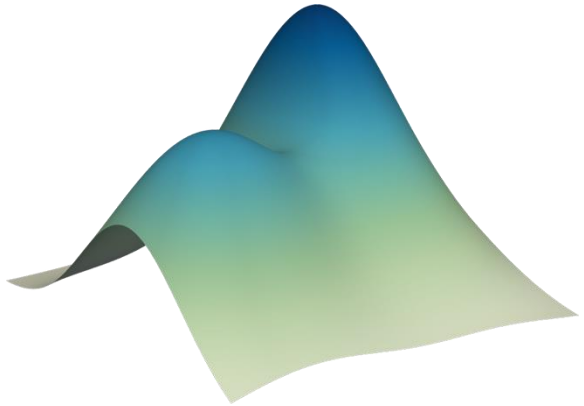
Slide credit: Bilal Taha

What is noise?

- Adding Gaussian noise ⇔ sampling $x \sim \mathcal{N}(x \mid x_0, \sigma)$



$p_{\text{data}}(x)$

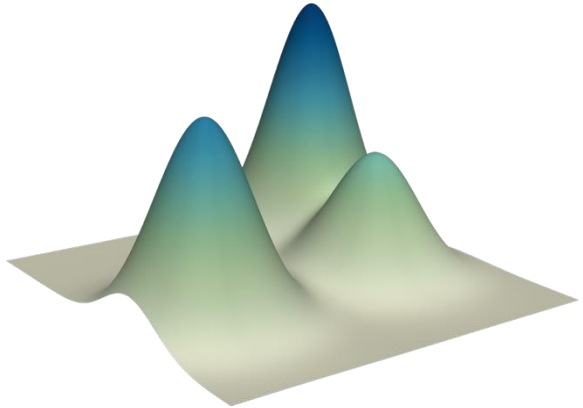


$p_{\text{data}}(x) * \mathcal{N}(x \mid 0, \sigma)$

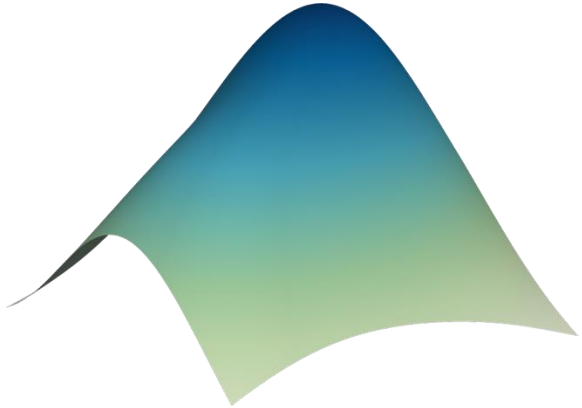
Slide credit: Bilal Taha

What is noise?

- Adding Gaussian noise ⇔ sampling $x \sim \mathcal{N}(x \mid x_0, \sigma)$



$p_{\text{data}}(x)$

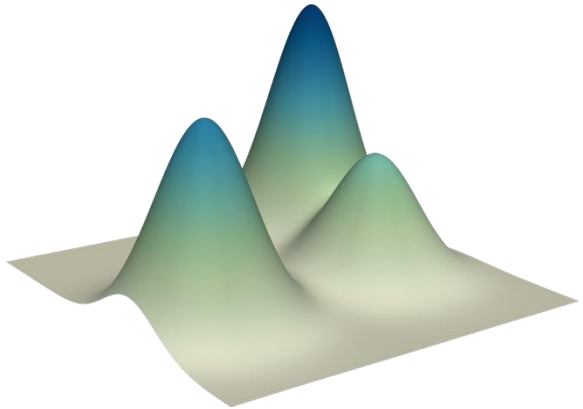


$p_{\text{data}}(x) * \mathcal{N}(x \mid 0, \sigma)$

Slide credit: Bilal Taha

What is noise?

- Adding Gaussian noise ⇔ sampling $x \sim \mathcal{N}(x \mid x_0, \sigma)$



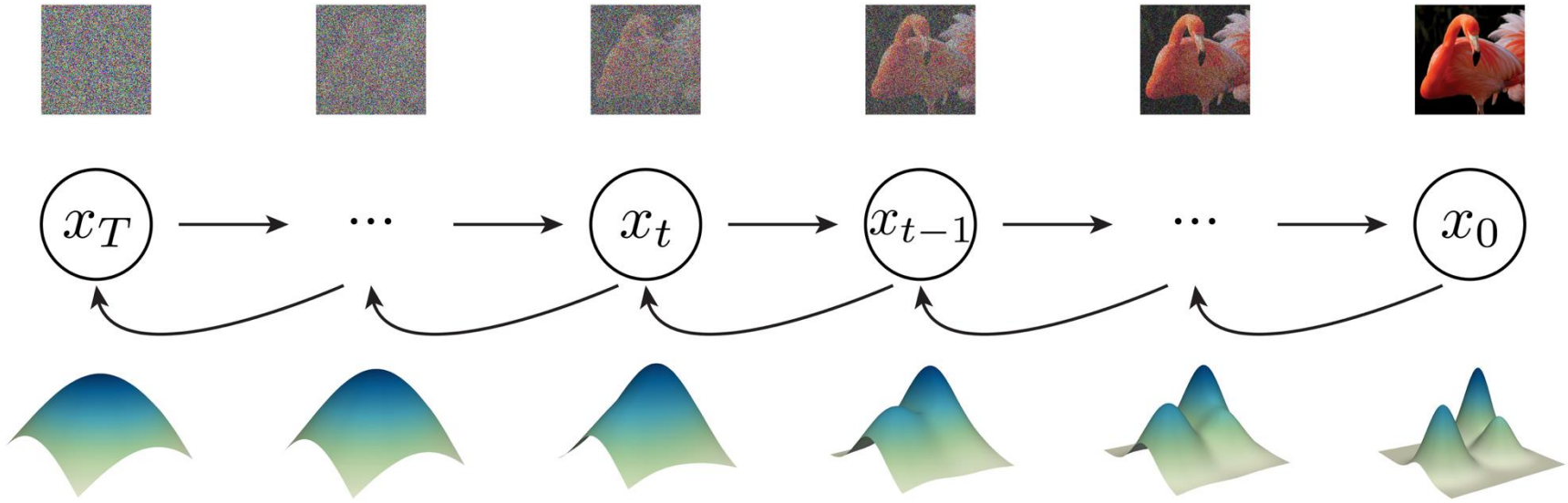
$p_{\text{data}}(x)$



$p_{\text{data}}(x) * \mathcal{N}(x \mid 0, \sigma)$

Slide credit: Bilal Taha

A divide and conquer problem



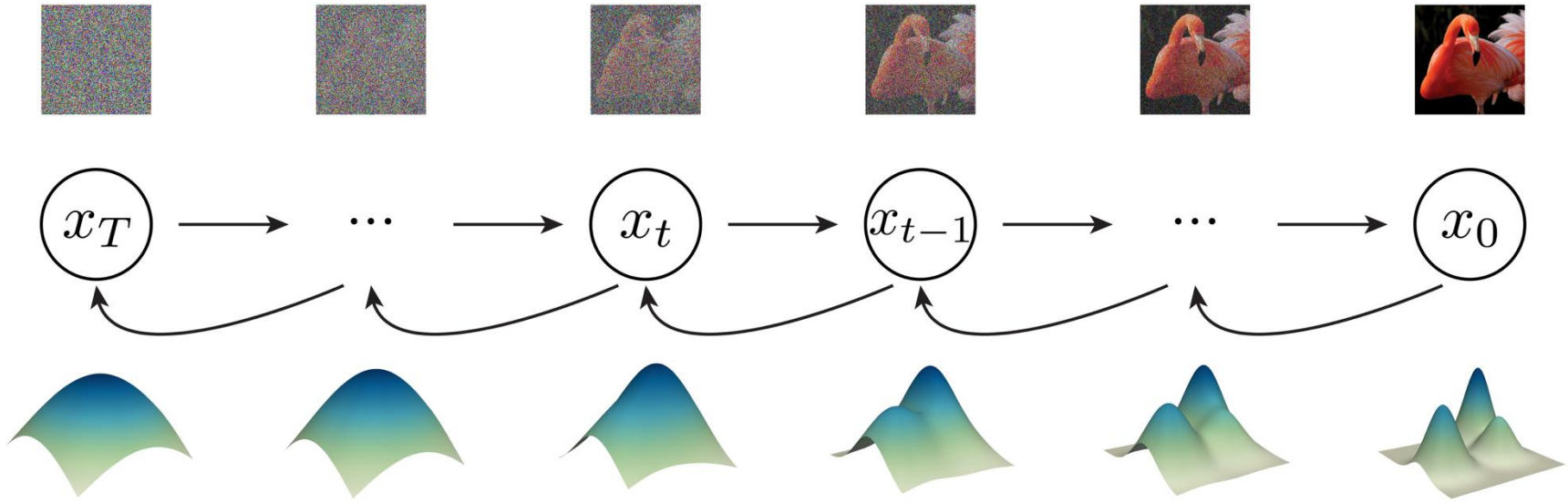
noise distribution

Mapping noise straight to data is hard.
Mapping one landscape to an *almost identical* one is easy.
Turn one hard problem into a thousand easy ones.

data distribution

Slide credit: Bilal Taha

A divide and conquer problem



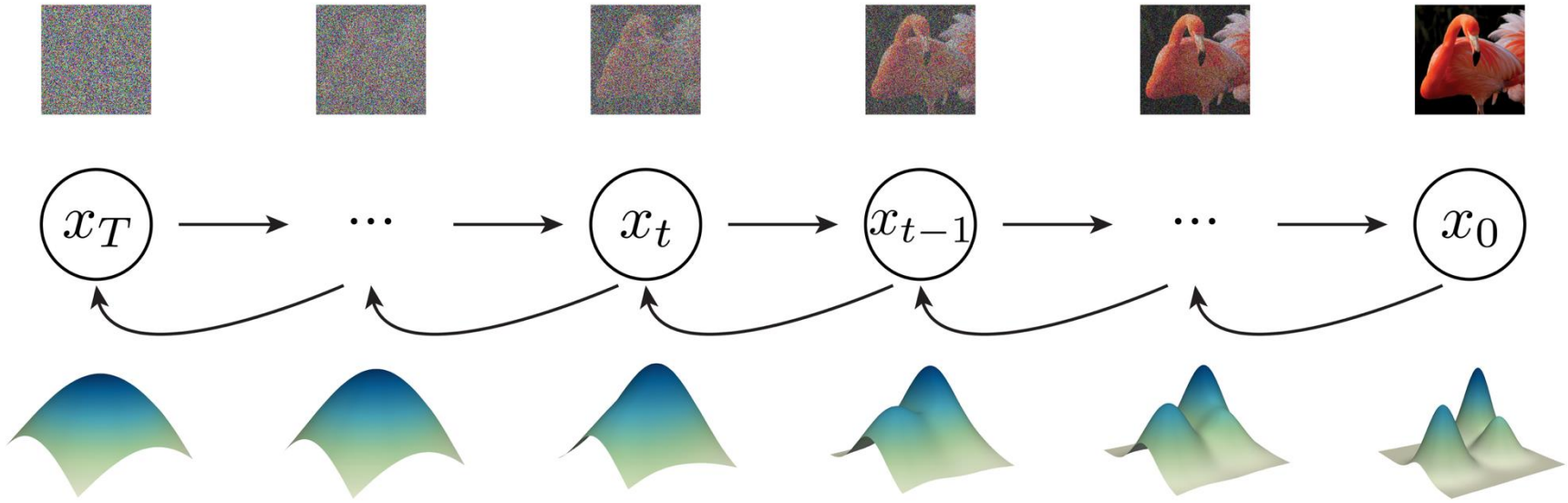
noise distribution

Why do we want neighboring distributions to be close?

data distribution

Slide credit: Bilal Taha

A divide and conquer problem



noise distribution

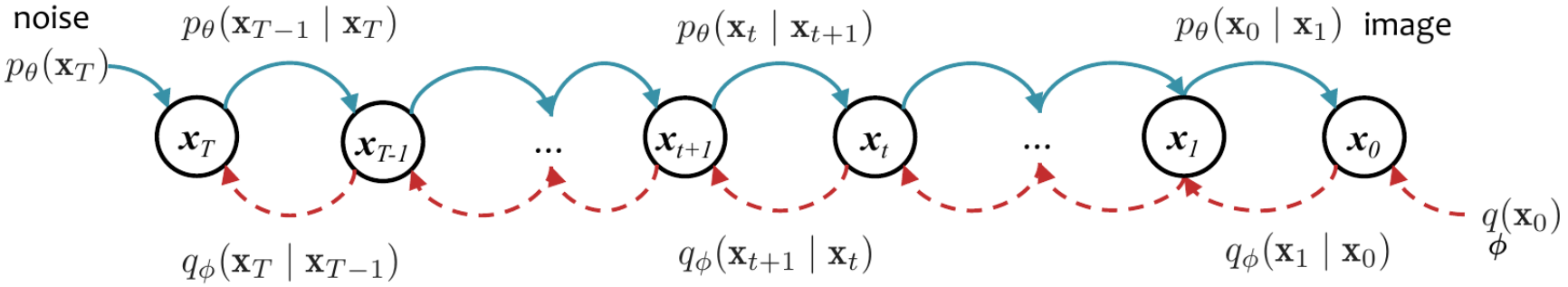
Why do we want neighboring distributions to be close?

If each step changes little, the reverse step is nearly Gaussian, so a simple Gaussian model per step is enough.

data distribution

Slide credit: Bilal Taha

Denoising Diffusion Probabilistic Model (DDPM)



Forward process:

$$q_\phi(x_{0:T}) = q(x_0) \prod_{t=1}^{T} q_\phi(x_t \mid x_{t-1})$$

$$q(x_0) = \text{data distribution}$$

$$q_\phi(x_t \mid x_{t-1}) \sim \mathcal{N}\left(\sqrt{\alpha_t}\, x_{t-1}, (1-\alpha_t)\mathbf{I}\right)$$

(Learned) Reverse process:

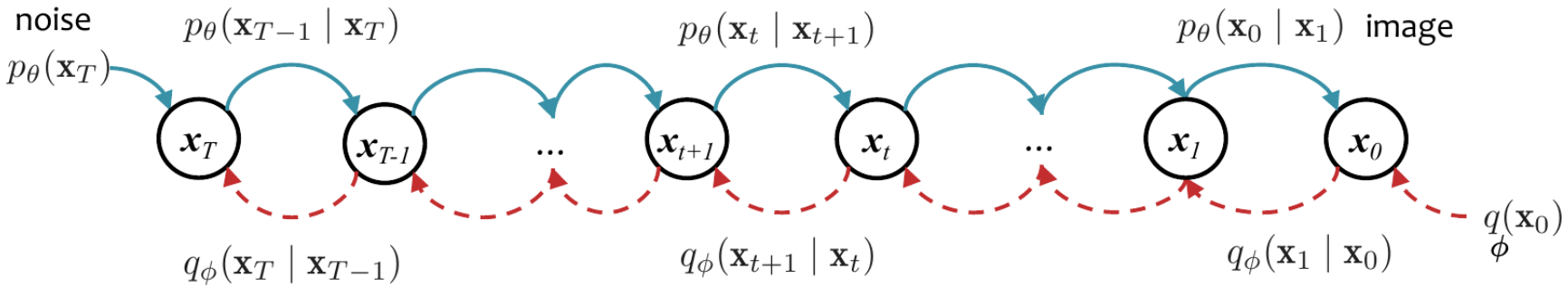
$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$$

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

$$p_\theta(x_{t-1} \mid x_t) \sim \mathcal{N}\left(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right)$$

Slide credit: CMU MLD, Matt Gormley

Denoising Diffusion Probabilistic Model (DDPM)



Compared with the VAE, which arrows did we learn there, and which do we learn here?

a. Blue arrows
b. Red arrows
c. Both
d. None

Slide credit: Mohammad Salameh. Image credit: CMU MLD, Matt Gormley

Diffusion Models

1 **Forward process**

add noise to data

2 **Reverse process**

learn to denoise

3 **Training objective**

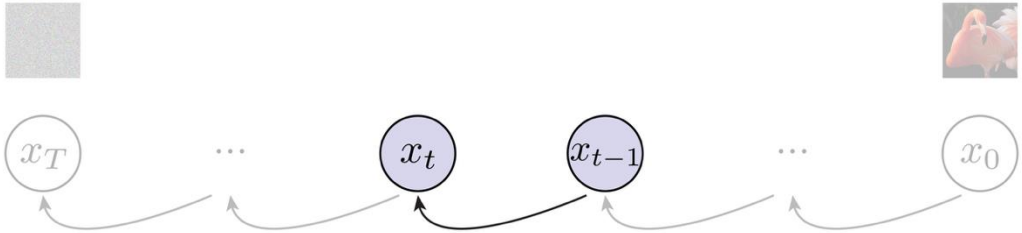
from Hierarchical VAE to L2 loss

4 **Noise Conditional Network**

represent distributions

Slide credit: Bilal Taha

Forward Process

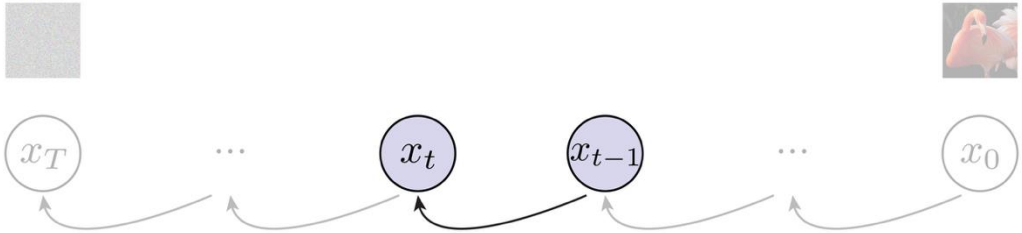


The forward process q adds a little Gaussian noise at each of T steps. It has **no learned parameters**.

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

Slide credit: Bilal Taha

Forward Process



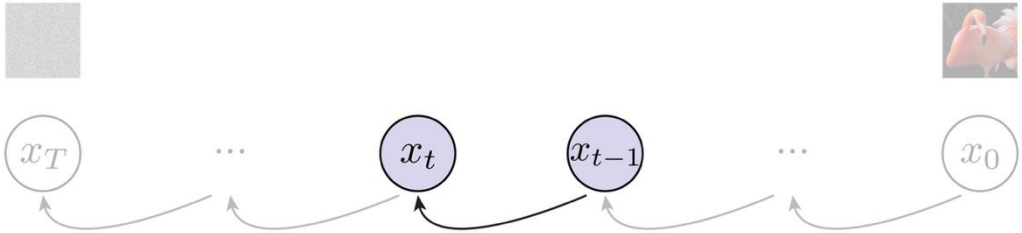
the image after one more step, slightly noisier

the image after $t-1$ noising steps (x_0 is the clean image)

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

Slide credit: Bilal Taha

Forward Process



shrinks the previous image a little, so the signal fades.

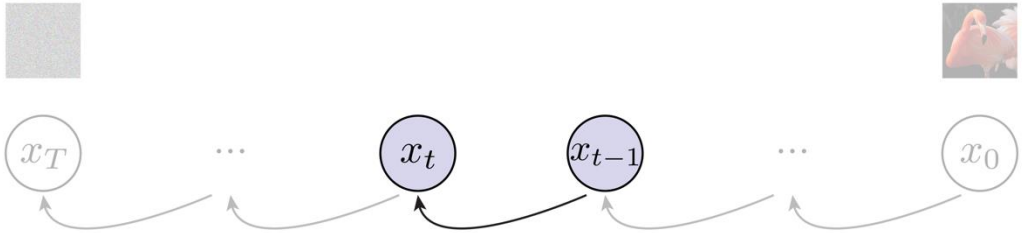
β_t denotes the fraction of the image swapped for noise at step. It is the noise variance added at step t. It's a small number from the schedule, e.g. 0.0001 to 0.02 in DDPM, set by hand, not learned. $\sqrt{\beta_t}$ scales that noise to standard deviation $\sqrt{\beta_t}$

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

Variance preserving. If x_{t-1} has unit variance, then $Var(x_t) = (1-\beta_t) + \beta_t = 1$. The signal fades and the noise grows, but pixel values never blow up.
The schedule β_t is chosen by hand. DDPM uses T = 1,000 and β rising linearly from 0.0001 to 0.02. Small early steps destroy fine detail gently.

Slide credit: Bilal Taha

Forward Process



Mean of x_t | *std* of x_t

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

Equivalently, as a conditional distribution:

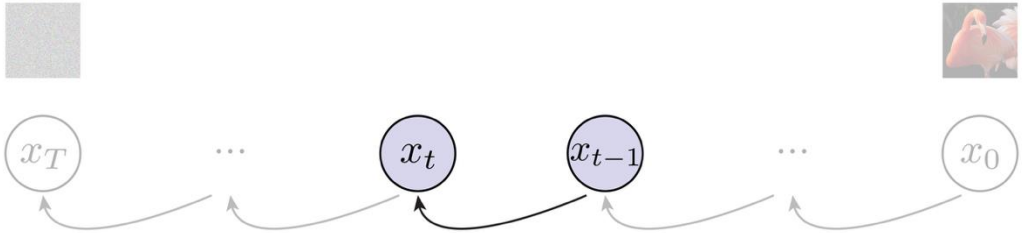
$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(x_t \mid \sqrt{1-\beta_t}\, x_{t-1}, \beta_t \mathbf{I}\right)$$

This is the reparameterization trick from the VAE lecture, written the other way

The covariance is $\beta_t\mathbf{I}$. Every pixel gets its own independent noise, and the noise has exactly the dimension of the image. Nothing is compressed.

Slide credit: Bilal Taha

Forward Process: From β_t to $\bar{\alpha}_t$



$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(x_t \mid \sqrt{1-\beta_t}\, x_{t-1}, \beta_t \mathbf{I}\right)$$

To get x_{500} for training, do we really have to run 500 noising steps?

Slide credit: Bilal Taha

Forward Process: From β_t to $\bar{\alpha}_t$

1. One step adds variance β_t

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I)$$

2. Define retained signal

$$\alpha_t = 1 - \beta_t \implies x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \epsilon_t$$

- β is the fraction replaced with the noise
- $\alpha = 1 - \beta$ is the fraction of image kept

Slide credit: Bilal Taha

Forward Process: From β_t to $\bar{\alpha}_t$

1. One step adds variance β_t

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I)$$

2. Define retained signal

$$\alpha_t = 1 - \beta_t \implies x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \epsilon_t$$

3. Substitute the previous step

$$x_{t-1} = \sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_{t-1}}\, \epsilon_{t-1}$$

$$\begin{aligned} x_t &= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_{t-1}}\, \epsilon_{t-1}\right) + \sqrt{1-\alpha_t}\, \epsilon_t \\ &= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \underbrace{\sqrt{\alpha_t(1-\alpha_{t-1})}\, \epsilon_{t-1} + \sqrt{1-\alpha_t}\, \epsilon_t}_{\text{two independent Gaussians}} \end{aligned}$$

Slide credit: Bilal Taha

Forward Process: From β_t to $\bar{\alpha}_t$

1. One step adds variance β_t

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I)$$

2. Define retained signal

$$\alpha_t = 1 - \beta_t \implies x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \epsilon_t$$

3. Substitute the previous step

$$x_{t-1} = \sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_{t-1}}\, \epsilon_{t-1}$$

$$\begin{aligned} x_t &= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_{t-1}}\, \epsilon_{t-1}\right) + \sqrt{1-\alpha_t}\, \epsilon_t \\ &= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \underbrace{\sqrt{\alpha_t(1-\alpha_{t-1})}\, \epsilon_{t-1} + \sqrt{1-\alpha_t}\, \epsilon_t}_{\text{two independent Gaussians}} \end{aligned}$$

Gaussian closure property

If $\epsilon_1, \epsilon_2 \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, I)$, then

$$a\epsilon_1 + b\epsilon_2 \sim \mathcal{N}\left(0, (a^2+b^2)I\right)$$

The two noise terms therefore have total variance

$$\alpha_t(1-\alpha_{t-1}) + (1-\alpha_t) = 1 - \alpha_t\alpha_{t-1}$$

After t steps $\quad \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$

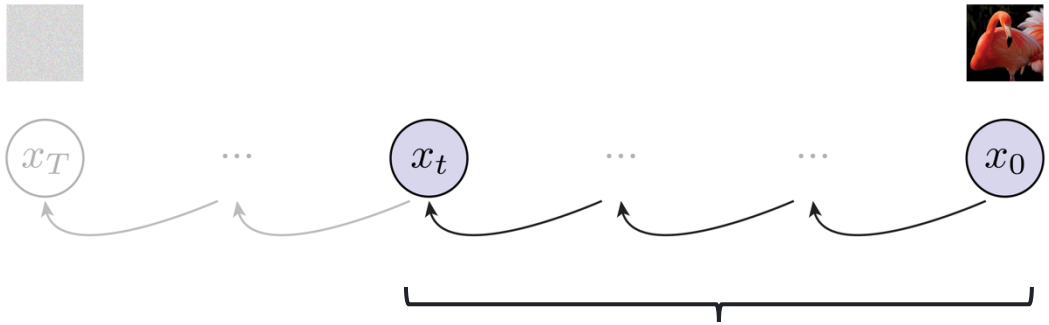
$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

β_t: how much noise each step adds

$\bar{\alpha}_t$: cumulative retained signal after t steps

Slide credit: Bilal Taha

Forward Process



- Sampling without simulation
- x_t from x_0 in closed form

$$q(x_t \mid x_0) = \mathcal{N}\left(x_t \mid \sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)\mathbf{I}\right)$$

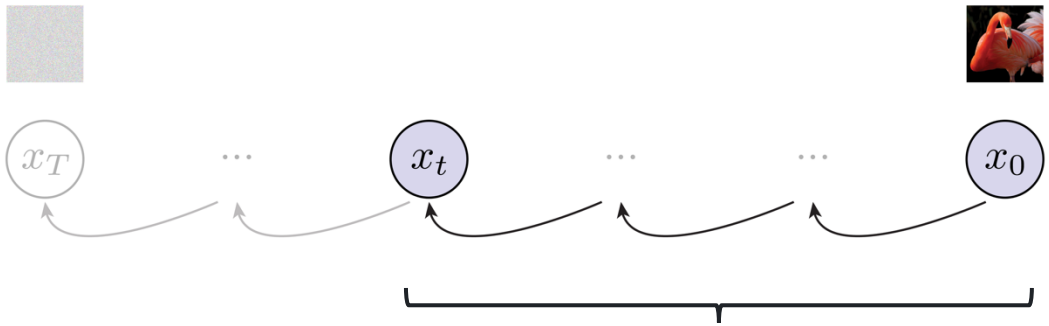
$$\alpha_t := 1 - \beta_t, \qquad \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

Coefficients are given by β.

Why will this matter for training?

Slide credit: Bilal Taha

Forward Process



- Sampling without simulation
- x_t from x_0 in closed form

$$q(x_t \mid x_0) = \mathcal{N}\left(x_t \mid \sqrt{\bar{\alpha}_t}\, x_0, (1 - \bar{\alpha}_t)\mathbf{I}\right)$$

$$\alpha_t := 1 - \beta_t, \qquad \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

Coefficients are given by β.

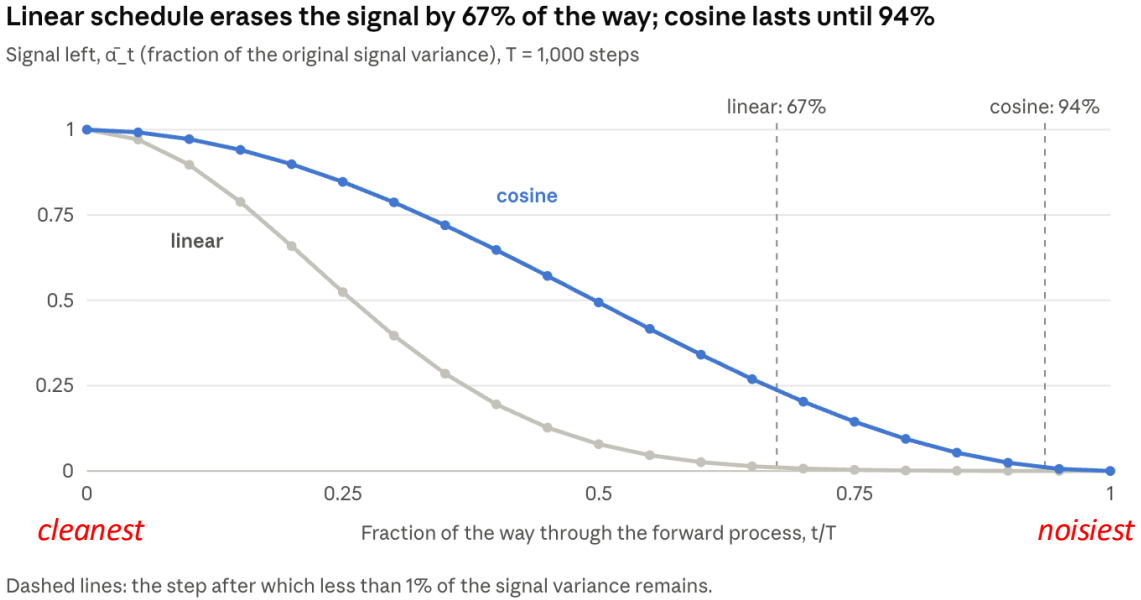
Why will this matter for training?
We can train on a random step t without running the chain.
Each training example costs one line of arithmetic plus one network call.

Slide credit: Bilal Taha

Forward Process: Noise Schedule

The schedule decides how fast $\bar{\alpha}_t$ falls from 1 (clean) to almost 0 (pure noise).

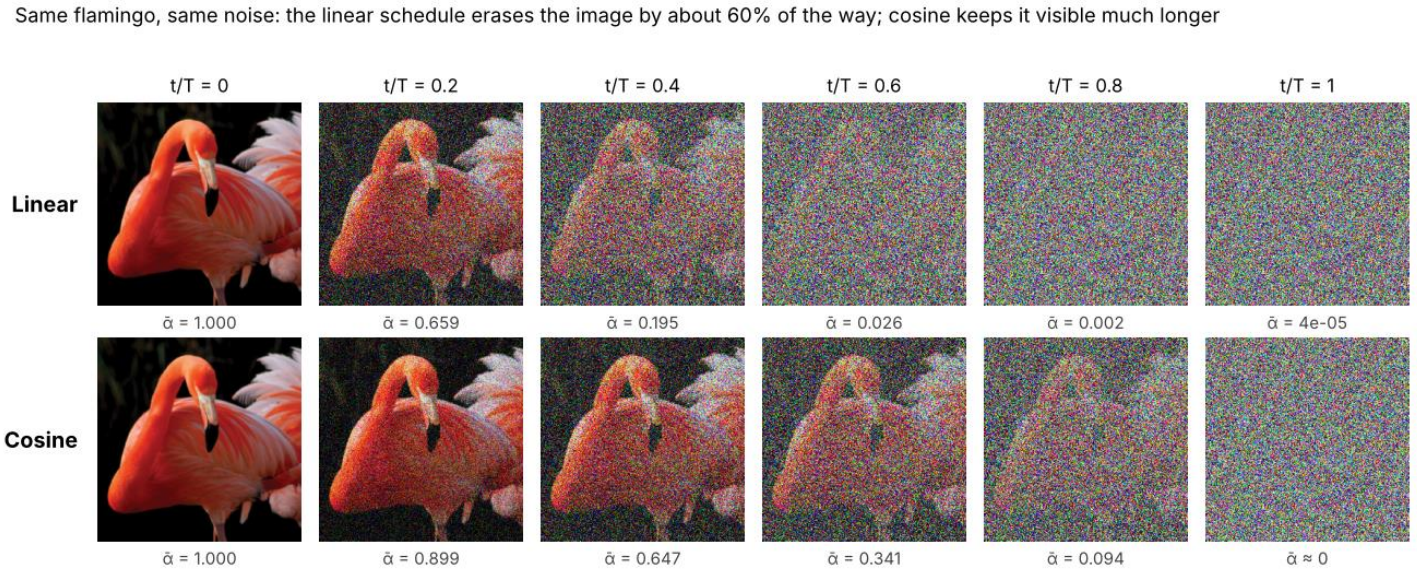
With the DDPM linear schedule, $\bar{\alpha}_T$ is about 4×10^{-5}, so $\sqrt{\bar{\alpha}_T} \approx 0.006$: less than 1% of the original signal's amplitude survives.



The **linear** schedule reaches nearly zero signal well before the last step, so its final steps only turn noise into slightly different noise.
The **cosine** schedule falls more evenly and spends more steps where the image is partly visible.

Slide credit: Bilal Taha

Forward Process: Noise Schedule

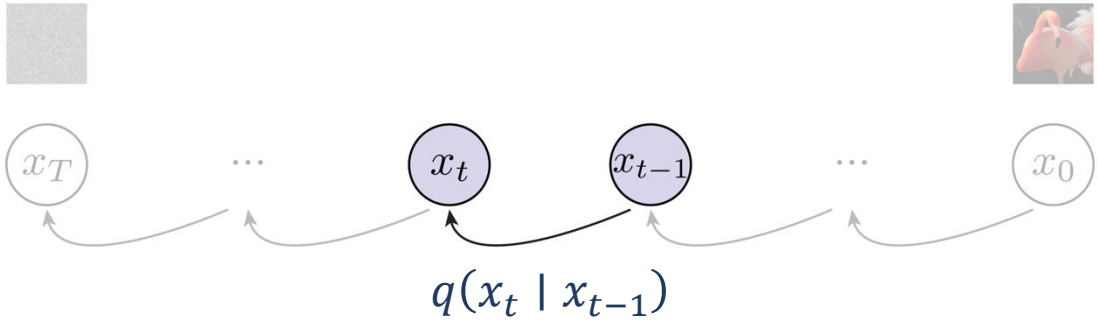


The schedule involves a trade-off:

- **Steps too large:** each reverse step must undo a big change, and the true reverse step is no longer close to Gaussian.
- **Steps too small:** x_T still contains signal, so starting generation from $N(0, I)$ is wrong, unless T is made very large.

Slide credit: Mohammad Salameh

Forward Process



Three takeaways

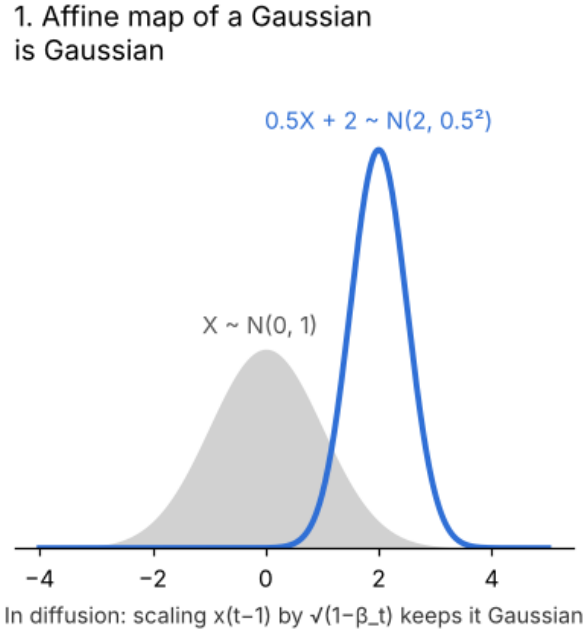
- pre-defined conditional distributions
- Gaussian with controllable mean/std
- **divide** and conquer

Slide credit: Bilal Taha

Gaussian Identities Used by Diffusion

Affine transformations preserve Gaussianity

$$X \sim \mathcal{N}(\mu, \Sigma) \implies AX + b \sim \mathcal{N}(A\mu + b, A\Sigma A^\top)$$



Scaling and shifting $X \sim N(0,1)$ gives another Gaussian, $N(2,0.5^2)$

Slide credit: Bilal Taha

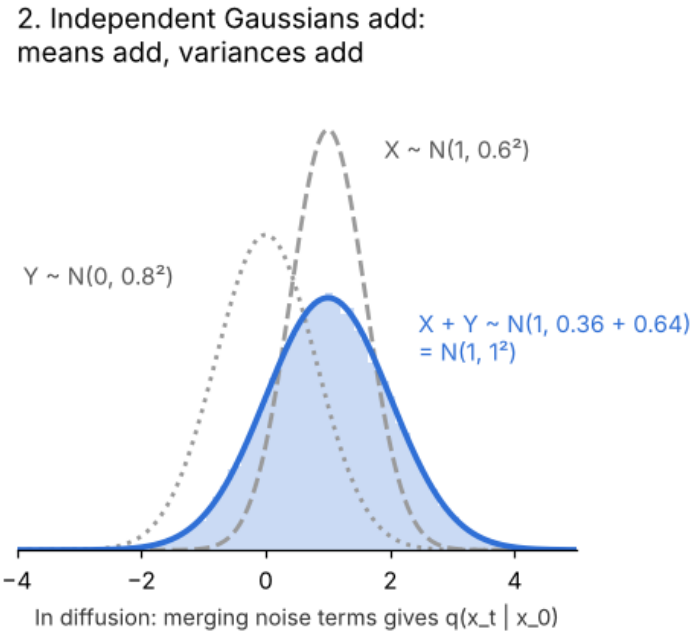
Gaussian Identities Used by Diffusion

Affine transformations preserve Gaussianity

$$X \sim \mathcal{N}(\mu, \Sigma) \implies AX + b \sim \mathcal{N}(A\mu + b, A\Sigma A^\top)$$

Independent Gaussian terms remain Gaussian when added

$$X \sim \mathcal{N}(\mu_X, \Sigma_X),\ Y \sim \mathcal{N}(\mu_Y, \Sigma_Y) \implies$$
$$X + Y \sim \mathcal{N}(\mu_X + \mu_Y, \Sigma_X + \Sigma_Y)$$



Adding independent $X \sim N(1,0.6^2)$ and $Y \sim N(0,0.8^2)$ gives $N(1,1^2)$. The shaded histogram of 40,000 sampled sums matches that curve.

Slide credit: Bilal Taha

Gaussian Identities Used by Diffusion

Affine transformations preserve Gaussianity

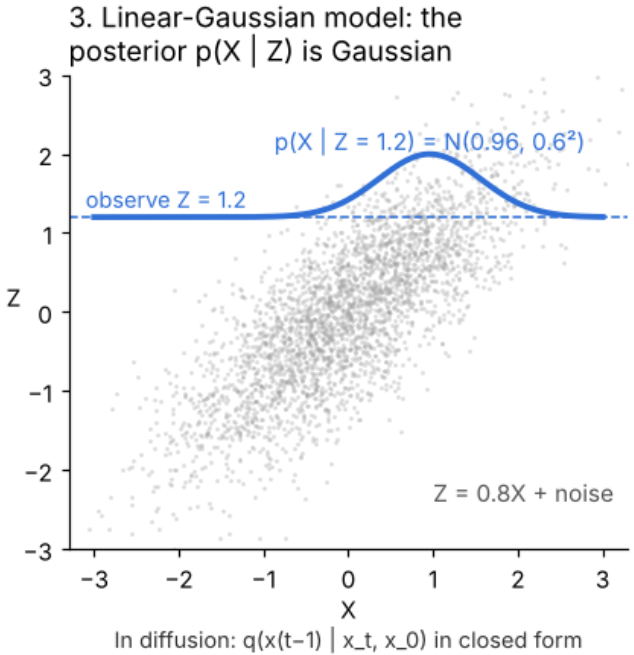
$$X \sim \mathcal{N}(\mu, \Sigma) \implies AX + b \sim \mathcal{N}(A\mu + b, A\Sigma A^\top)$$

Independent Gaussian terms remain Gaussian when added

$$X \sim \mathcal{N}(\mu_X, \Sigma_X),\ Y \sim \mathcal{N}(\mu_Y, \Sigma_Y) \implies$$
$$X + Y \sim \mathcal{N}(\mu_X + \mu_Y, \Sigma_X + \Sigma_Y)$$

Linear-Gaussian models have Gaussian marginals and posteriors

$$Z \mid X \sim \mathcal{N}(AX + b, \Sigma_Z),\ X \text{ Gaussian} \implies$$
$$p(Z) \text{ and } p(X \mid Z) \text{ are Gaussian}$$



If $Z = 0.8X +$ Gaussian noise, then fixing $Z = 1.2$(the dashed line) leaves a Gaussian over X, $N(0.96, 0.6^2)$. In diffusion, x_{t-1} plays the role of X and x_t the role of Z.

Slide credit: Bilal Taha

Gaussian Identities Used by Diffusion

Affine transformations preserve Gaussianity

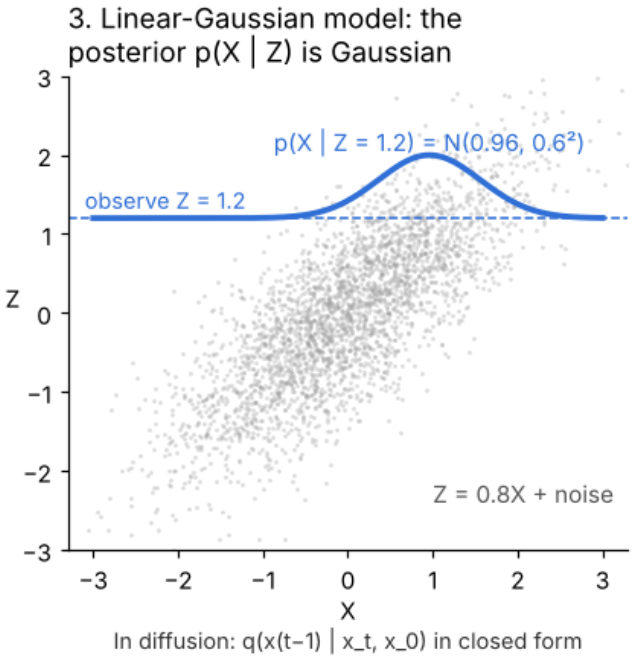
$$X \sim \mathcal{N}(\mu, \Sigma) \implies AX + b \sim \mathcal{N}(A\mu + b, A\Sigma A^\top)$$

Independent Gaussian terms remain Gaussian when added

$$X \sim \mathcal{N}(\mu_X, \Sigma_X),\ Y \sim \mathcal{N}(\mu_Y, \Sigma_Y) \implies$$
$$X + Y \sim \mathcal{N}(\mu_X + \mu_Y, \Sigma_X + \Sigma_Y)$$

Linear-Gaussian models have Gaussian marginals and posteriors

$$Z \mid X \sim \mathcal{N}(AX + b, \Sigma_Z),\ X \text{ Gaussian} \implies$$
$$p(Z) \text{ and } p(X \mid Z) \text{ are Gaussian}$$



These closure properties make both $q(x_t \mid x_0)$ and $q(x_{t-1} \mid x_t, x_0)$ available in closed form.

Slide credit: Bilal Taha

Diffusion Models

1 **Forward process**

add noise to data

2 **Reverse process**

learn to denoise

3 **Training objective**

from Hierarchical VAE to L2 loss

4 **Noise Conditional Network**

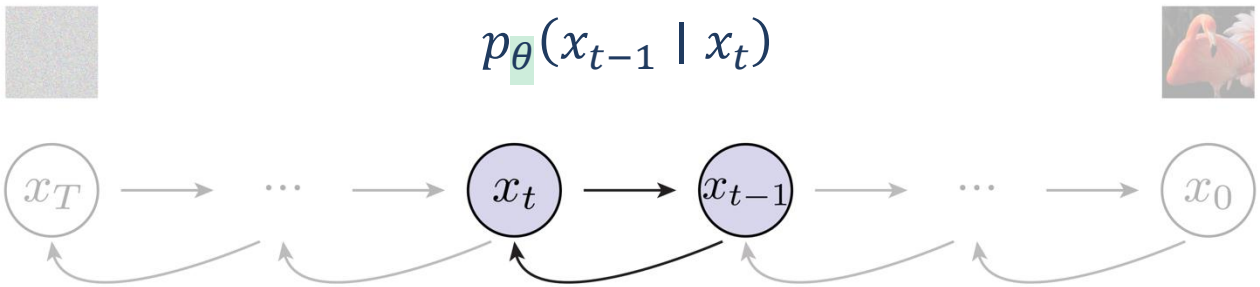
represent distributions

Slide credit: Bilal Taha

Reverse Process

parameterized by
a network

$$p_\theta(x_{t-1} \mid x_t)$$

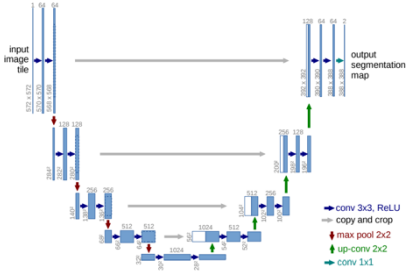


$$q(x_t \mid x_{t-1})$$

known, but not our target

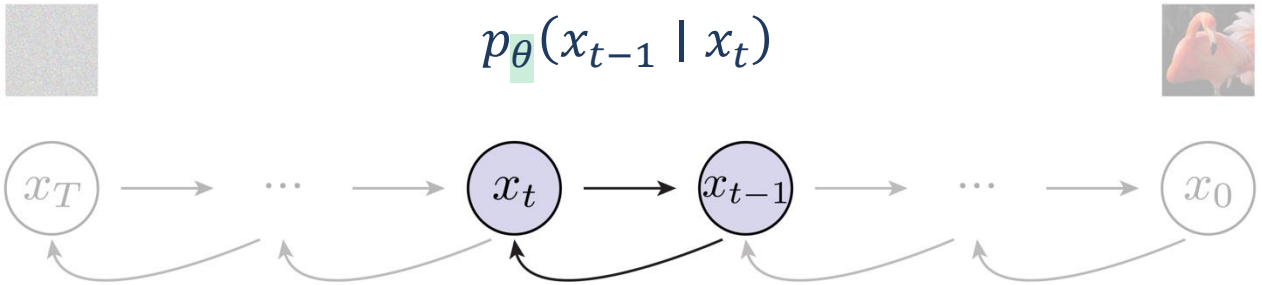
Slide credit: Bilal Taha

Reverse Process



parameterized by a network

$$p_\theta(x_{t-1} \mid x_t)$$

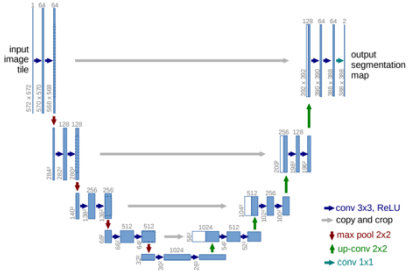


$$q(x_t \mid x_{t-1})$$

known, but not our target

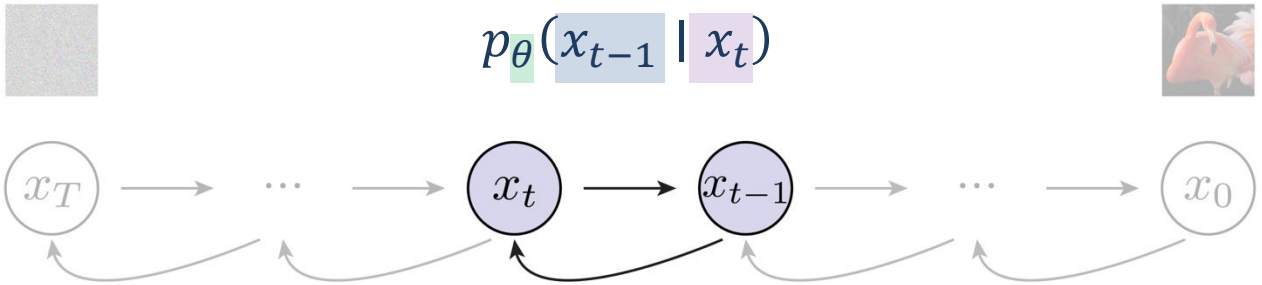
Slide credit: Bilal Taha. Image credit: Ronneberger et al. (2015)

Reverse Process



parameterized by a network

$$p_\theta(x_{t-1} \mid x_t)$$



$$q(x_t \mid x_{t-1})$$

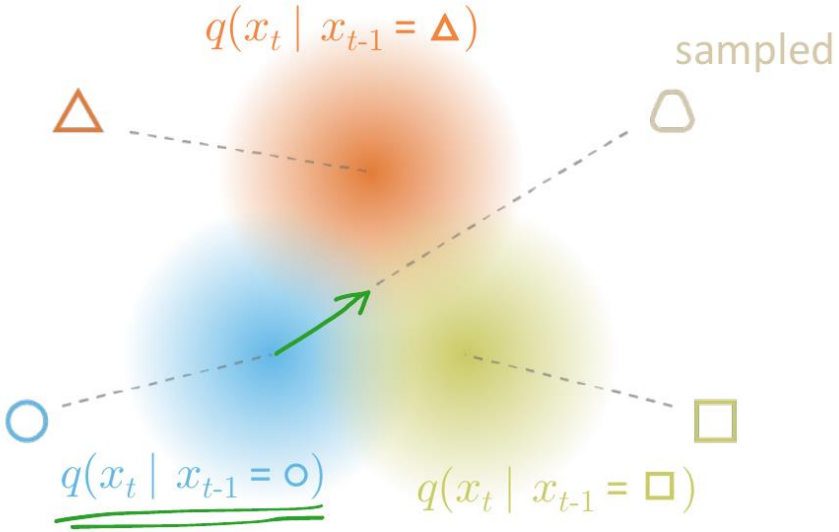
known, but not our target

Slide credit: Bilal Taha. Image credit: Ronneberger et al. (2015)

Why Are the Reverse Conditionals Unknown?

The forward step is known, so why not apply Bayes' rule? Because the reverse step depends on where the data is i,e where p_{data} puts its probability: which images are common and which are rare.

Each forward step sends a point into a fuzzy Gaussian blob around it.
Forward is easy: start at the circle and you land somewhere in the blue blob. Known and Gaussian



- known (Gaussian)

Figure adapted from: Joseph Rocca, "Understanding Variational Autoencoders (VAEs)", towardsdatascience.com

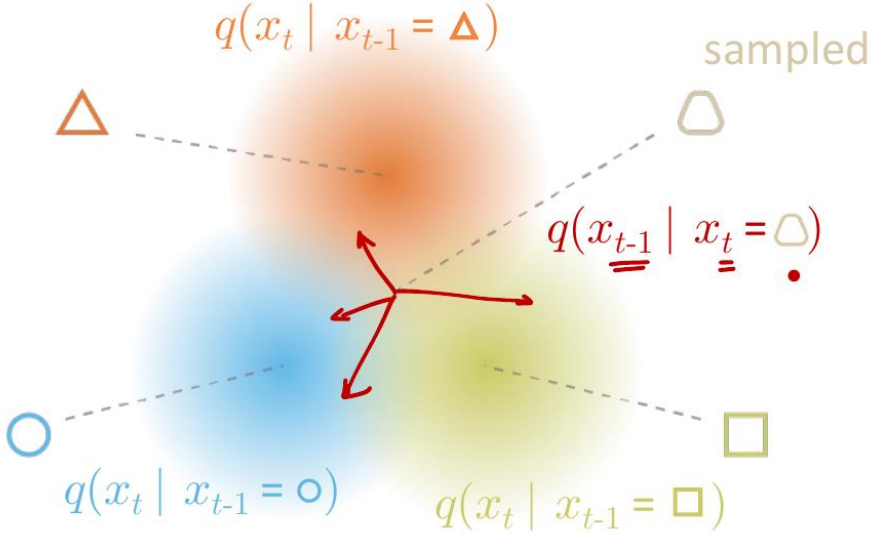
Slide credit: Bilal Taha

Why Are the Reverse Conditionals Unknown?

Now consider the sampled shape where the blobs overlap.
Where did it come from: circle, triangle or square?

It depends on how much data sits at each, which is exactly p_{data}, the thing we don't know.

So the true reverse step depends on the data distribution, and it can even be multimodal."



So why can we still use a Gaussian for each reverse step?

$q(x_{t-1} \mid x_t = \text{sampled})$ is unknown (depends on the data distribution)

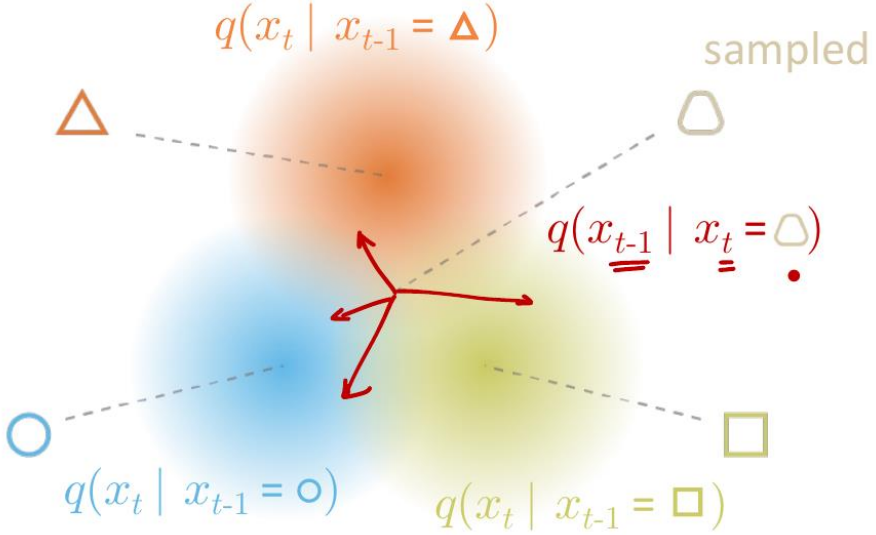
Slide credit: Bilal Taha

Why Are the Reverse Conditionals Unknown?

Now consider the sampled shape where the blobs overlap.
Where did it come from: circle, triangle or square?

It depends on how much data sits at each, which is exactly p_{data}, the thing we don't know.

So the true reverse step depends on the data distribution, and it can even be multimodal."



So why can we still use a Gaussian for each reverse step?

When the step is small, the blobs barely spread and overlap only slightly, so the true reverse step is nearly Gaussian. This is why we wanted small β_t.

$q(x_{t-1} \mid x_t = \text{sampled})$ is unknown (depends on the data distribution)

Slide credit: Bilal Taha

Exact Reverse Conditional

From a noisy sample x_t, the unconditional reverse step would require

$$q(x_{t-1} \mid x_t) = \int q(x_{t-1}, x_0 \mid x_t)\, dx_0 \qquad \text{(marginalize over } x_0\text{)}$$

$$= \int q(x_{t-1} \mid x_t, x_0)\, q(x_0 \mid x_t)\, dx_0 \qquad \text{(product rule)}$$

Slide credit: Bilal Taha

Exact Reverse Conditional

From a noisy sample x_t, the unconditional reverse step would require

$$q(x_{t-1} \mid x_t) = \int q(x_{t-1}, x_0 \mid x_t)\, dx_0 \qquad \text{(marginalize over } x_0\text{)}$$

$$= \int q(x_{t-1} \mid x_t, x_0)\, q(x_0 \mid x_t)\, dx_0 \qquad \text{(product rule)}$$

known Gaussian

unknown data posterior

This integral is intractable because the data distribution is unknown.

Slide credit: Bilal Taha

Exact Reverse Conditional

From a noisy sample x_t, the unconditional reverse step would require

$$q(x_{t-1} \mid x_t) = \int q(x_{t-1}, x_0 \mid x_t)\, dx_0 \qquad \text{(marginalize over } x_0\text{)}$$

$$= \int q(x_{t-1} \mid x_t, x_0)\, q(x_0 \mid x_t)\, dx_0 \qquad \text{(product rule)}$$

known Gaussian

unknown data posterior

This integral is intractable because the data distribution is unknown.

$$q(x_0 \mid x_t) = \frac{q(x_t|x_0)\, p_{\text{data}}(x_0)}{\int q(x_t|x_0')\, p_{\text{data}}(x_0')\, dx_0'}$$

$q(x_t|x_0)$ is known, but $p_{data}(x_0)$ is not, and the integral in the denominator runs over every possible clean image. So the reverse step averages an easy Gaussian over a posterior we ***can't*** compute.

Slide credit: Bilal Taha

Exact Reverse Conditional

From a noisy sample x_t, the unconditional reverse step would require

$$q(x_{t-1} \mid x_t) = \int q(x_{t-1}, x_0 \mid x_t)\, dx_0 \qquad \text{(marginalize over } x_0\text{)}$$

$$= \int q(x_{t-1} \mid x_t, x_0)\, q(x_0 \mid x_t)\, dx_0 \qquad \text{(product rule)}$$

known Gaussian

unknown data posterior

This integral is intractable because the data distribution is unknown.

$$q(x_0 \mid x_t) = \frac{q(x_t|x_0)\, p_{\text{data}}(x_0)}{\int q(x_t|x_0')\, p_{\text{data}}(x_0')\, dx_0'}$$

$q(x_t|x_0)$ is known, but $p_{data}(x_0)$ is not, and the integral in the denominator runs over every possible clean image. So the reverse step averages an easy Gaussian over a posterior we ***can't*** compute.

BUT…when β_t is small, each blob is narrow, and the true reverse step is close to Gaussian. That is why a Gaussian p_θ for each step is good enough.

Slide credit: Bilal Taha

Exact Reverse Conditional

From a noisy sample x_t, the unconditional reverse step would require

$$q(x_{t-1} \mid x_t) = \int q(x_{t-1}, x_0 \mid x_t)\, dx_0 \qquad \text{(marginalize over } x_0\text{)}$$

$$= \int q(x_{t-1} \mid x_t, x_0)\, q(x_0 \mid x_t)\, dx_0 \qquad \text{(product rule)}$$

known Gaussian

unknown data posterior

This integral is intractable because the data distribution is unknown.

During training, however, the clean sample x_0 is observed. Bayes' rule gives

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

Slide credit: Bilal Taha

Exact Reverse Conditional

From a noisy sample x_t, the unconditional reverse step would require

$$q(x_{t-1} \mid x_t) = \int q(x_{t-1}, x_0 \mid x_t)\, dx_0 \qquad \text{(marginalize over } x_0\text{)}$$

$$= \int q(x_{t-1} \mid x_t, x_0)\, q(x_0 \mid x_t)\, dx_0 \qquad \text{(product rule)}$$

unknown data posterior

This integral is intractable because the data distribution is unknown.

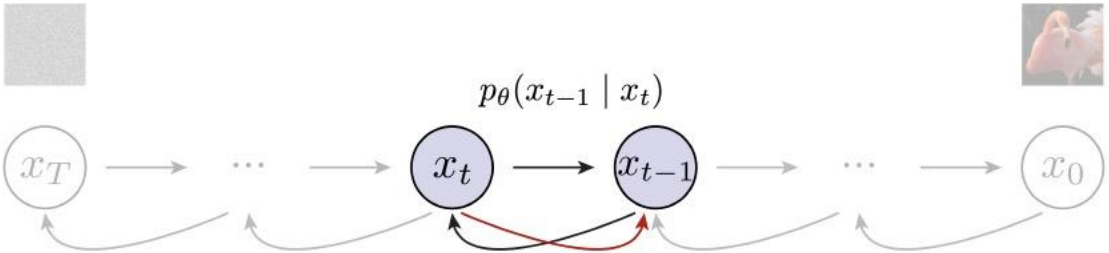
During training, however, the clean sample x_0 is observed.
Conditioning on it, Bayes' rule needs only forward-process terms, all Gaussian:

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

Since, every term on the right is Gaussian and known from the forward noise schedule, therefore $q(x_{t-1} \mid x_t, x_0)$ is available in closed form.

Slide credit: Bilal Taha

Reverse Process



~~$q(x_t \mid x_{t-1})$~~ *is known but points the wrong way: noisier, not cleaner.*

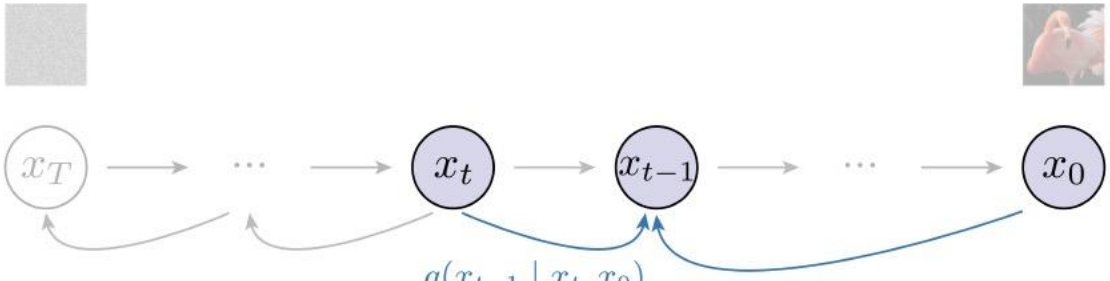
~~$q(x_{t-1} \mid x_t)$~~ *is the one that's unknown.*

$q(x_{t-1} \mid x_t, x_0)$

- known
- Gaussian

Slide credit: Bilal Taha

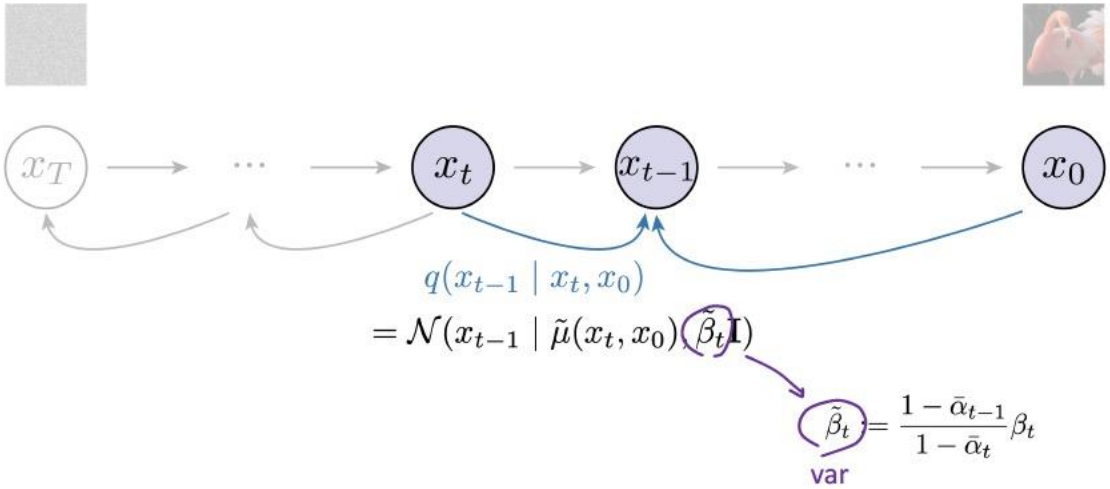
Reverse Process



$$q(x_{t-1} \mid x_t, x_0)$$
$$= \mathcal{N}(x_{t-1} \mid \tilde{\mu}(x_t, x_0), \tilde{\beta}_t \mathbf{I})$$

Slide credit: Bilal Taha

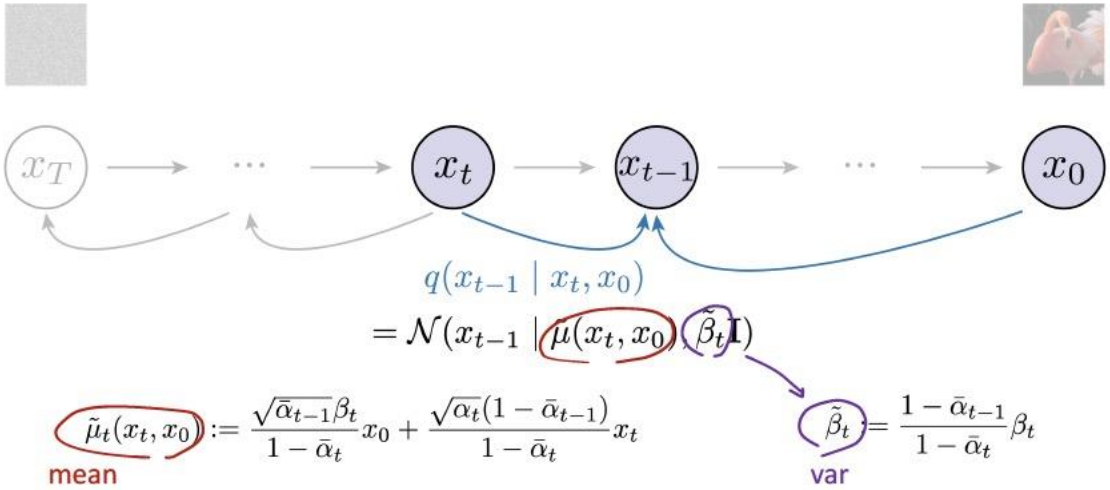
Reverse Process



β_t is only schedule numbers: no data, no network. We precompute it for every t before training starts

Slide credit: Bilal Taha

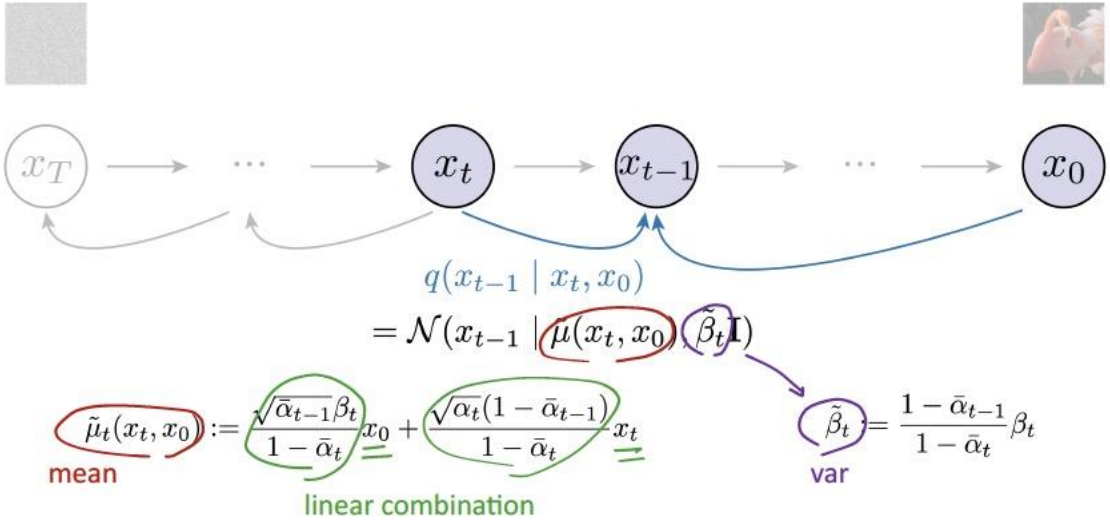
Reverse Process



The mean is a weighted combination of the clean image x_0 and the current noisy image x_t. The weights again come only from the schedule.

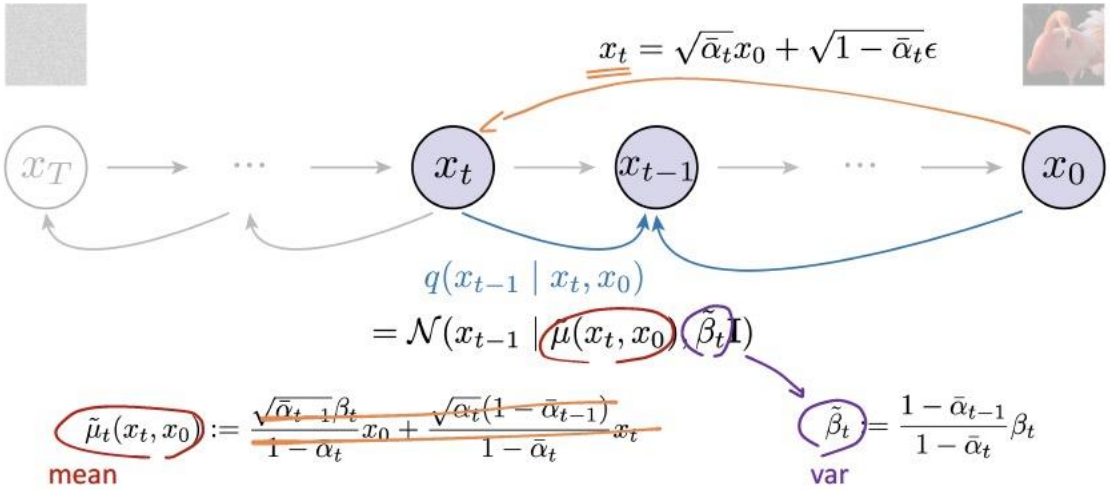
Slide credit: Bilal Taha

Reverse Process



Slide credit: Bilal Taha

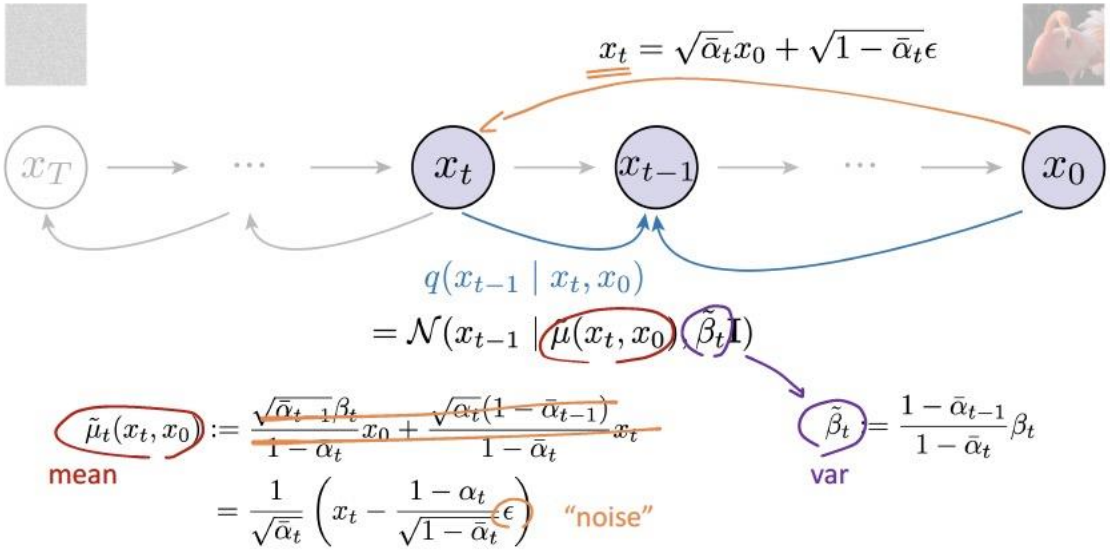
Reverse Process



Now the key move. We won't know x_0 at sampling time, but the forward formula links x_0, x_t and the noise:
$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{(1-\bar{\alpha}_t)}\varepsilon$. Solve for x_0 and substitute.

Slide credit: Bilal Taha

Reverse Process

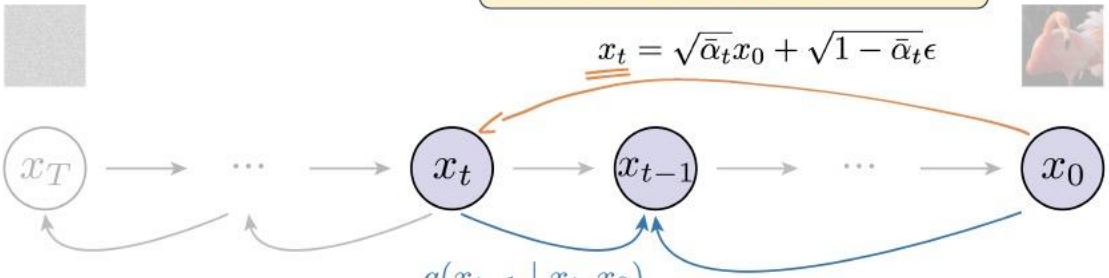


Everything here is known at sampling time except one thing: ε, the noise inside x_t.
To step back from x_t, subtract a scaled copy of the noise inside it, and rescale.
If you know the noise, you know the reverse step.

Slide credit: Bilal Taha

Reverse Process

tl; dr:
- outcome of the dependency graph
- some linear combinations



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

mean

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

"noise"

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

var

Slide credit: Bilal Taha

Closed-Form Reverse Posterior

Conditioning on the clean sample turns the exact reverse step into a Gaussian:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\left(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t \mathbf{I}\right)$$

$$\tilde{\mu}_t(x_t, x_0) = \underbrace{\frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t}\, x_0}_{\text{clean-data contribution}} + \underbrace{\frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\, x_t}_{\text{current noisy state}}$$

$$\tilde{\beta}_t = \underbrace{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\,\beta_t}_{\text{fixed by the noise schedule}}$$

The posterior mean is a weighted combination of x_0 and x_t.
Its variance contains no learned parameters.

Slide credit: Bilal Taha

Parameterizing the Learned Reverse Step

We model

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right), \qquad \Sigma_\theta(x_t, t) = \tilde{\beta}_t \mathbf{I}$$

The schedule supplies the variance, so the network only needs to determine the mean.

Direct mean	$\mu_\theta(x_t, t) = \mathrm{UNet}_\theta(x_t, t)$
Clean sample	$\hat{x}_{0,\theta} = \mathrm{UNet}_\theta(x_t, t), \quad \mu_\theta = \tilde{\mu}_t(x_t, \hat{x}_{0,\theta})$
Forward noise	$\epsilon_\theta(x_t, t) = \mathrm{UNet}_\theta(x_t, t)$ *(then recover $\hat{x}_{0,\theta}$ and substitute it into $\tilde{\mu}_t$)* DDPM predicts the noise. Reason: ε is always N(0, I), so the target has the same scale at every timestep. The network never has to output tiny corrections at some steps and large ones at others, which makes optimization easier

All three choices target the same posterior mean. They differ only in what the network predicts.

Slide credit: Bilal Taha

Noise Prediction Parameterization

Solve the forward sampling equation for the clean sample:

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon \implies \hat{x}_{0,\theta}(x_t, t) = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\, \epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}$$

A neural network that guesses the noise. Then, we get a guess of x_0

Substituting $\hat{x}_{0,\theta}$ into the closed-form posterior mean gives

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

Therefore

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \tilde{\beta}_t \mathbf{I} \right)$$

Interpretation: the network predicts the forward noise component. The fixed schedule converts that prediction into the mean and variance of ***one*** reverse step.

Slide credit: Bilal Taha

Gaussian Reverse Steps Can Model Non-Gaussian Data

Each local transition is Gaussian:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_t\right)$$

The data distribution comes from marginalizing the complete latent path:

$$p_\theta(x_0) = \int p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)\, dx_{1:T}$$

Since every reverse step is Gaussian, does that mean the images we generate follow a Gaussian?

a. Yes
b. No

Slide credit: Bilal Taha

Gaussian Reverse Steps Can Model Non-Gaussian Data

Each local transition is Gaussian:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_t\right)$$

The data distribution comes from marginalizing the complete latent path:

$$p_\theta(x_0) = \int p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)\, dx_{1:T}$$

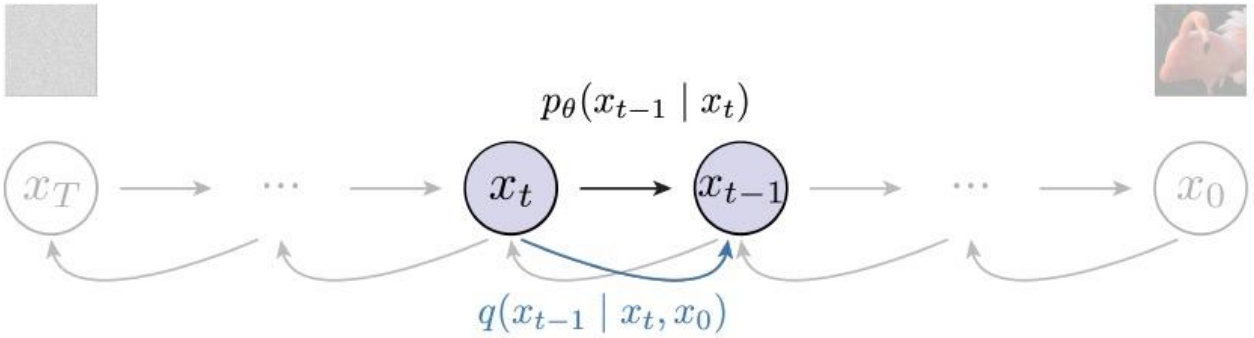
The mean $\mu_\theta(x_t, t)$ is a nonlinear neural network. After many reverse steps and integration over the latent states, the marginal $p_\theta(x_0)$ generally is not Gaussian.

This distinction is essential: simple local conditionals can produce a complex, multimodal distribution over clean images.

Simple local pieces, complex global result: cats, cars and flamingos in one distribution

Slide credit: Bilal Taha

Reverse Process



Recap:
We picked the **target**: the true reverse step $q(x_{t-1}|x_t)$ is unknown, so we switch to $q(x_{t-1}|x_t, x_0)$, which is known and Gaussian.
Its formula hasn't been derived yet

- **tl; dr**: a known Gaussian
- we want to learn it by p_θ
- we can represent p_θ by a Gaussian
- minimize KL divergence

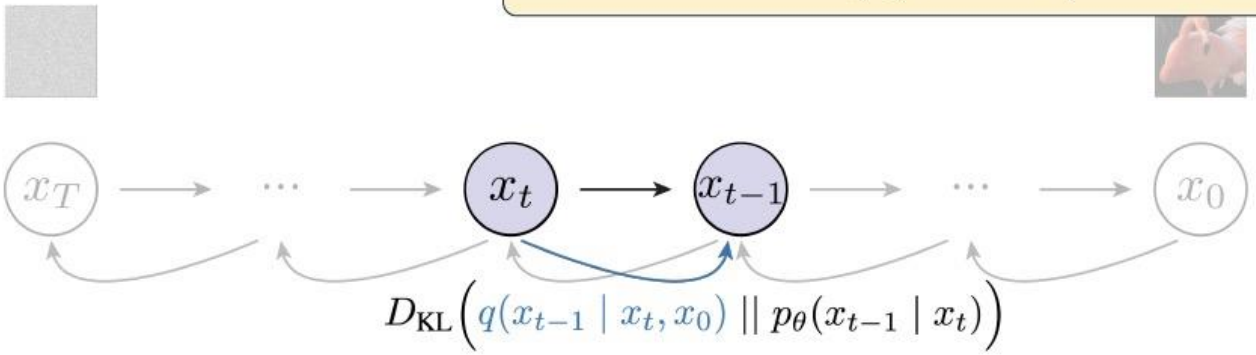
We model it with a Gaussian p_θ.
To make them match, we will minimize the KL divergence between them, just as in the VAE.

Slide credit: Bilal Taha

Reverse Process

D_{KL} of two Gaussians is like L2 loss: (pset 1)

$$D_{KL}(\mathcal{N}_1 \parallel \mathcal{N}_2) = \log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + \|\mu_1 - \mu_2\|^2}{2\sigma_2^2} - \frac{1}{2}$$



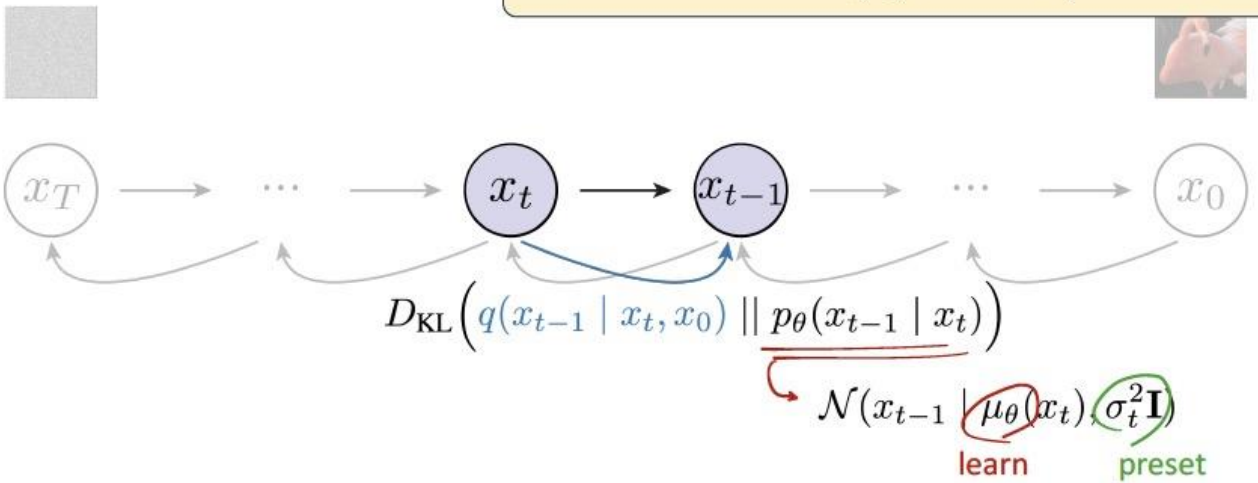
Slide credit: Bilal Taha

σ_1^2 belongs to the target and σ_2^2 to the model

Reverse Process

D_{KL} of two Gaussians is like L2 loss: (pset 1)

$$D_{KL}(\mathcal{N}_1 \parallel \mathcal{N}_2) = \log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + \|\mu_1 - \mu_2\|^2}{2\sigma_2^2} - \frac{1}{2}$$



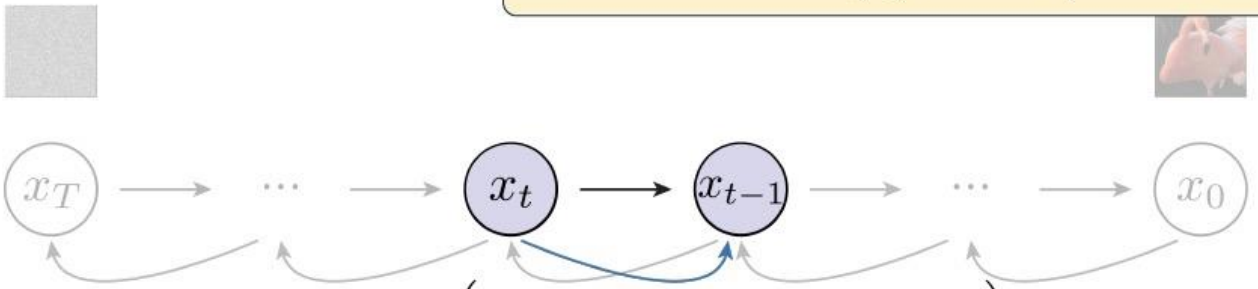
Both variances here are preset by the schedule, so the *log* term and the variance ratio are constants. Only the mean depends on θ. The KL collapses to a squared distance between two means, divided by $2{\sigma_t}^2$

Slide credit: Bilal Taha

Reverse Process

D_{KL} of two Gaussians is like L2 loss: (pset 1)

$$D_{KL}(\mathcal{N}_1 \parallel \mathcal{N}_2) = \log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + \|\mu_1 - \mu_2\|^2}{2\sigma_2^2} - \frac{1}{2}$$



$$D_{\mathrm{KL}}\left(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\right)$$

$$\mathcal{N}(x_{t-1} \mid \tilde{\mu}(x_t, x_0), \tilde{\beta}_t \mathbf{I}) \qquad \mathcal{N}(x_{t-1} \mid \mu_\theta(x_t), \sigma_t^2 \mathbf{I})$$

learn preset

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

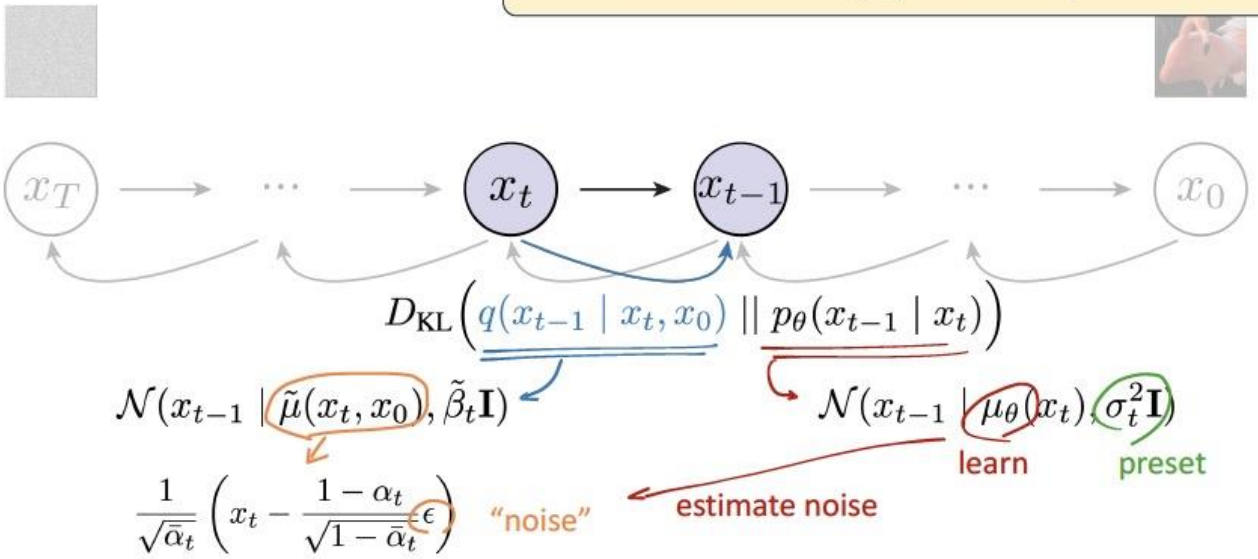
The true mean in noise form, with the true ε.

Slide credit: Bilal Taha

Reverse Process

D_{KL} of two Gaussians is like L2 loss: (pset 1)

$$D_{KL}(\mathcal{N}_1 \parallel \mathcal{N}_2) = \log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + \|\mu_1 - \mu_2\|^2}{2\sigma_2^2} - \frac{1}{2}$$



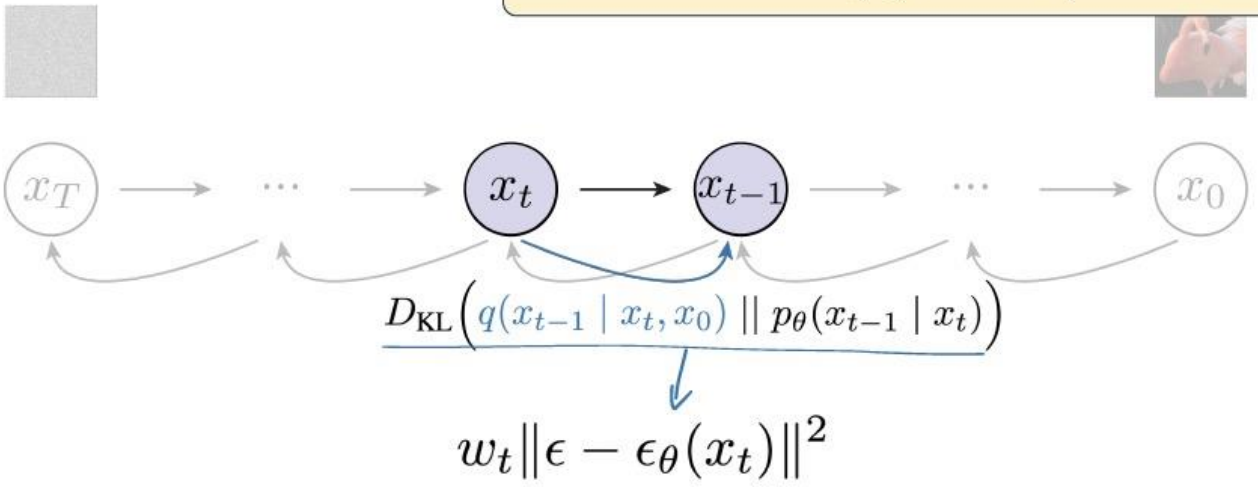
Our mean has the same form, with ε_θ. Everything else cancels. What's left is a squared distance between the true noise and the predicted noise, times a constant that depends only on t

Slide credit: Bilal Taha

Reverse Process

D_{KL} of two Gaussians is like L2 loss: (pset 1)

$$D_{KL}(\mathcal{N}_1 \parallel \mathcal{N}_2) = \log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + \|\mu_1 - \mu_2\|^2}{2\sigma_2^2} - \frac{1}{2}$$



So our loss is the mean squared error between two noise vectors. Plain regression.

Slide credit: Bilal Taha

Reverse Process

D_{KL} of two Gaussians is like L2 loss: (pset 1)

$$D_{KL}(\mathcal{N}_1 \parallel \mathcal{N}_2) = \log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + \|\mu_1 - \mu_2\|^2}{2\sigma_2^2} - \frac{1}{2}$$



$$D_{\mathrm{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \;||\; p_\theta(x_{t-1} \mid x_t)\Big)$$

$$w_t \|\epsilon - \epsilon_\theta(x_t)\|^2$$

- a network to predict noise
- input: noisy image

Slide credit: Bilal Taha

Reverse Process

D_{KL} of two Gaussians is like L2 loss: (pset 1)

$$D_{KL}(\mathcal{N}_1 \parallel \mathcal{N}_2) = \log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + \|\mu_1 - \mu_2\|^2}{2\sigma_2^2} - \frac{1}{2}$$



$$D_{\mathrm{KL}}\left(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\right)$$

$$w_t \left\| \epsilon - \epsilon_\theta(x_t) \right\|^2$$

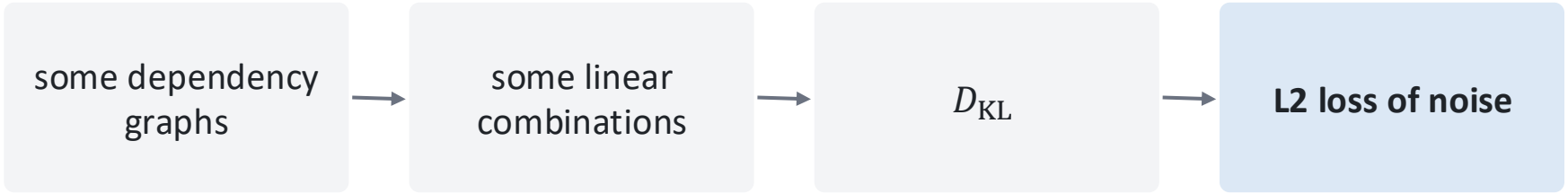
- weights due to α_t , β_t
- but set as 1 **(critical)**

- a network to predict noise
- input: noisy image

w_t is a weight that falls out of the α's and β's: ${\beta_t}^2 / \left(2{\sigma_t}^2 \alpha_t (1 - \bar{\alpha}_t)\right)$. Then the move that surprised everyone in 2020: set $w_t = 1$. Throw the weight away.

Slide credit: Bilal Taha

tl;dr



Slide credit: Bilal Taha

Diffusion Models

1 **Forward process**

add noise to data

2 **Reverse process**

learn to denoise

3 **Training objective**

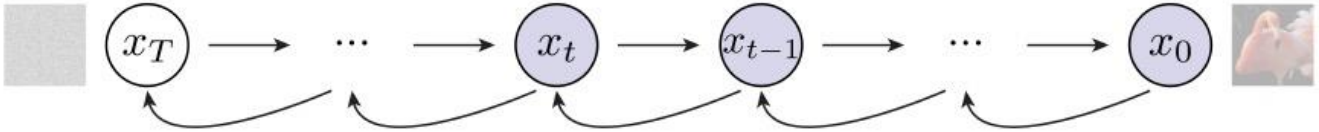
from Hierarchical VAE to L2 loss

4 **Noise Conditional Network**

represent distributions

Slide credit: Bilal Taha

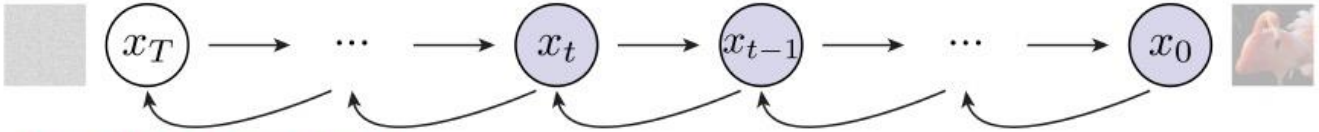
Training Objective



$$
\begin{aligned}
\mathcal{L}_{\text{VLB}} &:= \mathcal{L}_T + \mathcal{L}_{T-1} + \ldots + \mathcal{L}_0 \\
\mathcal{L}_T &:= D_{\text{KL}}\Big(q(x_T \mid x_0) \,||\, p_\theta(x_T)\Big) \\
\mathcal{L}_{t-1} &:= D_{\text{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\Big) \\
\mathcal{L}_0 &:= -\log p_\theta(x_0 \mid x_1)
\end{aligned}
$$

Slide credit: Bilal Taha

Training Objective



- variational lower bound
- like ELBO

$$\mathcal{L}_{\text{VLB}} := \mathcal{L}_T + \mathcal{L}_{T-1} + \ldots + \mathcal{L}_0$$

$$\mathcal{L}_T := D_{\text{KL}}\left(q(x_T \mid x_0) \,||\, p_\theta(x_T)\right)$$

$$\mathcal{L}_{t-1} := D_{\text{KL}}\left(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\right)$$

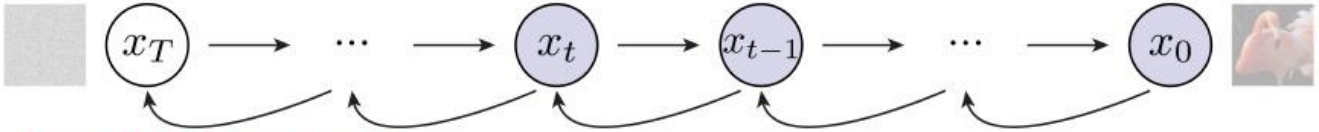
$$\mathcal{L}_0 := -\log p_\theta(x_0 \mid x_1)$$

VLB means variational lower bound. It's the ELBO from last lecture applied to a chain of latents instead of one. Diffusion is a hierarchical VAE with T layers of latents

is the fully noised image close to the N(0, I) we start sampling from? It has no learnable parameters, so it's a constant and is dropped.

Slide credit: Bilal Taha

Training Objective



- variational lower bound
- like ELBO

$$\mathcal{L}_{\text{VLB}} := \mathcal{L}_T + \mathcal{L}_{T-1} + \ldots + \mathcal{L}_0$$

$$\mathcal{L}_T := D_{\text{KL}}\Big(q(x_T \mid x_0) \,||\, p_\theta(x_T)\Big)$$

$$\mathcal{L}_{t-1} := D_{\text{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\Big)$$

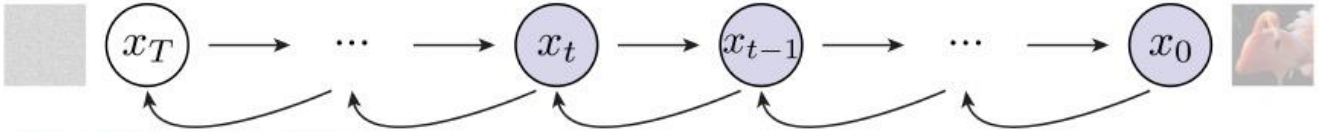
$$\mathcal{L}_0 := -\log p_\theta(x_0 \mid x_1)$$

L_{t-1}, one term for each $t = 2 \ldots T$: $KL\big(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)\big)$. How well each learned reverse step matches the true one. This is where learning happens, and each becomes the noise L2 loss

$L_0 = -log\, p_\theta(x_0|x_1)$: the final reconstruction, like the VAE's reconstruction term

Slide credit: Bilal Taha

Training Objective



- variational lower bound
- like ELBO

$$\mathcal{L}_{\text{VLB}} := \mathcal{L}_T + \mathcal{L}_{T-1} + \ldots + \mathcal{L}_0$$

$$\mathcal{L}_T := D_{\text{KL}}\left(q(x_T \mid x_0) \,||\, p_\theta(x_T)\right)$$

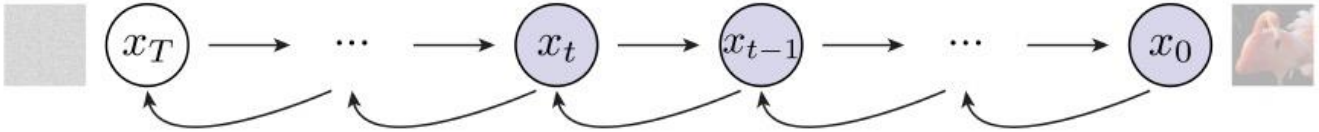
it's ELBO if one step

$$\mathcal{L}_{t-1} := D_{\text{KL}}\left(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\right)$$

$$\mathcal{L}_0 := -\log p_\theta(x_0 \mid x_1)$$

Slide credit: Bilal Taha

Training Objective



$$\mathcal{L}_{\text{VLB}} := \mathcal{L}_T + \mathcal{L}_{T-1} + \ldots + \mathcal{L}_0$$

no parameter, unlike VAE's q_ϕ

$$\mathcal{L}_T := D_{\text{KL}}\Big(q(x_T \mid x_0) \,||\, p_\theta(x_T)\Big)$$

constant — Gaussian

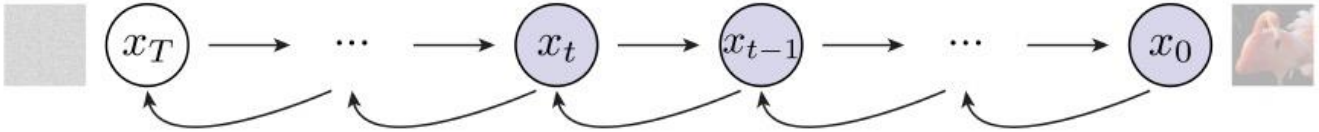
$$\mathcal{L}_{t-1} := D_{\text{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\Big)$$

$$\mathcal{L}_0 := -\log p_\theta(x_0 \mid x_1)$$

L_T: $q(x_T|x_0)$ has no parameters, unlike the VAE's q_φ, and $p(x_T)$ is a fixed standard Gaussian. Nothing here depends on θ. It's a constant, so we drop it.

Slide credit: Bilal Taha

Training Objective



$$\mathcal{L}_{\text{VLB}} := \mathcal{L}_T + \mathcal{L}_{T-1} + \ldots + \mathcal{L}_0$$

$$\mathcal{L}_T := D_{\text{KL}}\Big(q(x_T \mid x_0) \,||\, p_\theta(x_T)\Big)$$

$$\mathcal{L}_{t-1} := D_{\text{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\Big)$$

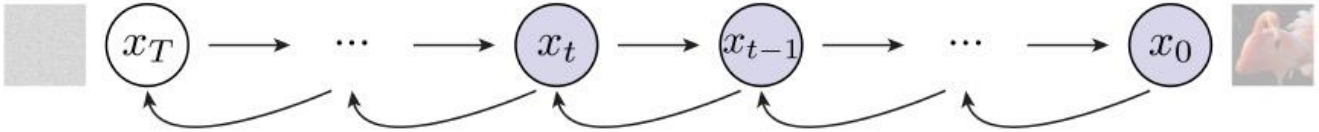
reconstruction loss, like VAE

$$\mathcal{L}_0 := -\log p_\theta(x_0 \mid x_1)$$

L_0 is a reconstruction loss, like the VAE's. It's a tiny part of the sum, and the simplified loss absorbs it into the same form at t = 1.

Slide credit: Bilal Taha

Training Objective



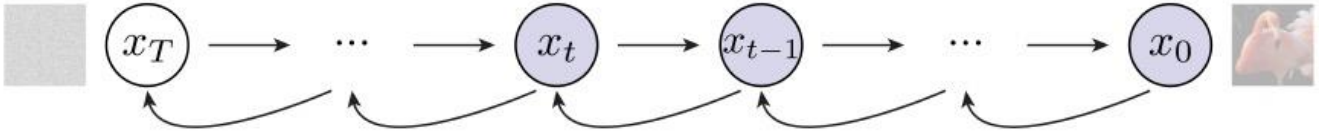
$$
\begin{aligned}
\mathcal{L}_{\text{VLB}} &:= \mathcal{L}_T + \mathcal{L}_{T-1} + \ldots + \mathcal{L}_0 \\
\mathcal{L}_T &:= D_{\text{KL}}\Big(q(x_T \mid x_0) \,||\, p_\theta(x_T)\Big) \\
\mathcal{L}_{t-1} &:= D_{\text{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\Big) \\
\mathcal{L}_0 &:= -\log p_\theta(x_0 \mid x_1)
\end{aligned}
$$

L2 loss on noise

Every middle term is what we derived: an L2 loss on noise. So the whole bound is, up to constants and weights, a sum of noise-prediction errors

Slide credit: Bilal Taha

Training Objective



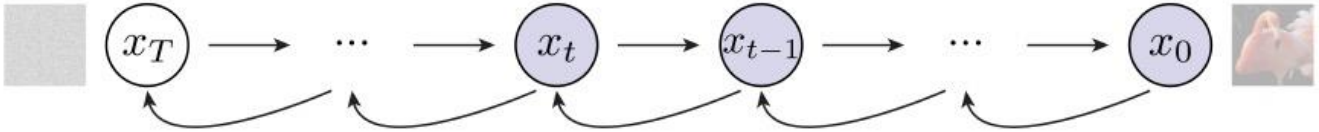
$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

Rather than summing T terms, we can write it as one expectation.
Three things are random:

- the image x_0 from the data
- the timestep t uniform over 1 to T
- the noise ε from $N(0, I)$.

Slide credit: Bilal Taha

Training Objective



$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

over p_{data}

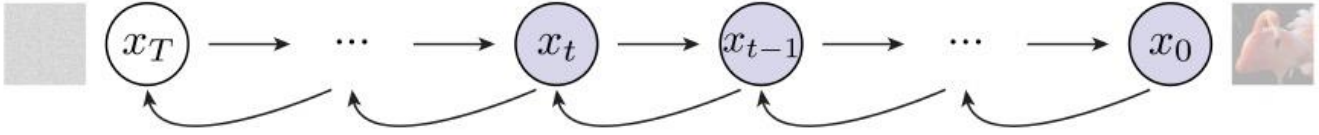
over $[1, T]$

over $\mathcal{N}(0, \mathbf{I})$

Do we need all 1,000 timesteps for every image on every update?

Slide credit: Bilal Taha

Training Objective



$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

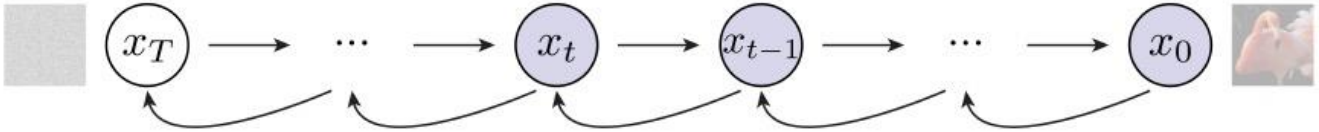
over p_{data} over $[1, T]$ over $\mathcal{N}(0, \mathbf{I})$

Do we need all 1,000 timesteps for every image on every update?

No. Draw one random t per image; that gives an unbiased estimate of the full sum.
Remember the optimization lecture: Same argument that justified SGD. A minibatch gradient is a noisy but unbiased estimate of the full gradient. Here we sample timesteps the way SGD samples data points. More noise in the gradient, much cheaper updates

Slide credit: Bilal Taha

Training Objective



$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

set as 1 **(critical)**

When we turned each KL term into a squared error, every timestep came with its own factor in front: $w_t \cdot \|\varepsilon - \varepsilon_\theta(x_t, t)\|^2$, with $w_t = {\beta_t}^2/\left(2{\sigma_t}^2\alpha_t(1-\bar{\alpha}_t)\right)$.

Keep those factors and you are optimizing the exact bound. Set every w_t to 1 and you get L_{simple}

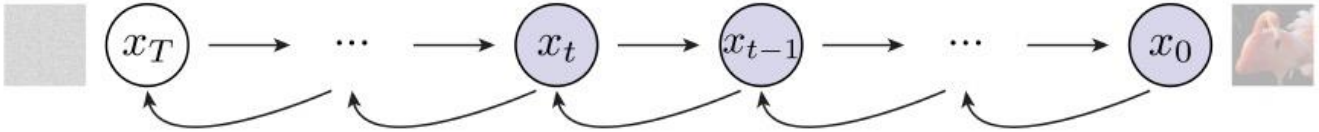
Objective	IS	FID
L, learned diagonal $\boldsymbol{\Sigma}$	–	–
L, fixed isotropic $\boldsymbol{\Sigma}$	7.67 ± 0.13	13.51
$\|\tilde{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}_\theta\|^2$ (L_{simple})	$\mathbf{9.46 \pm 0.11}$	**3.17**

[Ho et al. 2020]; see more in [Salimans & Ho, 2022]

We're no longer optimizing a true bound on the likelihood. Should that worry us?" Discussion point: the bound is a means, not the goal. We care about sample quality, and the reweighting trades a little likelihood for much better samples

Slide credit: Bilal Taha

Training Objective



$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

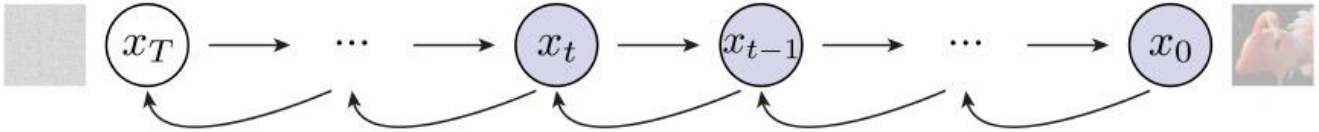
network to predict noise

conditioned on noise level **(critical)**

Why does the network need t? Couldn't it guess the noise level from the image?

Slide credit: Bilal Taha

Training Objective



$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

network to predict noise

conditioned on noise level **(critical)**

Why does the network need t? Couldn't it guess the noise level from the image?
Partly, but telling it removes the ambiguity, and its job differs a lot between t = 10 (fix fine texture) and t = 900 (invent a global layout)

Slide credit: Bilal Taha

DDPM Variational Bound

$$\mathcal{L}_{\mathrm{VLB}} = \underbrace{D_{\mathrm{KL}}(q(x_T \mid x_0) \,\|\, p(x_T))}_{\mathcal{L}_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t))}_{\mathcal{L}_{t-1}} + \underbrace{-\log p_\theta(x_0 \mid x_1)}_{\mathcal{L}_0}.$$

Term	Role	What happens in a DDPM
$\mathcal{L}_T$	Match the terminal latent to the prior	The fixed schedule makes $q(x_T \mid x_0)$ close to $\mathcal{N}(0, I)$; this term has no learned reverse mean.
$\mathcal{L}_{t-1}$	Learn each reverse transition	With Gaussian conditionals, each KL becomes a weighted error between the exact and learned reverse means.
$\mathcal{L}_0$	Reconstruct the clean sample	The final transition assigns probability to x_0 given x_1.

The common $\mathcal{L}_{\mathrm{simple}}$ objective retains the denoising regression and removes the analytic timestep weights.

Slide credit: Bilal Taha

Training Losses for the Three Prediction Targets

Every option receives the same pair (x_t, t). The parameterization determines the supervised target used for that randomly sampled timestep.

Network output	**Known training target**	**Per-timestep loss**
Reverse mean	$\tilde{\mu}_t(x_t, x_0)$ from the exact posterior	$\|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\|_2^2$
Clean sample	x_0	$\|x_0 - \hat{x}_{0,\theta}(x_t, t)\|_2^2$
Forward noise	sampled $\epsilon \sim \mathcal{N}(0, I)$	$\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2$

The clean sample and sampled noise are already available when constructing x_t. Noise prediction therefore turns each update into a direct supervised regression problem.

Slide credit: Bilal Taha

Noise-Prediction Training Algorithm

For each training update:

1. Sample a clean example $x_0 \sim p_{\text{data}}$.
2. Sample a timestep $t \sim \text{Uniform}\{1, \ldots, T\}$ and noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$.
3. Construct the noisy input directly:

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

4. Predict the added noise and update θ using

$$\boxed{\mathcal{L}_{\text{simple}}(\theta) = \|\epsilon - \epsilon_\theta(x_t, t)\|_2^2}.$$

Sampling one timestep per example gives a Monte Carlo estimate of the loss over all timesteps. The model never needs to simulate the preceding forward steps $x_1, \ldots, x_{t-1}$.

Slide credit: Bilal Taha

Diffusion Models

1 **Forward process**

add noise to data

2 **Reverse process**

learn to denoise

3 **Training objective**

from Hierarchical VAE to L2 loss

4 **Noise Conditional Network**

represent distributions

Slide credit: Bilal Taha

Noise Conditional Network

- Diffusion models decompose a distribution into **many** simpler ones.
- We need the same # networks to fit **all** of them.
- We can **combine** all into one "powerful" network.
- This network is conditioned on noise level t.
- **Noise Conditional Network** [Song & Ermon 2019]: things made work

*It is called Noise Conditional Score Network (NCSN) in [Song & Ermon 2019] in the context of score matching.

Slide credit: Bilal Taha

Noise Conditional Network

How to represent $p_\theta(x_{t-1} \mid x_t)$

- network input: x_t
- network output: μ and σ of a distribution
- parametrize μ by:

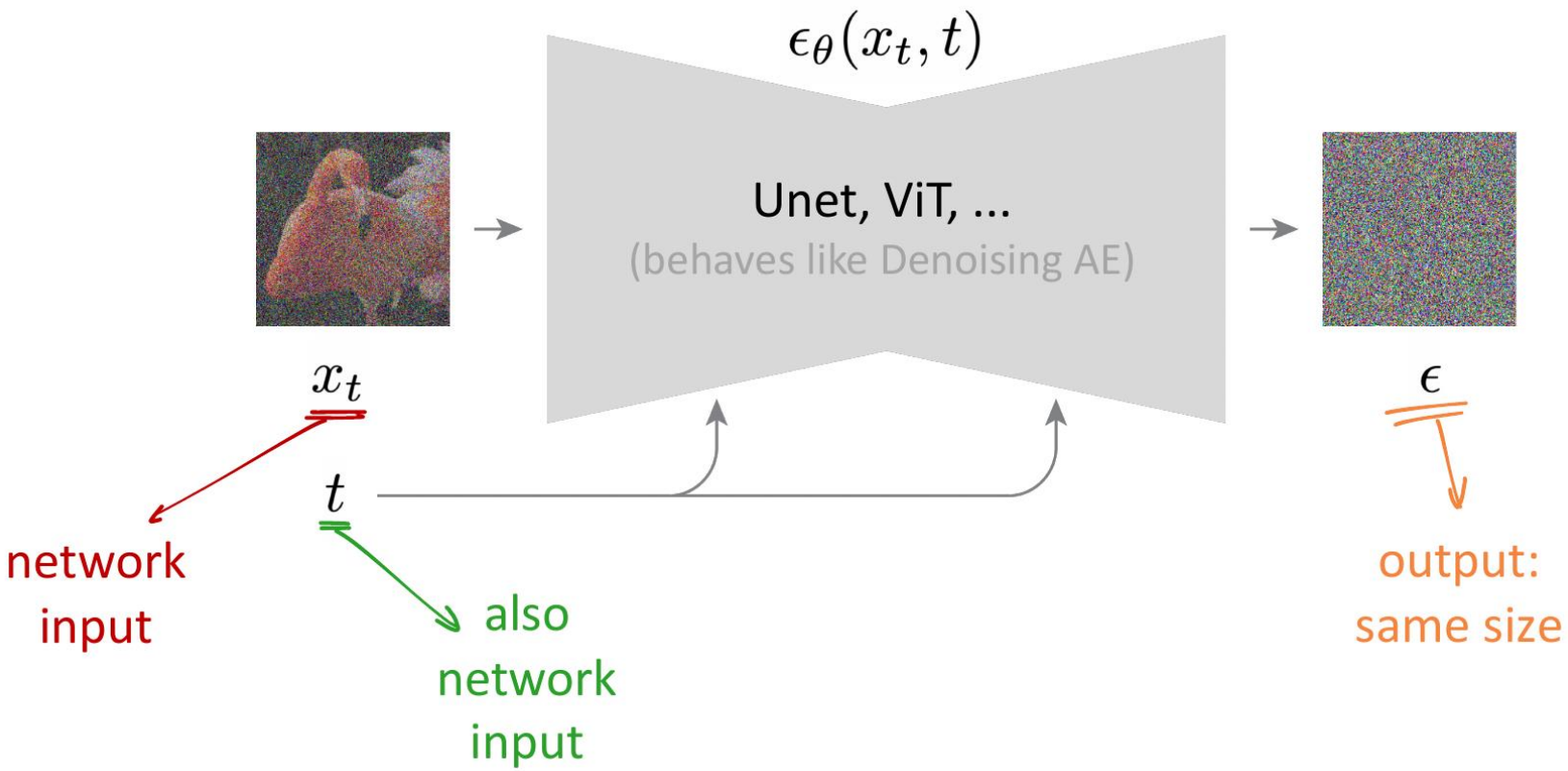
$$\epsilon_\theta(x_t, t)$$

noisy image x_t:
- condition
- network input

noise level t:
- condition
- network input

Slide credit: Bilal Taha

Noise Conditional Network



Slide credit: Bilal Taha

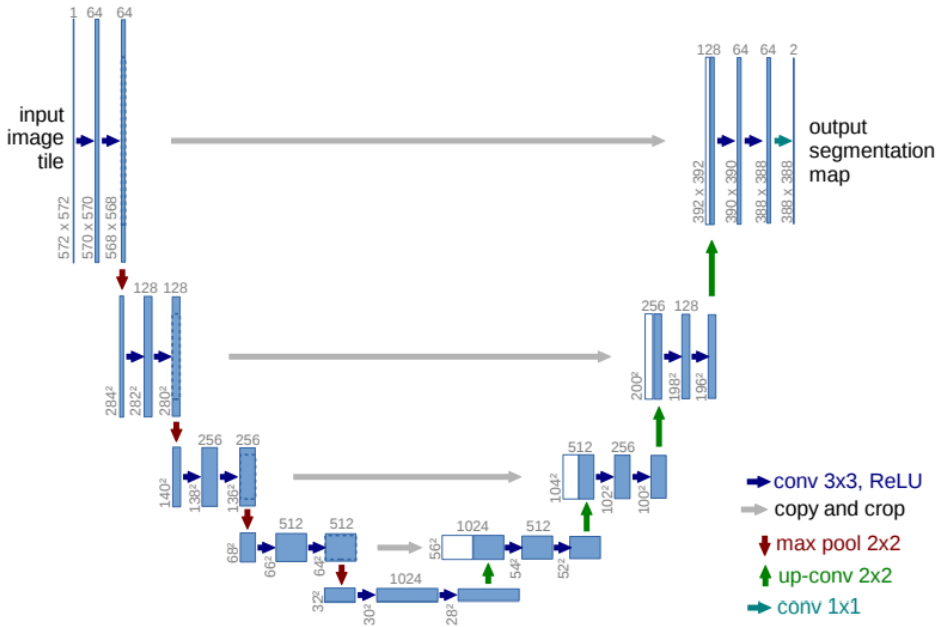
U-Net Architecture

Contracting path

- Downsamples the image
- Increases feature channels
- Captures large-scale context

Expanding path

- Restores spatial resolution
- Combines features through skip connections
- Produces a dense image-shaped output



The U-Net was built for dense segmentation. Diffusion reuses the same input-output geometry to predict noise at every pixel.

Slide credit: Bilal Taha. Image credit: Ronneberger et al. (2015)

Why Diffusion Models Use a U-Net

The denoising network maps an image-shaped input to an image-shaped prediction:

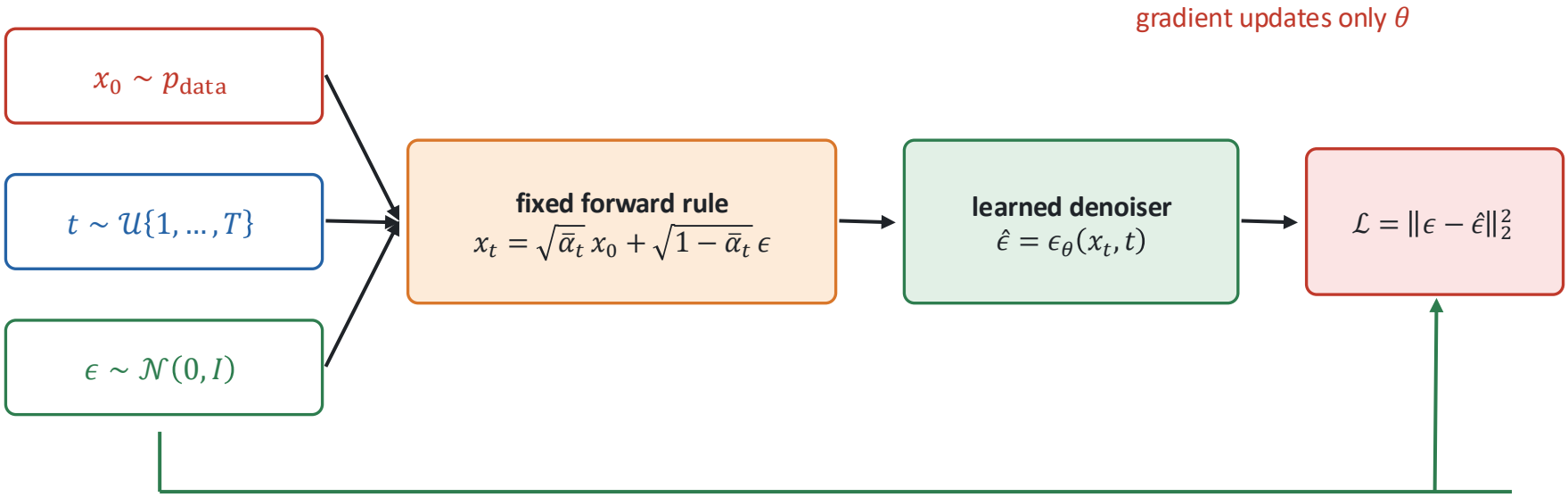
$$\underbrace{(x_t, t)}_{\text{noisy image and noise level}} \xrightarrow{\text{UNet}_\theta} \underbrace{\epsilon_\theta(x_t, t)}_{\text{predicted noise, same shape as } x_t}$$

- The contracting path provides the context needed to recognize image structure under noise.
- Skip connections preserve spatial details that the expanding path needs for pixel-level prediction.
- A timestep embedding conditions the network on the current noise level.

The architecture originally supported dense segmentation. Diffusion uses the same input-output geometry to predict noise at every pixel.

Slide credit: Bilal Taha

Training as a Computation Graph



The forward path has no learned parameters. Backpropagation changes only the U-Net, while x_0, t, and ϵ provide a supervised denoising target for that update.

Slide credit: Bilal Taha

Training and Sampling Use the Same Denoiser

	Training	Sampling
Starting point	Clean data $x_0 \sim p_{\text{data}}$	Pure noise $x_T \sim \mathcal{N}(0, I)$
State supplied to the U-Net	A directly constructed x_t and its timestep t	The current x_t and its timestep t
Network output	$\epsilon_\theta(x_t, t)$	The same $\epsilon_\theta(x_t, t)$
How the output is used	Compare with the known noise ϵ to form an L2 loss	Convert the prediction into a reverse mean and sample x_{t-1}
Number of network calls	One randomly sampled timestep per example and update	All timesteps $T, T-1, \ldots, 1$ for each generated sample

Training learns a local denoising rule. Sampling applies it repeatedly.

Slide credit: Bilal Taha

Sampling Under the Three Parameterizations

At timestep t, each network produces a different quantity, but every option must construct the same reverse mean before sampling $x_{t-1} = \mu_\theta(x_t, t) + \sigma_t z$.

Prediction	**Network call**	**Reverse mean used by the sampler**
Reverse mean	$\mu_\theta(x_t, t)$	Use the output directly.
Clean sample	$\hat{x}_{0,\theta}(x_t, t)$	Substitute it into the posterior formula: $\mu_\theta = \tilde{\mu}_t(x_t, \hat{x}_{0,\theta})$.
Forward noise	$\epsilon_\theta(x_t, t)$	First recover $\hat{x}_{0,\theta} = \left(x_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta\right)/\sqrt{\bar{\alpha}_t}$, then use $\tilde{\mu}_t(x_t, \hat{x}_{0,\theta})$.

The next slide writes the noise-prediction route in its usual direct form, which avoids explicitly materializing $\hat{x}_{0,\theta}$.

Slide credit: Bilal Taha

One Reverse Sampling Step

Given the current noisy state x_t:

1. Predict the forward noise:

$$\hat{\epsilon}_t = \epsilon_\theta(x_t, t)$$

2. Convert that prediction into the reverse-process mean:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \hat{\epsilon}_t \right)$$

3. Draw the preceding state:

$$x_{t-1} = \mu_\theta(x_t, t) + \sigma_t z, \qquad z \sim \mathcal{N}(0, I)$$

At $t = 1$, set $z = 0$. The last step returns the model's clean sample without adding new noise.

Slide credit: Bilal Taha

Diffusion algorithm annotated:

Algorithm 1 Training

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

x_t

Algorithm 2 Sampling

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

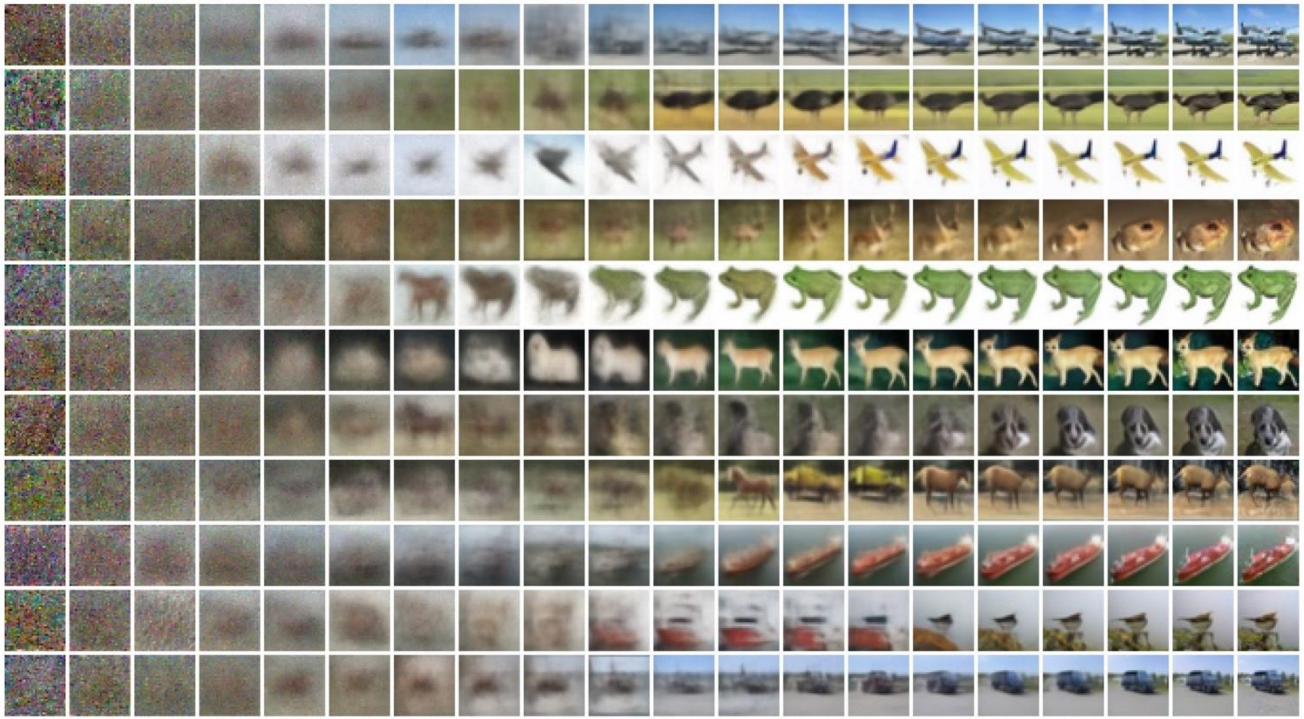
estimated μ

sampling from estimated distribution

Slide credit: Bilal Taha. Image credit: Ho et al. (2020)

Example: Unconditional Generation on CIFAR-10

noise
x_T



generated
x_0

[Ho et al., 2020]

Slide credit: Bilal Taha. Image credit: Ho et al. (2020)

Example: Shared Intermediate Latents



Each quadrant: the bottom-right image is the shared x_t (t = 1000, 750, 500, 250); the other three are samples generated from it.

[Ho et al., 2020]

Slide credit: Bilal Taha. Image credit: Ho et al. (2020)

Diffusion Model Summary

Fixed forward process The schedule constructs any noisy state directly:

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

Tractable training target Conditioning on the observed x_0 gives the Gaussian posterior $q(x_{t-1} \mid x_t, x_0)$ and its closed-form reverse mean.

Learned denoiser A timestep-conditioned U-Net predicts $\epsilon_\theta(x_t, t)$ by minimizing

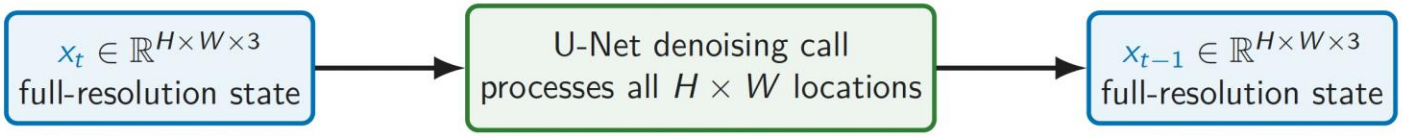
$$\mathcal{L}_{\text{simple}} = \|\epsilon - \epsilon_\theta(x_t, t)\|_2^2$$

Reverse sampling Starting from $x_T \sim \mathcal{N}(0, I)$, the model repeatedly samples $x_{t-1} \sim p_\theta(x_{t-1} \mid x_t)$ until it produces x_0

Diffusion combines a known corruption process with a learned sequence of local denoising steps.

Slide credit: Bilal Taha

The Cost of Pixel-Space Diffusion



the network runs repeatedly for $t = T, T-1, \ldots, 1$

Computation grows with image resolution

- Every reverse step stores and processes full spatial feature maps.
- Increasing H or W increases the cost of every denoising call.

Pixels contain spatial redundancy

- Neighboring pixels are strongly correlated.
- Full-resolution diffusion repeatedly models information that a compact representation could preserve.

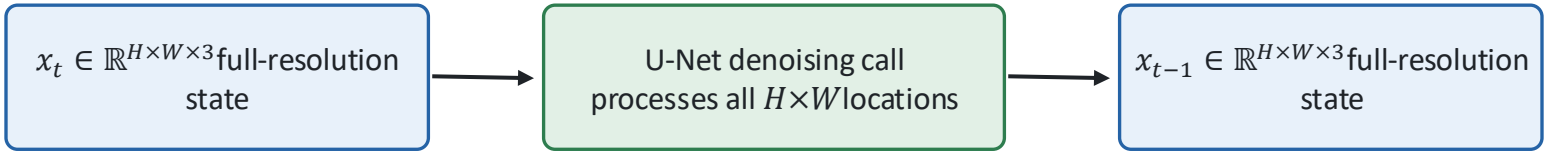
If an encoder reduces each spatial dimension by a factor f, the spatial map shrinks as

$$HW \longrightarrow \left(\frac{H}{f}\right)\left(\frac{W}{f}\right) = \frac{HW}{f^2}.$$

Can the diffusion process operate on this smaller representation?

Slide credit: Bilal Taha

The Cost of Pixel-Space Diffusion



the network runs repeatedly for $t = T, T-1, \ldots, 1$

Computation grows with image resolution

- Every reverse step stores and processes full spatial feature maps.
- Increasing H or W increases the cost of every denoising call.

Pixels contain spatial redundancy

- Neighboring pixels are strongly correlated.
- Full-resolution diffusion repeatedly models information that a compact representation could preserve.

If an encoder reduces each spatial dimension by a factor f, the spatial map shrinks as

$$HW \longrightarrow \left(\frac{H}{f}\right)\left(\frac{W}{f}\right) = \frac{HW}{f^2}$$

Can the diffusion process operate on this smaller representation?

Slide credit: Bilal Taha