Generative Model Zoo (part II)

Jun-Yan Zhu

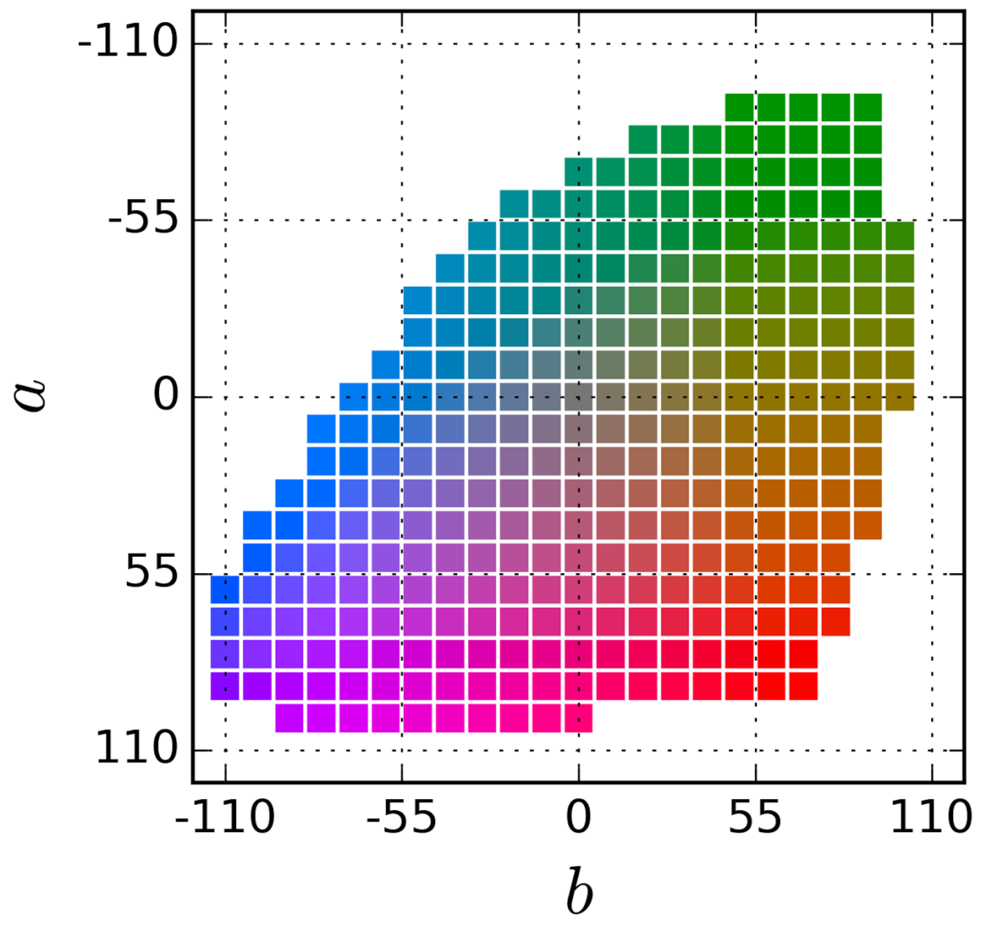
16-726 Learning-based Image Synthesis, Spring 2023

many slides from Phillip Isola, Richard Zhang, Alyosha Efros

Autoregressive Models

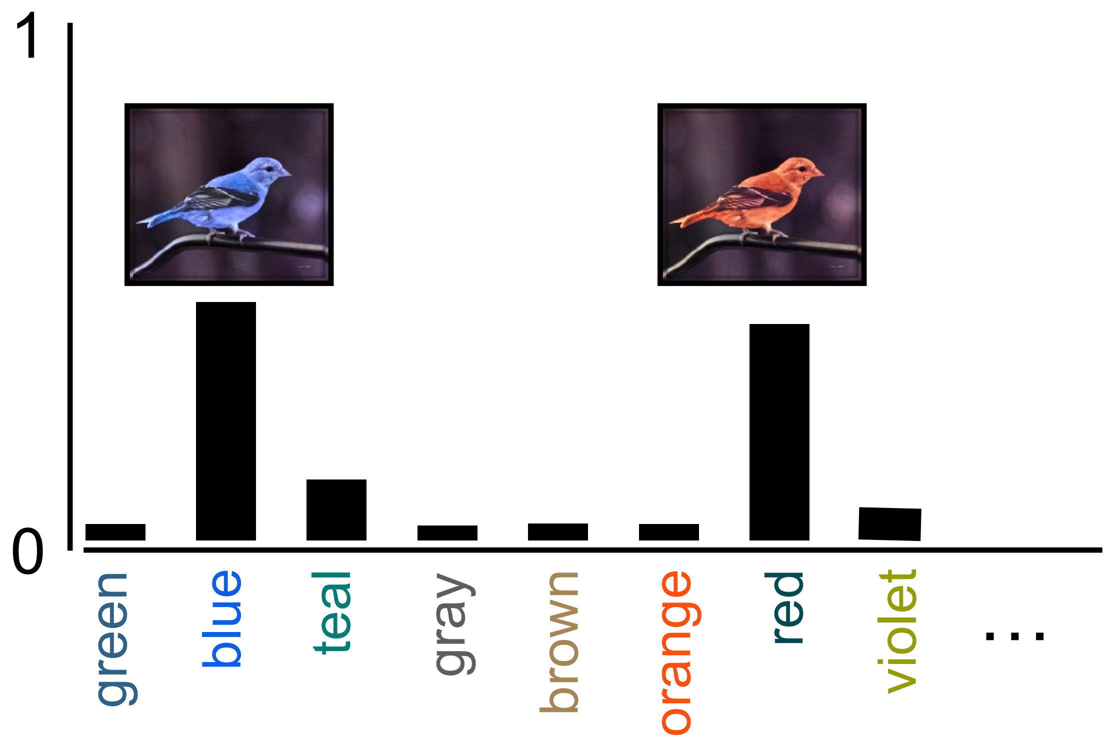
Recall: we can represent colors as discrete classes

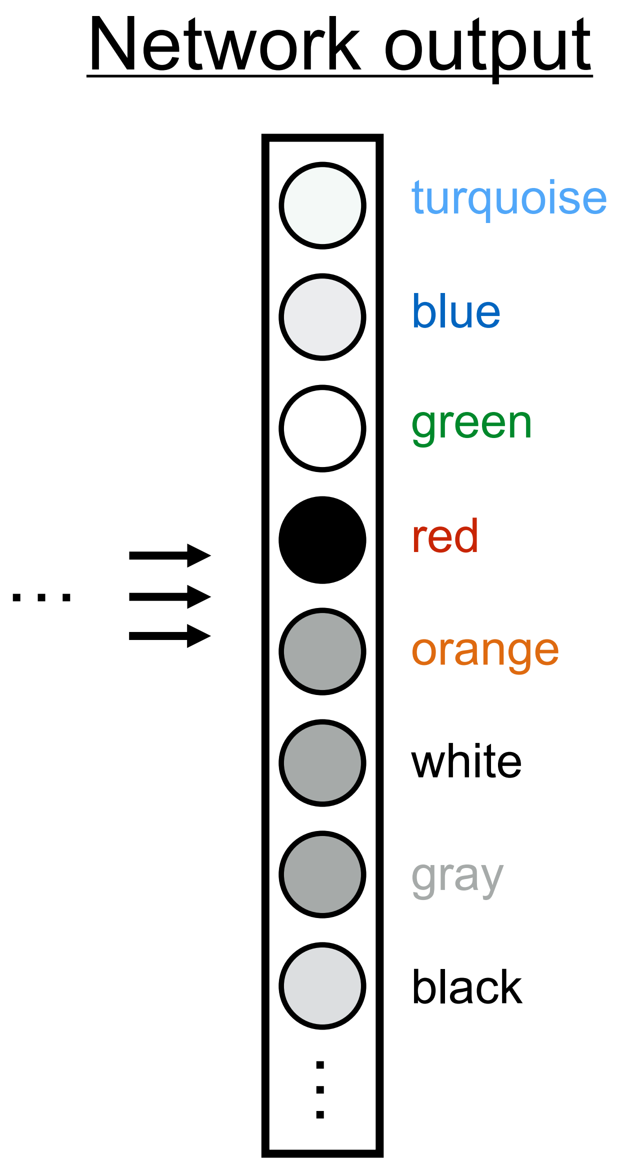
$$\mathbf{y} \in \mathbb{R}^{H \times W \times K}$$



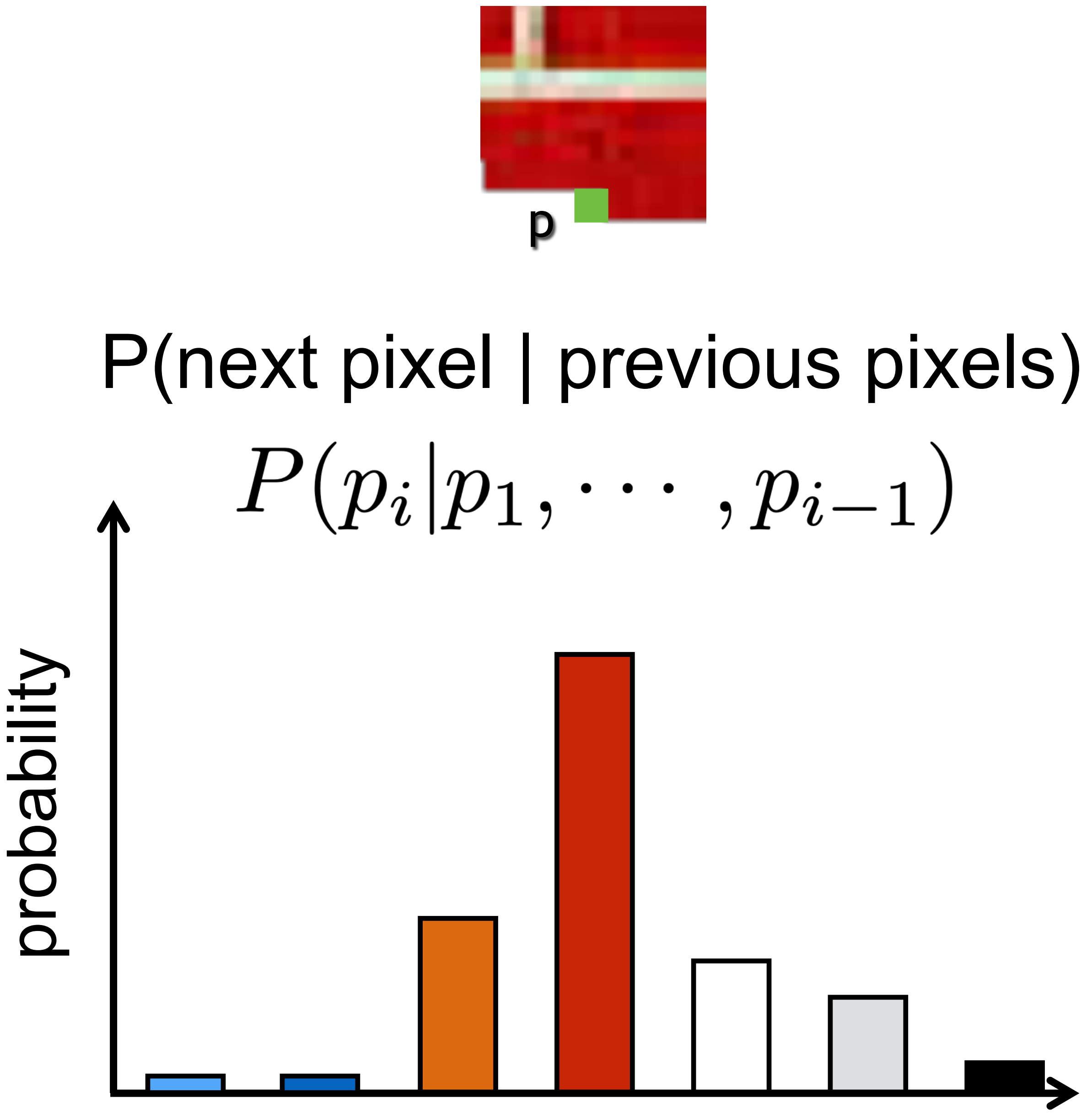
$$\mathcal{L}(\mathbf{y}, f_\theta(\mathbf{x})) = H(\mathbf{y}, \texttt{softmax}(f_\theta(\mathbf{x})))$$

Prediction for a single pixel i,j



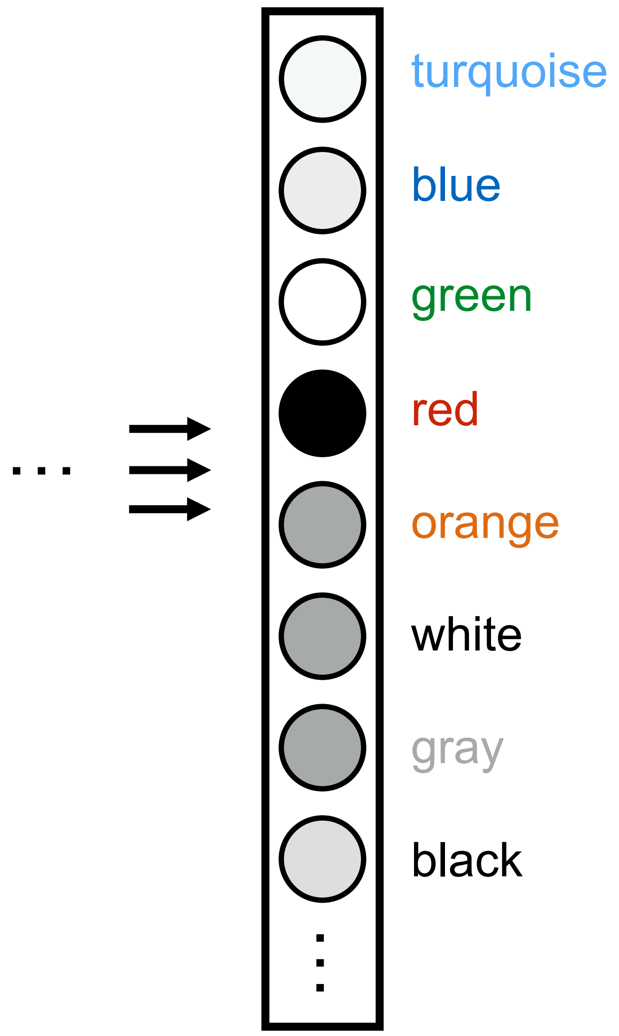


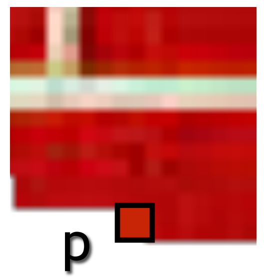
Network output
turquoise
blue
green
red
orange
white
gray
black



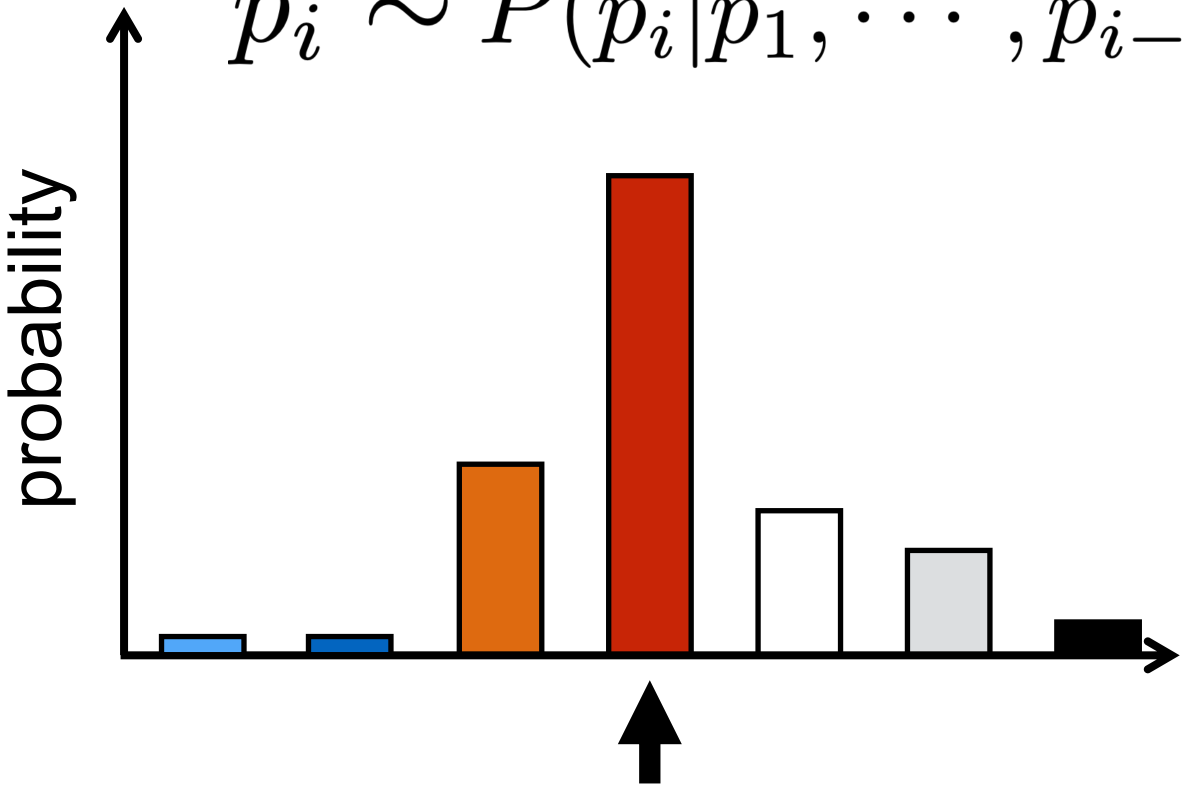
p
P(next pixel | previous pixels)
$P(p_i|p_1, \cdots, p_{i-1})$
probability

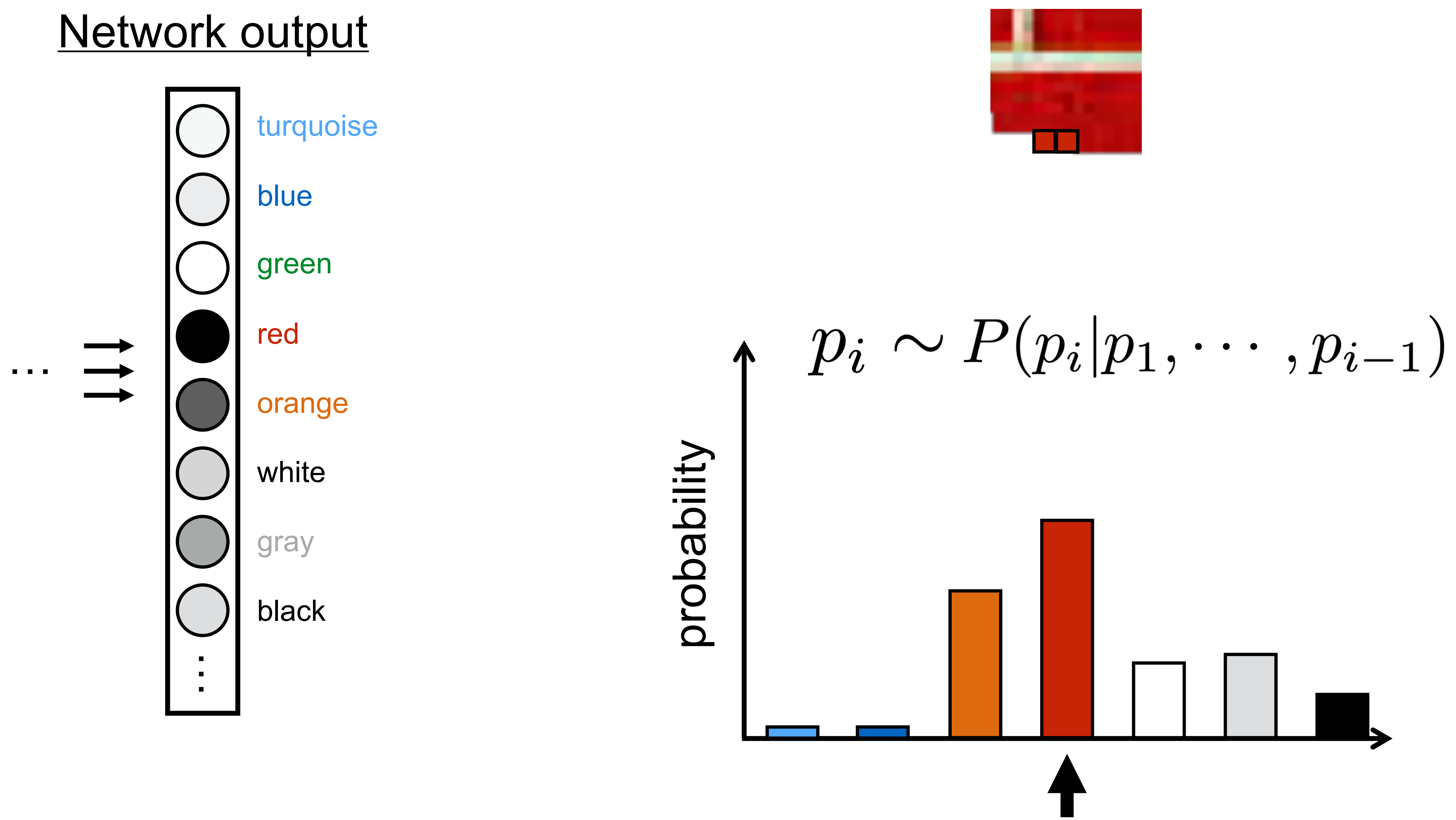
<u>Network output</u>



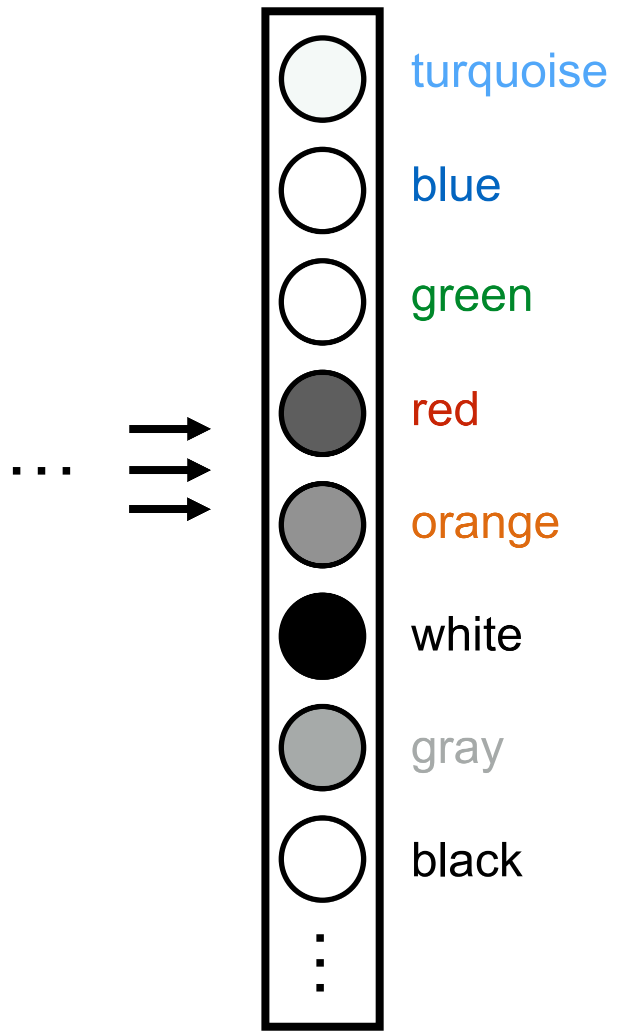


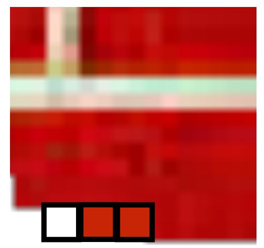
$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$

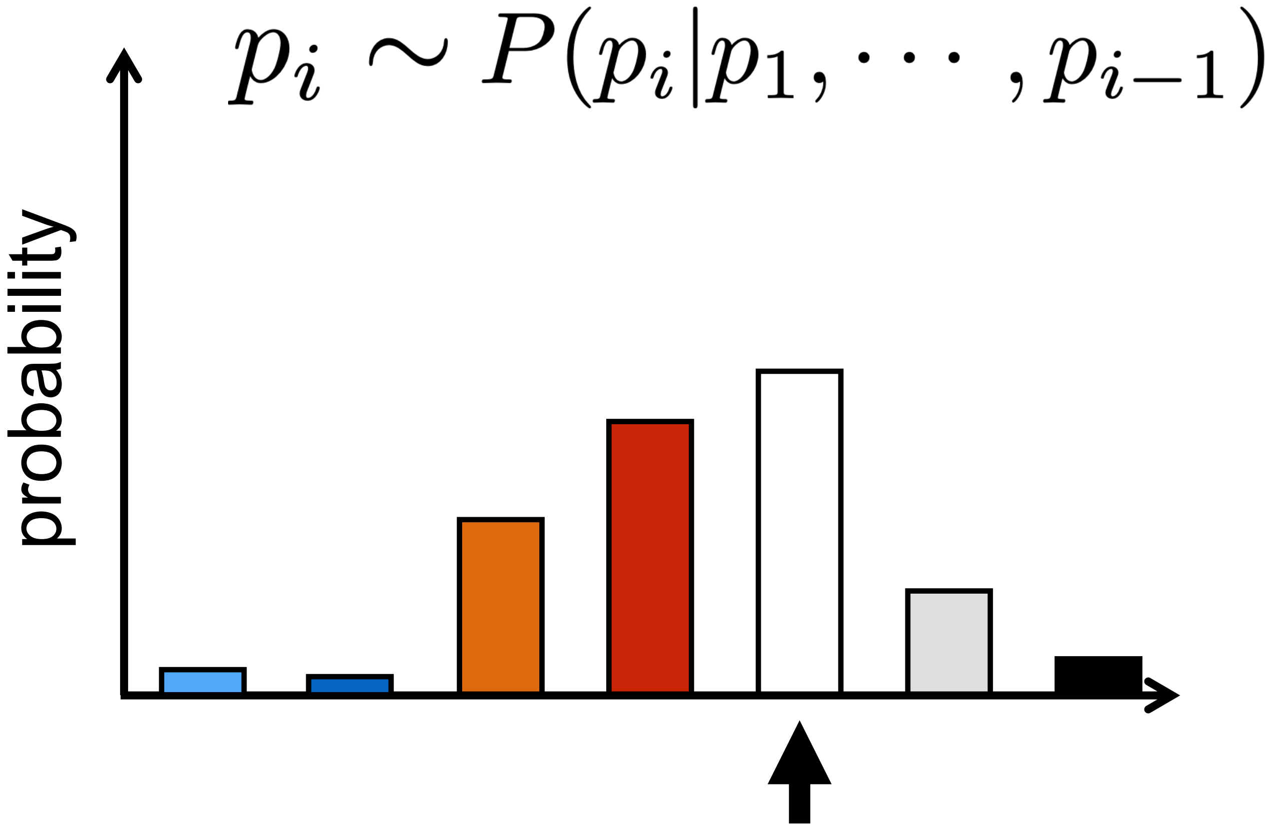



Network output
turquoise
blue
green
red
orange
white
gray
black
$p_i \sim P(p_i|p_1, \cdots, p_{i-1})$
probability

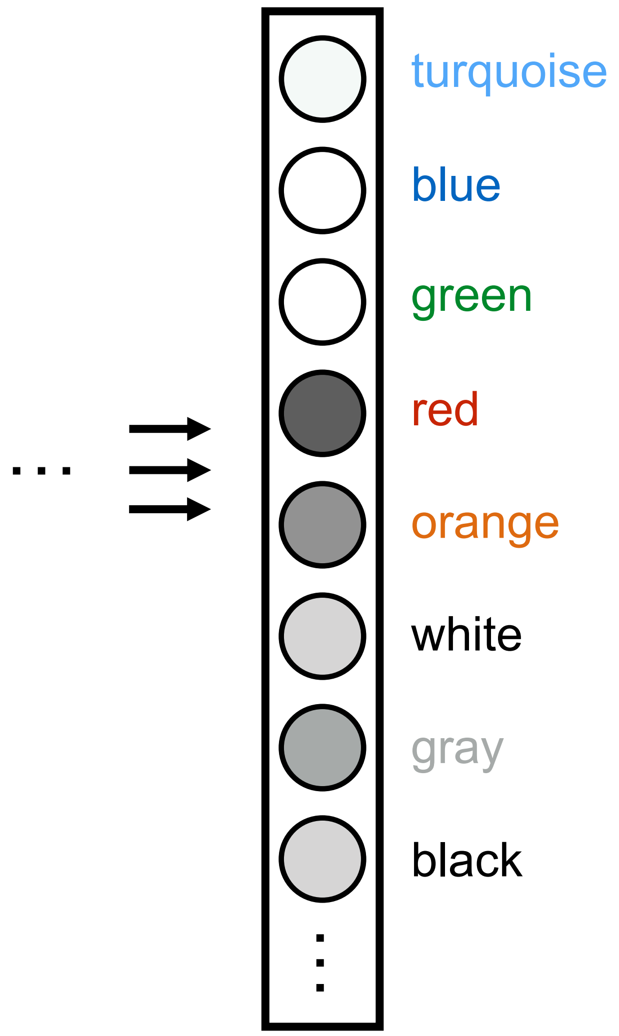
Network output



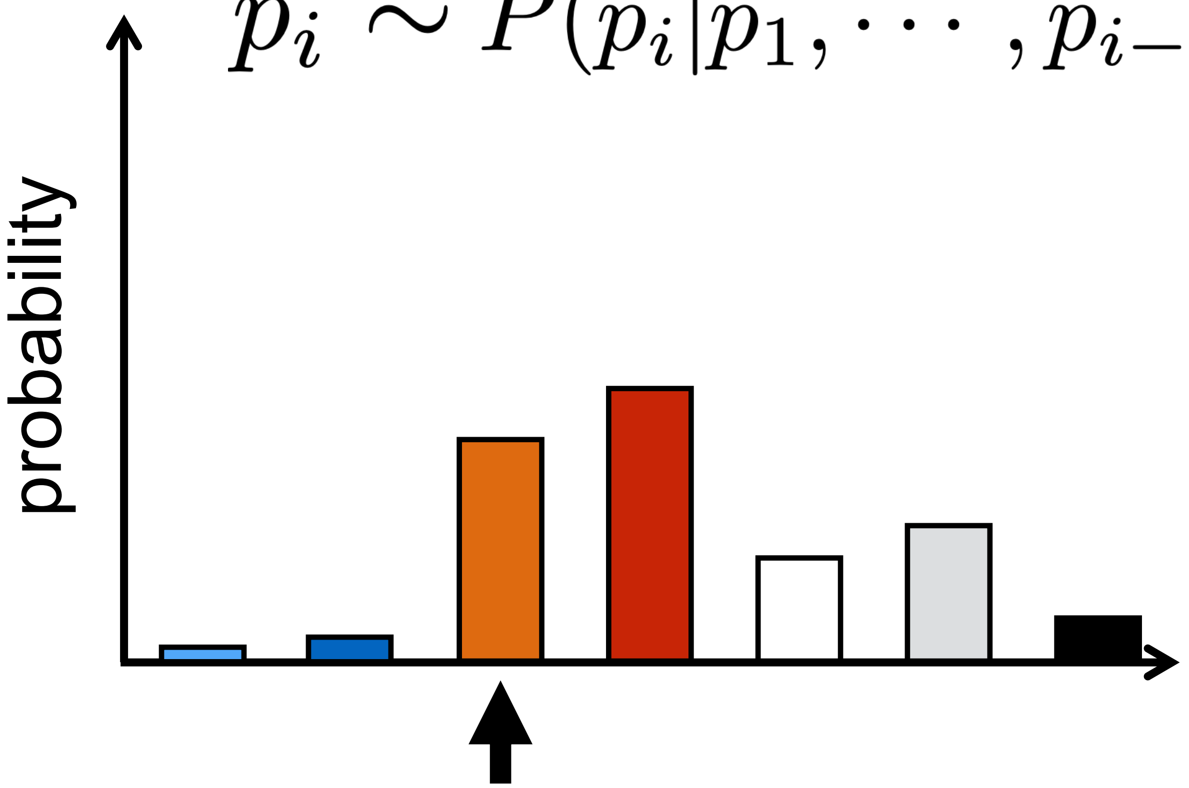


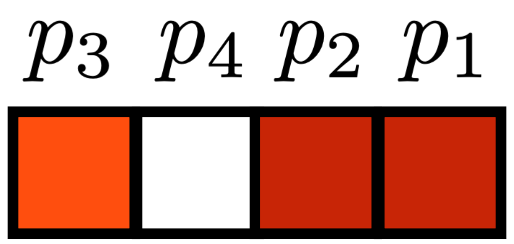


Network output



$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$





$$p_1 \sim P(p_1)$$

$$p_2 \sim P(p_2|p_1)$$

$$p_3 \sim P(p_3|p_1, p_2)$$

$$p_4 \sim P(p_4|p_1, p_2, p_3)$$

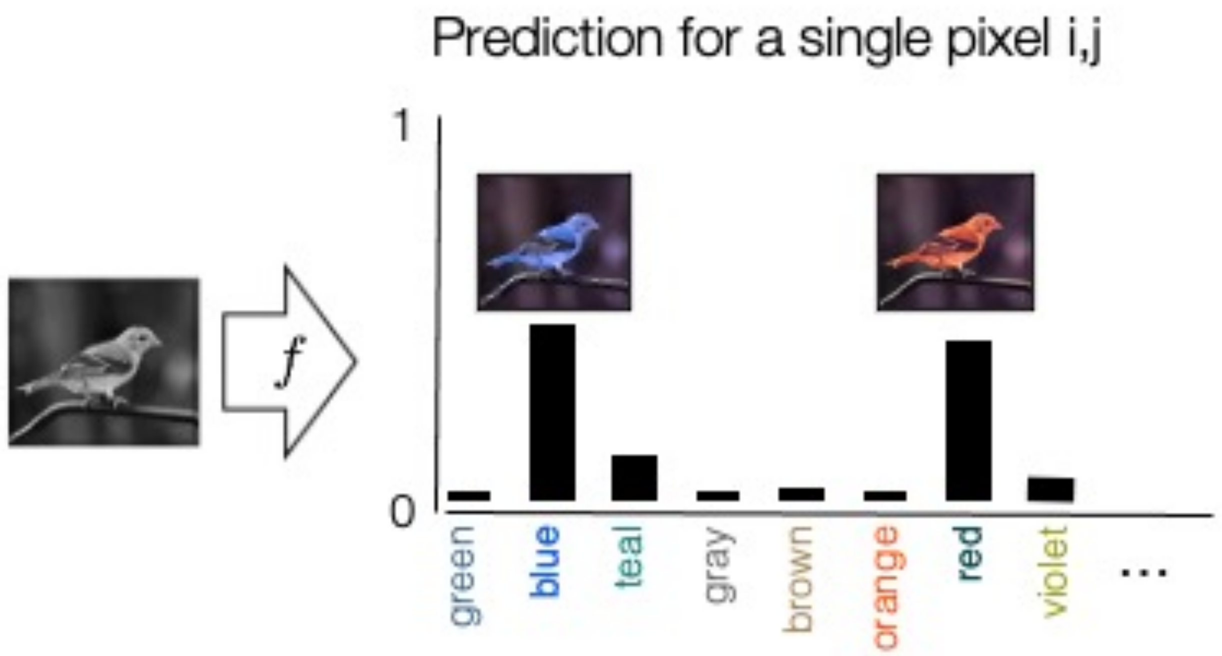
$$\{p_1, p_2, p_3, p_4\} \sim P(p_4|p_1, p_2, p_3)P(p_3|p_1, p_2)P(p_2|p_1)P(p_1)$$

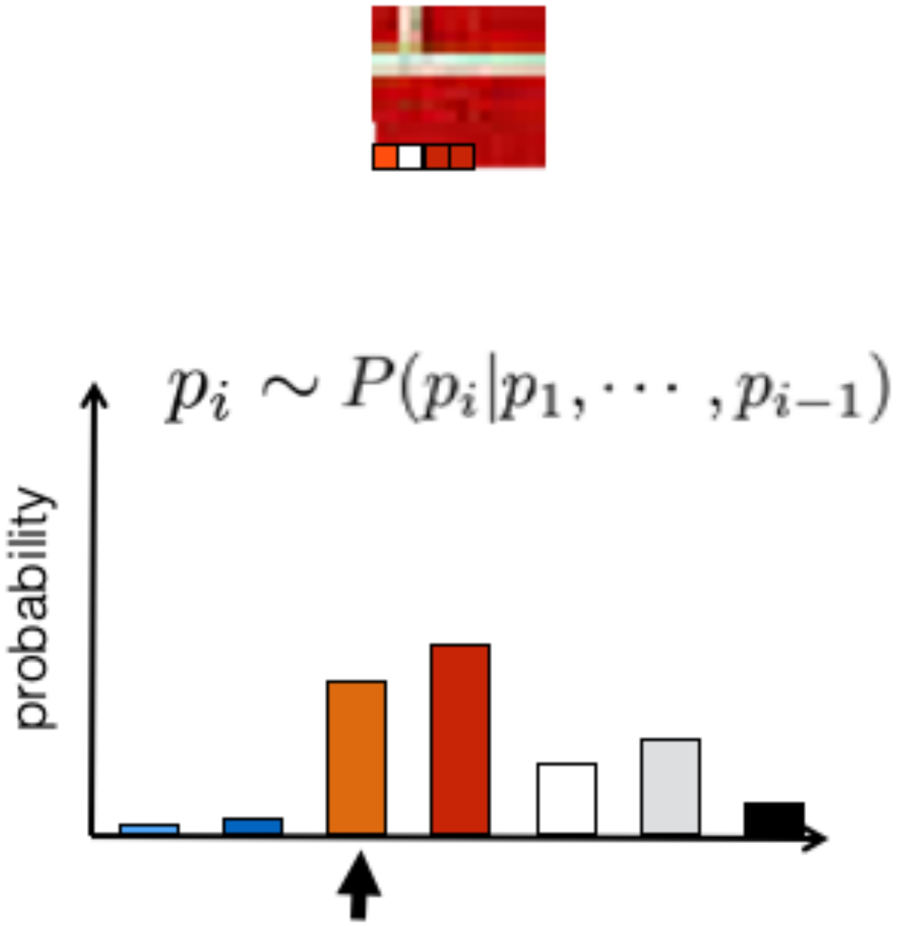
$$p_i \sim P(p_i|p_1, \ldots, p_{i-1})$$

$$\boxed{\mathbf{p} \sim \prod_{i=1}^{N} P(p_i|p_1, \ldots, p_{i-1})}$$

Per-pixel classification vs. Autoregressive

- Image colorization: per-pixel classification loss
- PixelCNN, VQ-VAE2: autoregressive model
- Key idea: it can only produce discrete representation (e.g., VQ codes, color bins)





Autoregressive model

$$\mathbf{p} \sim \prod_{i=1}^{N} P(p_i | p_1, \ldots, p_{i-1})$$

$$P(\mathbf{p}) = \prod_{i=1}^{N} P(p_i | p_1, \ldots, p_{i-1}) \longleftarrow \text{General product rule}$$

The sampling procedure we defined above takes exact samples from the learned probability distribution.

Multiplying all conditionals evaluates the probability of a full joint configuration of pixels.

Autoregressive model

Data $\rightarrow$

Learner

Objective

$$\max_{\theta} \mathbb{E}_{x \sim p_{\texttt{data}}}[\log p_\theta(x)]$$

Hypothesis space

$$p_\theta(x) = \prod_{i=1}^{N} p_\theta^{\texttt{auto}}(x_i | x_1, \ldots, x_{i-1})$$

$$x = [f^1(z^1), f^2(f^1, z^2), \ldots, f^n(f^1, \cdots, f^{n-1}, z^n)]$$

$\rightarrow$

Density

$$p_\theta : \mathcal{X} \rightarrow [0, 1]$$

Sampler

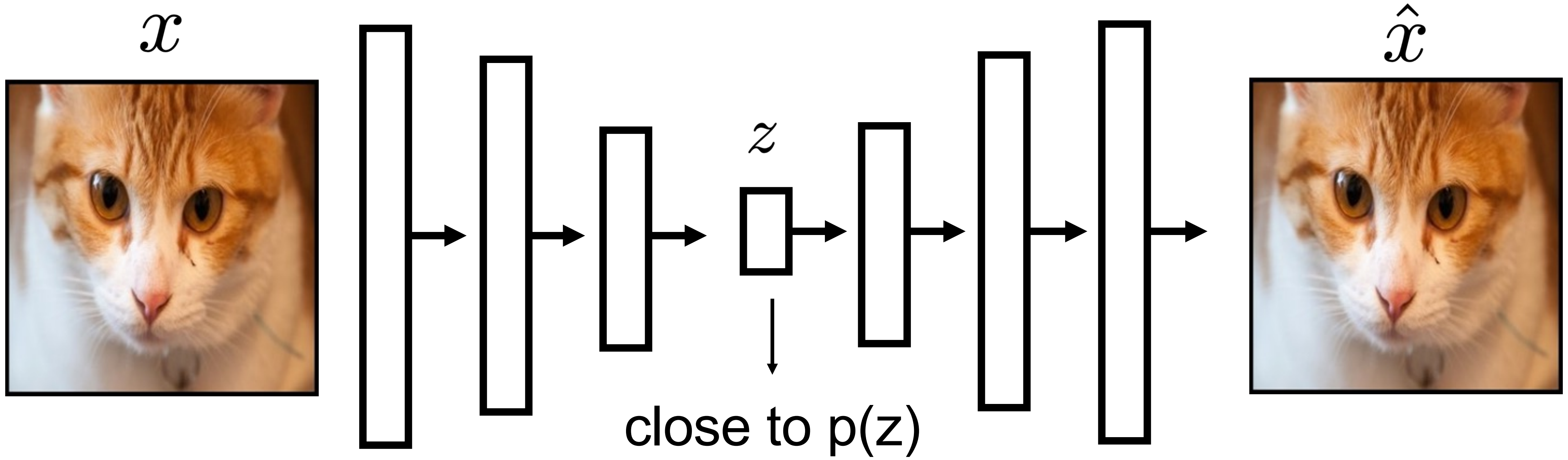
$$f_\theta : \mathcal{Z} \rightarrow \mathcal{X}$$

Variational Auto-encoder

Variational Autoencoders (VAEs)

encoder $q_\psi(z|x)$ $z = E^\mu_\psi(x) + E^\sigma_\psi(x) \cdot \epsilon_z$

generator $p_\theta(x|z)$ $\hat{x} = G^\mu_\theta(z)$



ELBO: Evidence Lower Bound

Multi-variate Gaussian

$$\max_\theta \mathbb{E}_{x\sim p_{\texttt{data}}}[\log p_\theta(x)]$$

$$\geq \max_{\theta,\psi} \mathbb{E}_{x_i\sim p_{\texttt{data}}}[\mathbb{E}_{q_\psi(z|x_i)}[p_\theta(x|z)] - \mathrm{KL}(q_\psi(z|x_i)||p(z))]$$

reconstruction loss

KLD loss

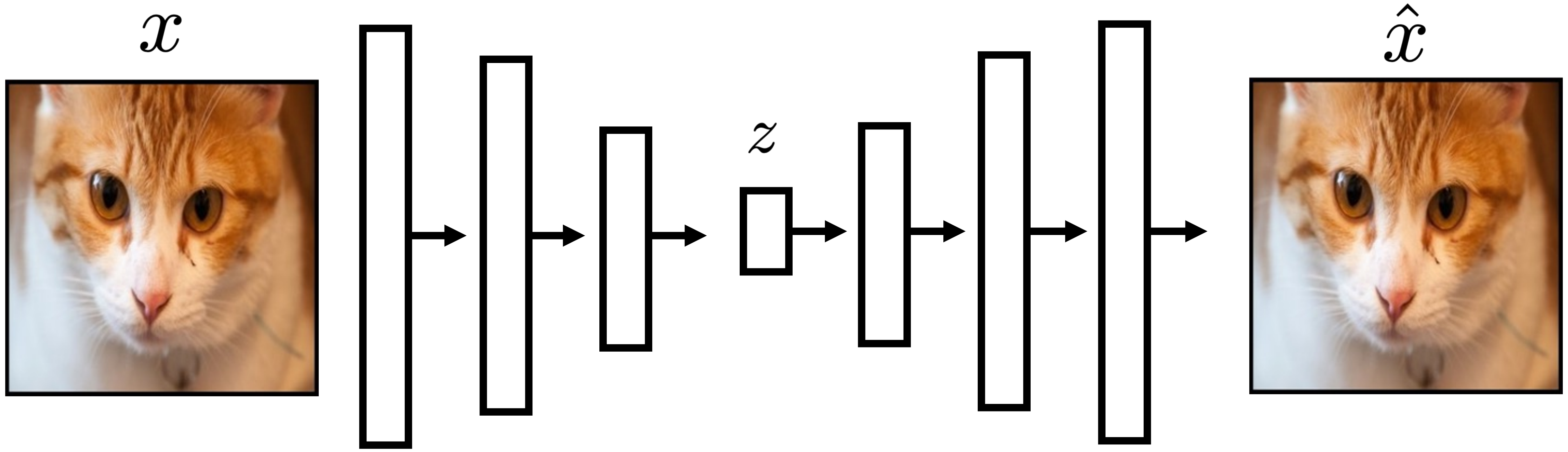
$$||x-\hat{x}||_2 \quad \mathrm{KLD}(\mathcal{N}(E^\mu_\psi(x), E^\sigma_\psi(x)) \mid \mathcal{N}(0, I))$$

[Kingma and Welling, 2014]

Autoencoders (AEs)

encoder $q_\psi(z|x)$ $z = E^\mu_\psi(x)$

generator $p_\theta(x|z)$ $\hat{x} = G^\mu_\theta(z)$



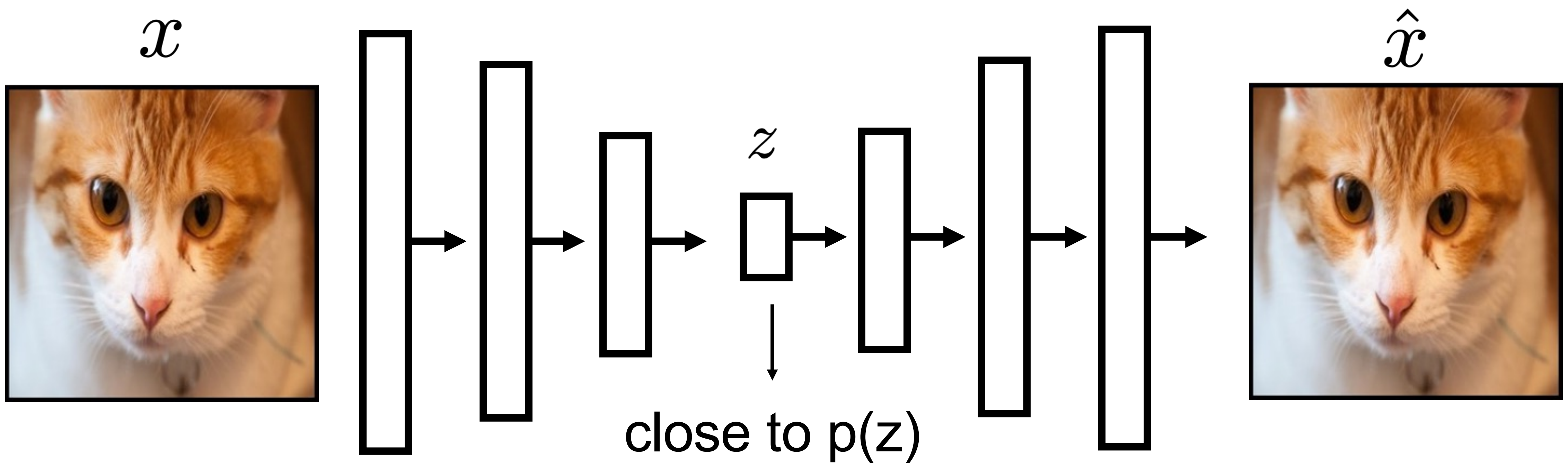
$$\max_{\theta,\psi} \mathbb{E}_{x_i \sim p_{\texttt{data}}}[\mathbb{E}_{q_\psi(z|x_i)}[p_\theta(x|z)]$$

reconstruction loss

$$||x - \hat{x}||_2$$

[Hinton and Salakhutdinov, Science 2006]

Autoencoders (AEs) + Easier Sampling



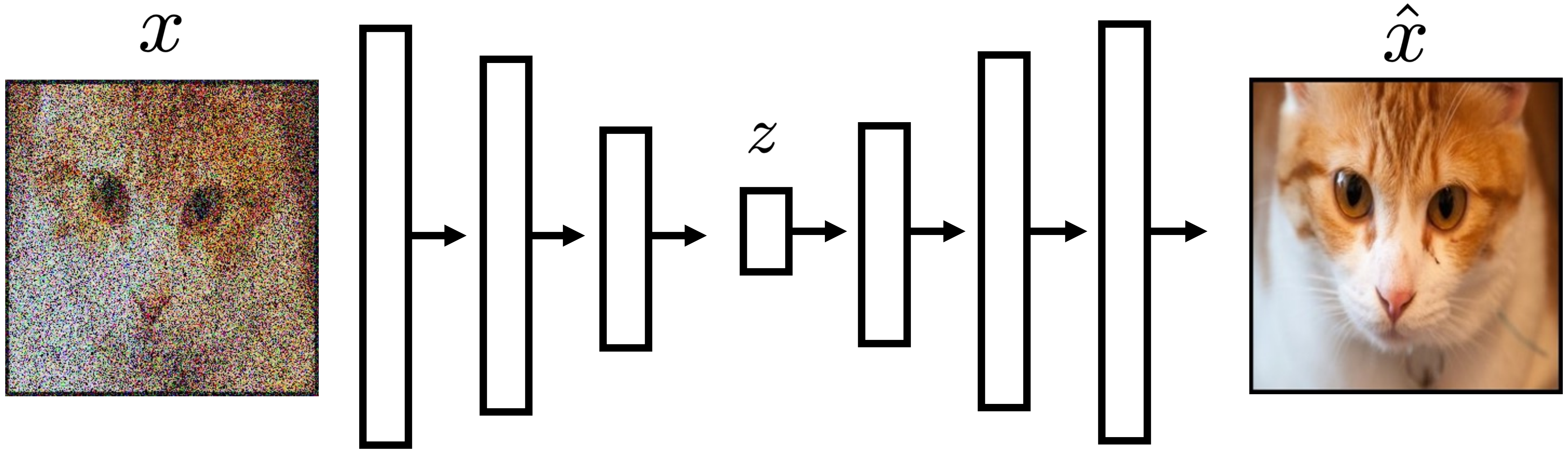
$$\max_{\theta,\psi} \mathbb{E}_{x_i \sim p_{\texttt{data}}} [\mathbb{E}_{q_\psi(z|x_i)} [p_\theta(x|z)] - \mathrm{KL}(q_\psi(z|x_i)||p(z))]$$

reconstruction loss

KLD loss

[Hinton and Salakhutdinov, Science 2006]

Denoising Autoencoders (AEs)



$$\max_{\theta,\psi} \mathbb{E}_{x_i \sim p_{\text{data}}} [\mathbb{E}_{q_\psi(z|x_i)} [p_\theta(x|z)]]$$

reconstruction loss
corrupt input

Denoising vs.
Compression

[Hinton and Salakhutdinov, Science 2006]

Variational Autoencoder (VAE)

Data $\rightarrow$

Learner

Objective

$$\max_{\theta} \mathbb{E}_{x \sim p_{\texttt{data}}}[\log p_{\theta}(x)]$$

Hypothesis space

$$x = G_{\theta}^{\mu}(z) + G_{\theta}^{\sigma}(z)\epsilon$$

Optimizer

Variational Bayes

$\longrightarrow$

Sampler

$$G_{\theta} : \mathcal{Z} \rightarrow \mathcal{X}$$

Flow-based Models

Flow-based models

Learner

Data $\rightarrow$

Objective

$$\max_{\theta} \mathbb{E}_{x \sim p_{\texttt{data}}} [\log p_\theta(x)]$$

Hypothesis space

Bijective: f and $G = f^{-1}$

$\longrightarrow$

Sampler

$$f_\theta(z) : \mathcal{Z} \rightarrow \mathcal{X}$$

Density

$$p_\theta(x) : \mathcal{X} \rightarrow [0, 1]$$

- x and z have the same number of dimensions (memory; training speed)
- limited choices of f and G

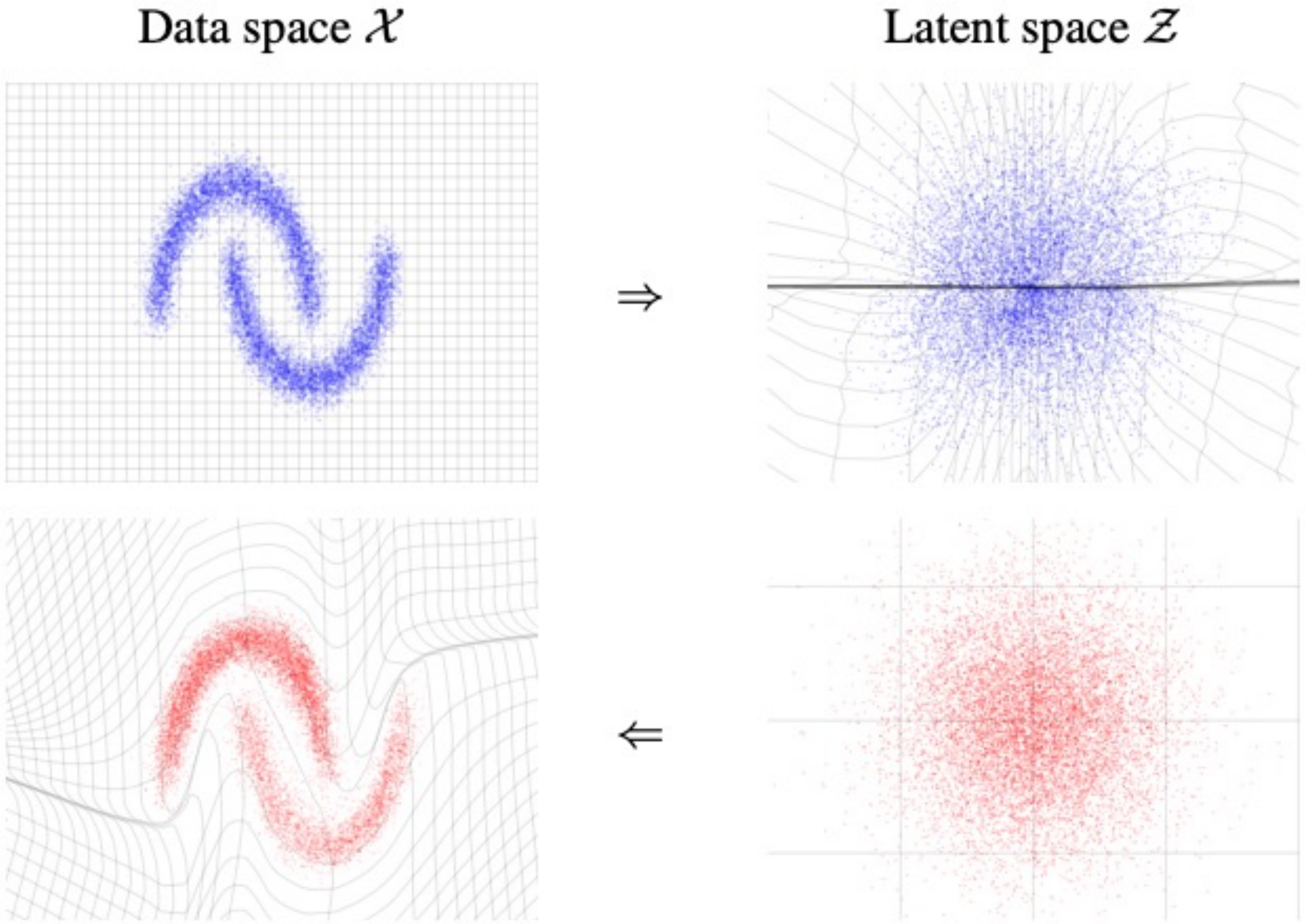
\+ Fast sample; accurate density estimate

[Dinh et al., 2016]

Flow-based models

- Density estimate

$$x \sim p_{data}(x)$$
$$z = f(x)$$



- Sampling

$$z \sim p(z)$$
$$x = G(z)$$

Generator $G = f^{-1}$

[Dinh et al., 2016]

Flow-based models

- Density estimate

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

$$z = f(x)$$

- Sampling

$$z \sim p(z)$$

$$x = G(z)$$

Generator $G = f^{-1}$

Training objective

Change of variable formula

$$p_{\mathrm{data}}(x) = p_z(f(x))|\det(\frac{\partial f(x)}{\partial x^T})|$$

$$\log p_{\mathrm{data}}(x) = \log(p_z(f(x))) + \log(|\det(\frac{\partial f(x)}{\partial x^T})|)$$

Easy to compute
as z follows Gaussian distribution

hard to compute
Jacobian determinant
for most layers

design layers whose Jacobian
is a triangular matrix

[Dinh et al., 2016]

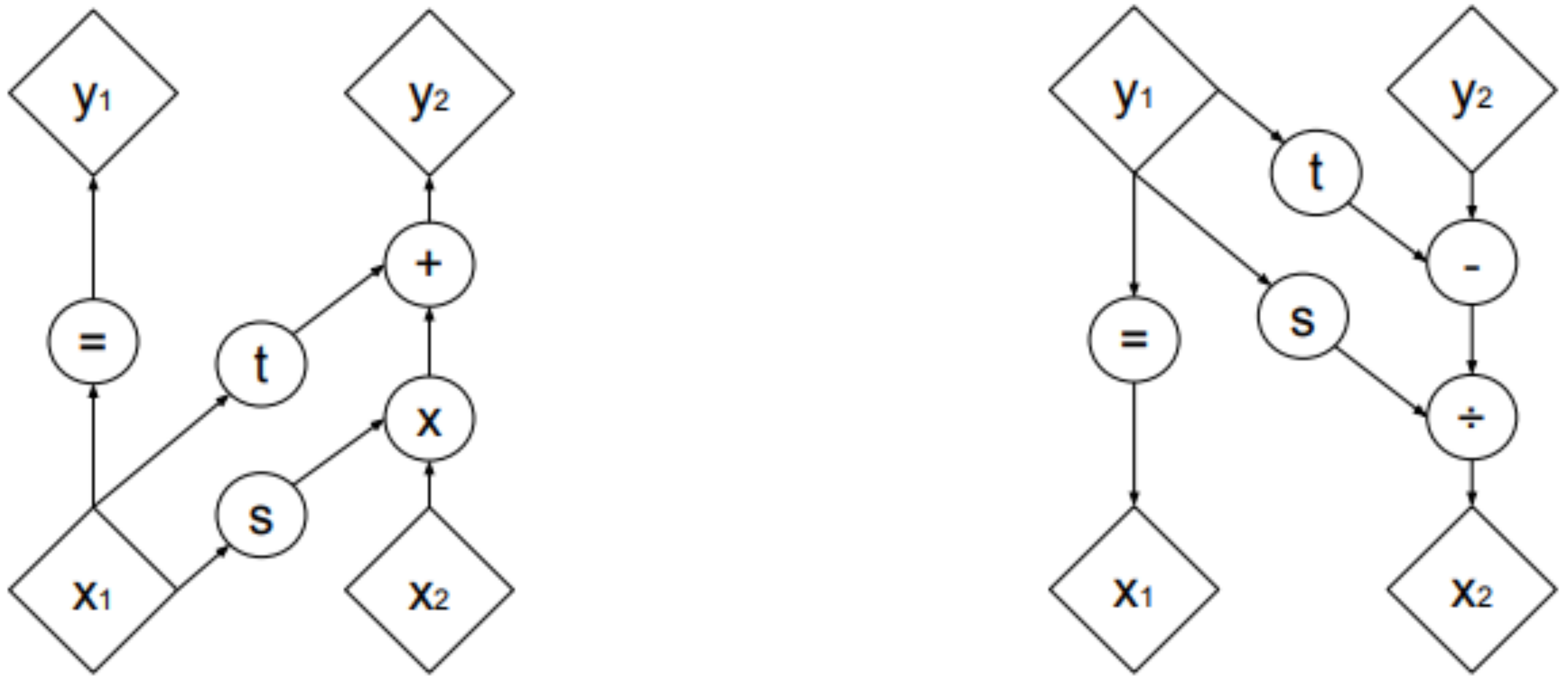
Design invertible layers

$$y_{1:d} = x_{1:d}$$
$$y_{d+1:D} = x_{d+1:D} \odot \exp\big(s(x_{1:d})\big) + t(x_{1:d}),$$

Jacobian matrix

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x^T_{1:d}} & \operatorname{diag}\big(\exp\left[s\left(x_{1:d}\right)\right]\big) \end{bmatrix},$$

s, t: affine transformation
with parameters



(a) Forward propagation (b) Inverse propagation

[Dinh et al., 2016]

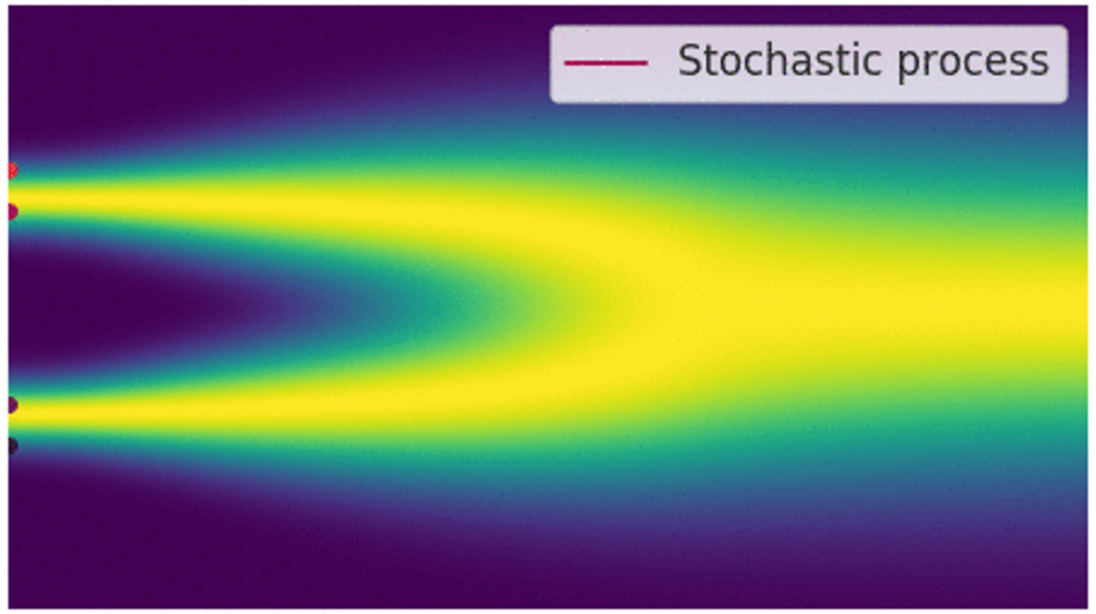
Flow-based Models

Real NVP [Dinh et al., ICLR 2017]

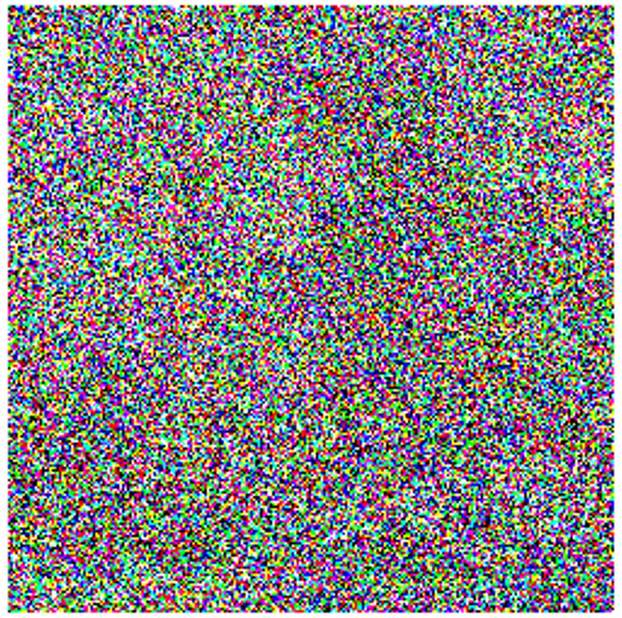
Glow [Kingma and Dhariwal, NeurIPS 2018]

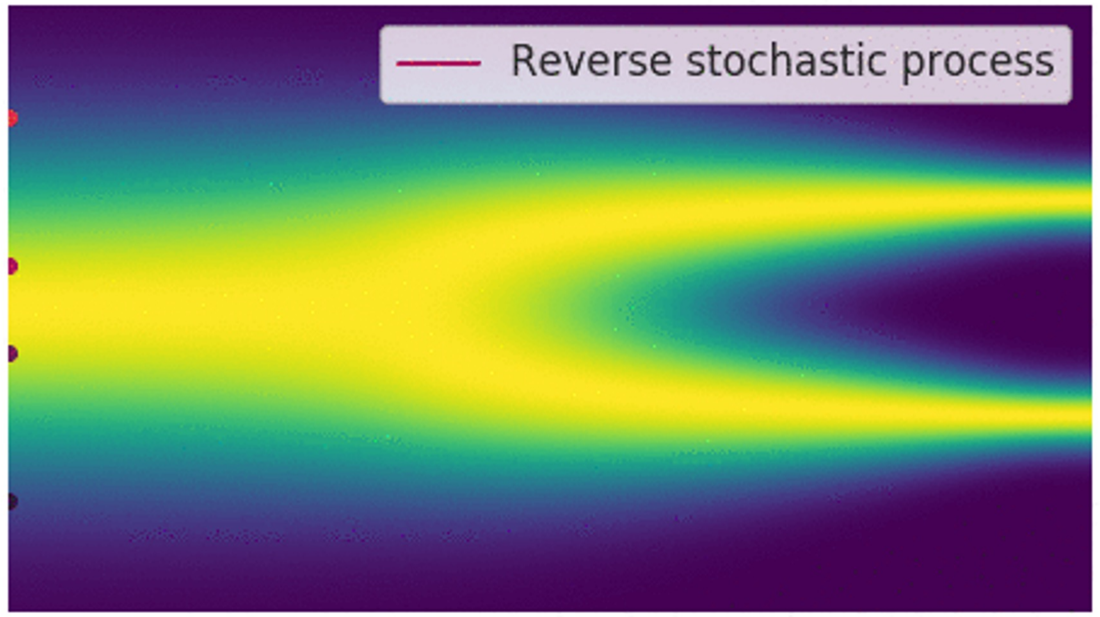
Diffusion Models

Diffusion Models



- "destroy" the data by gradually adding small amounts of gaussian noise





- "create" data by gradually denoising a noisy code from a stationary distribution

Animations from https://yang-song.github.io/blog/2021/score/

Denoising Diffusion Models

Learning to generate by denoising

Denoising diffusion models consist of two processes:

- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising

Forward diffusion process (fixed)

Data  Noise

Reverse denoising process (generative)

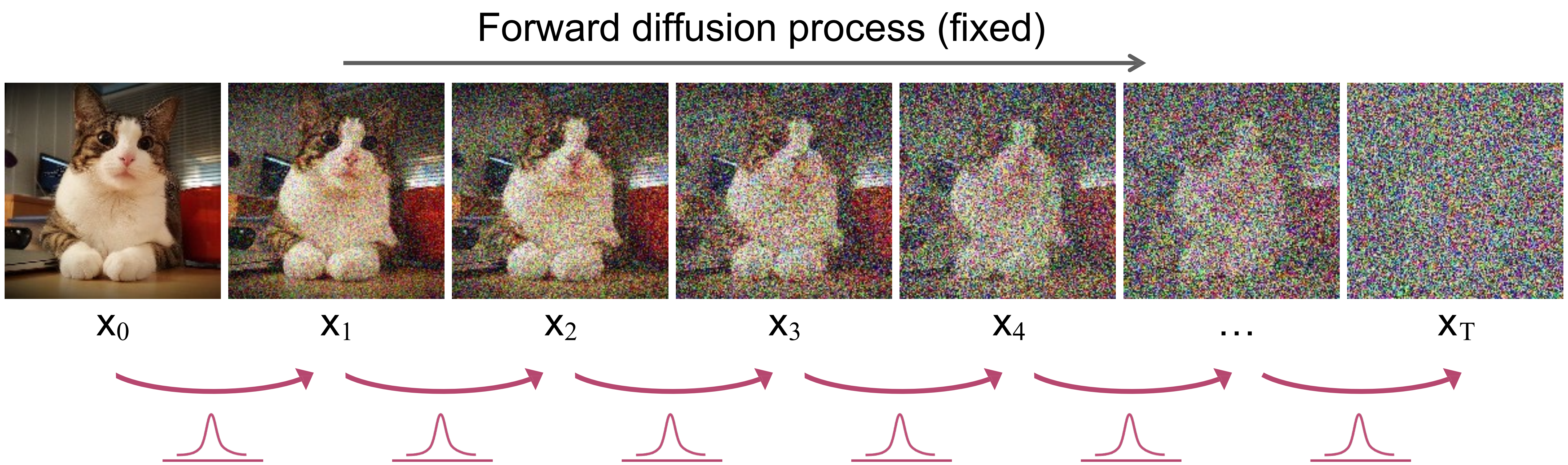
[Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015](#)
[Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020](#)
[Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021](#)

Slide credit: Karsten Kreis et al.

Forward Diffusion Process

The formal definition of the forward process in T steps:

Data

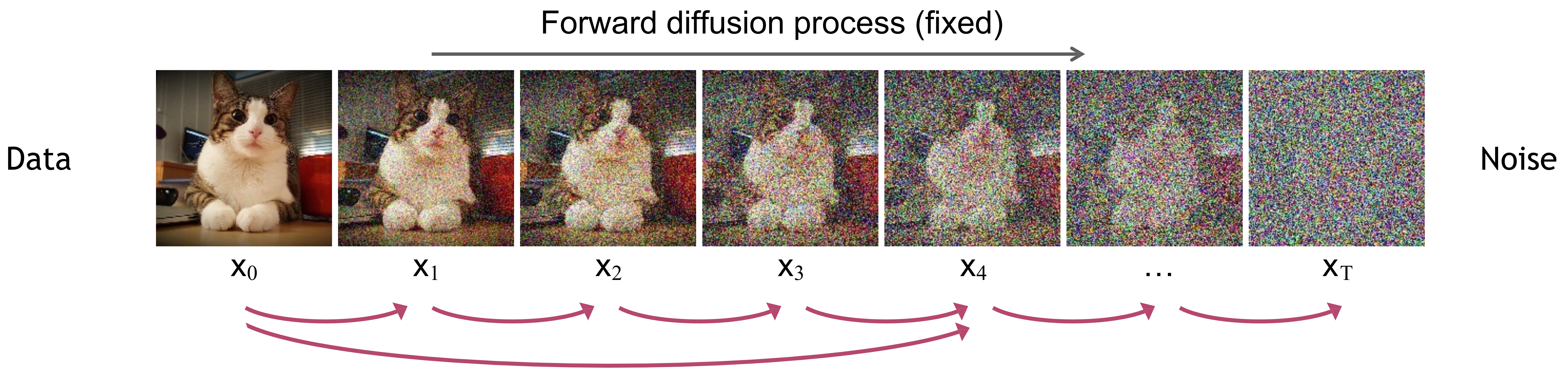


Noise

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

Slide credit: Karsten Kreis et al.

Diffusion Kernel



Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ $\Rightarrow$ $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$ (Diffusion Kernel)

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

β_t values schedule (i.e., the noise schedule) is designed such that $\bar{\alpha}_T \to 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Slide credit: Karsten Kreis et al.

Mathematic details: Merge Multiple Gaussian

- **Reparameterization Trick** $\alpha_t = 1 - \beta_t \quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

"What are Diffusion Models?" Blog

Mathematic details: Merge Multiple Gaussian

- **Reparameterization Trick** $\alpha_t = 1 - \beta_t \qquad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1}$$

“What are Diffusion Models?” Blog

Mathematic details: Merge Multiple Gaussian

- **Reparameterization Trick** $\alpha_t = 1 - \beta_t \qquad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t \alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2}
\end{aligned}
$$

“What are Diffusion Models?” Blog

Mathematic details: Merge Multiple Gaussian

- **Reparameterization Trick** $\alpha_t = 1 - \beta_t$ $\quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}
\end{aligned}
$$

“What are Diffusion Models?” Blog

Mathematic details: Merge Multiple Gaussian

- **Reparameterization Trick** $\alpha_t = 1 - \beta_t \qquad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t \alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}
\end{aligned}
$$

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

Direct sampling from 0 → t

“What are Diffusion Models?” Blog

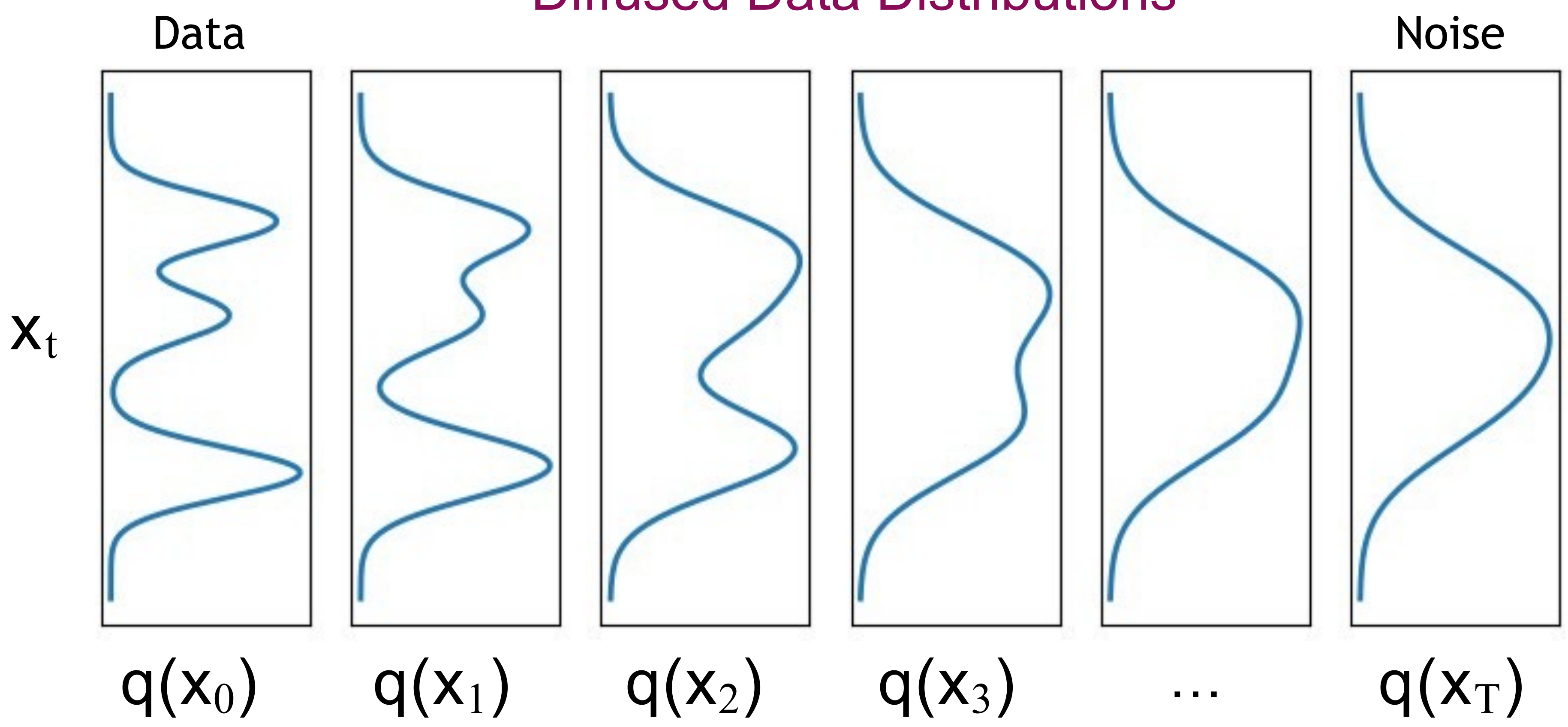
What happens to a distribution in the forward diffusion?

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

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

The diffusion kernel is Gaussian convolution.

Diffused Data Distributions



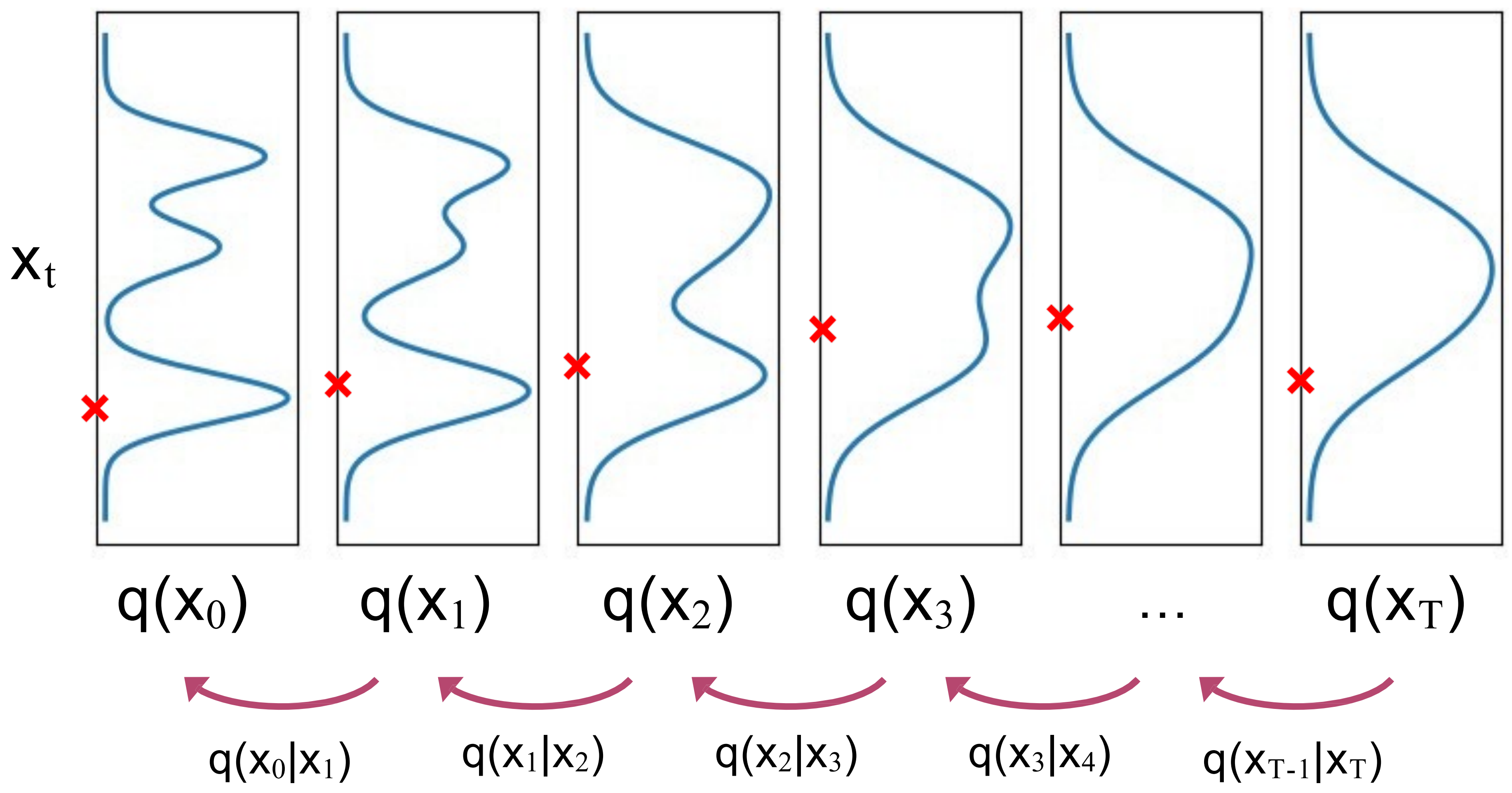
We can sample $\mathbf{x}_t \sim q(\mathbf{x}_t)$ by first sampling $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ and then sampling $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$

Slide credit: Karsten Kreis et al.

Generative Learning by Denoising

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

Diffused Data Distributions



Generation:

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

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

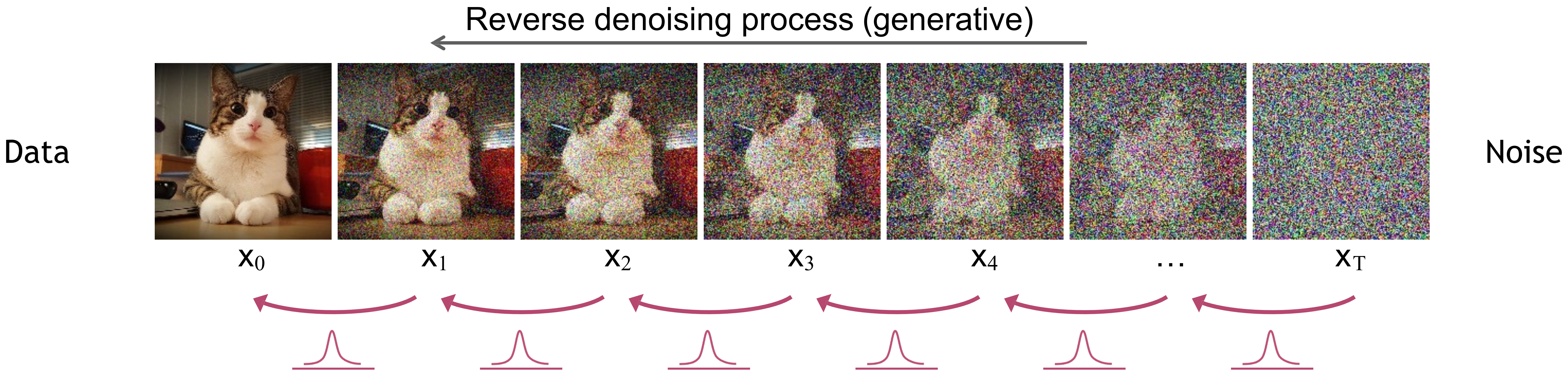
In general, $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is intractable.

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

Slide credit: Karsten Kreis et al.

Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



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

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

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

Slide credit: Karsten Kreis et al.