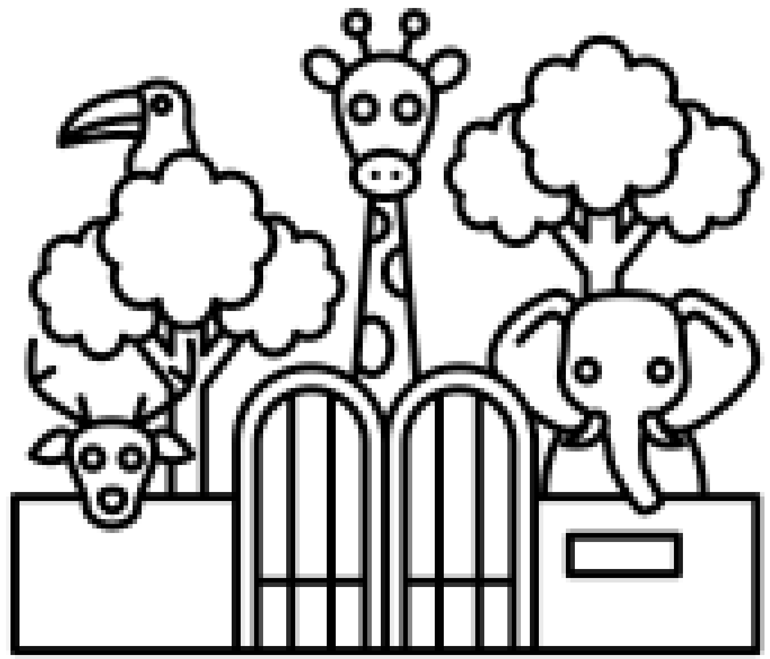


1

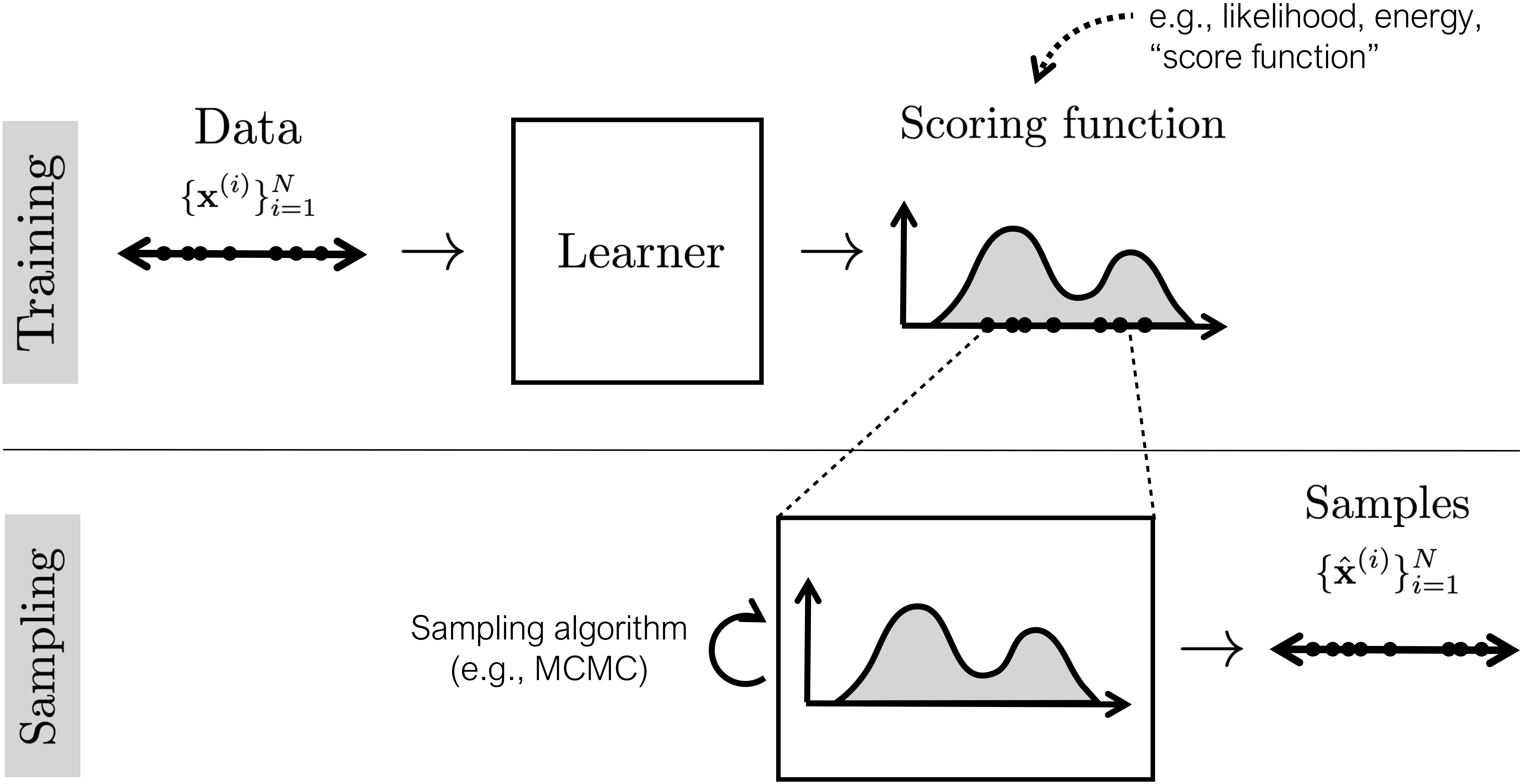
Generative Model Zoo (part II)

Jun-Yan Zhu

16-726 Learning-based Image Synthesis, Spring 2025

many slides from Phillip Isola, Kaiming He, Richard Zhang, Alyosha Efros

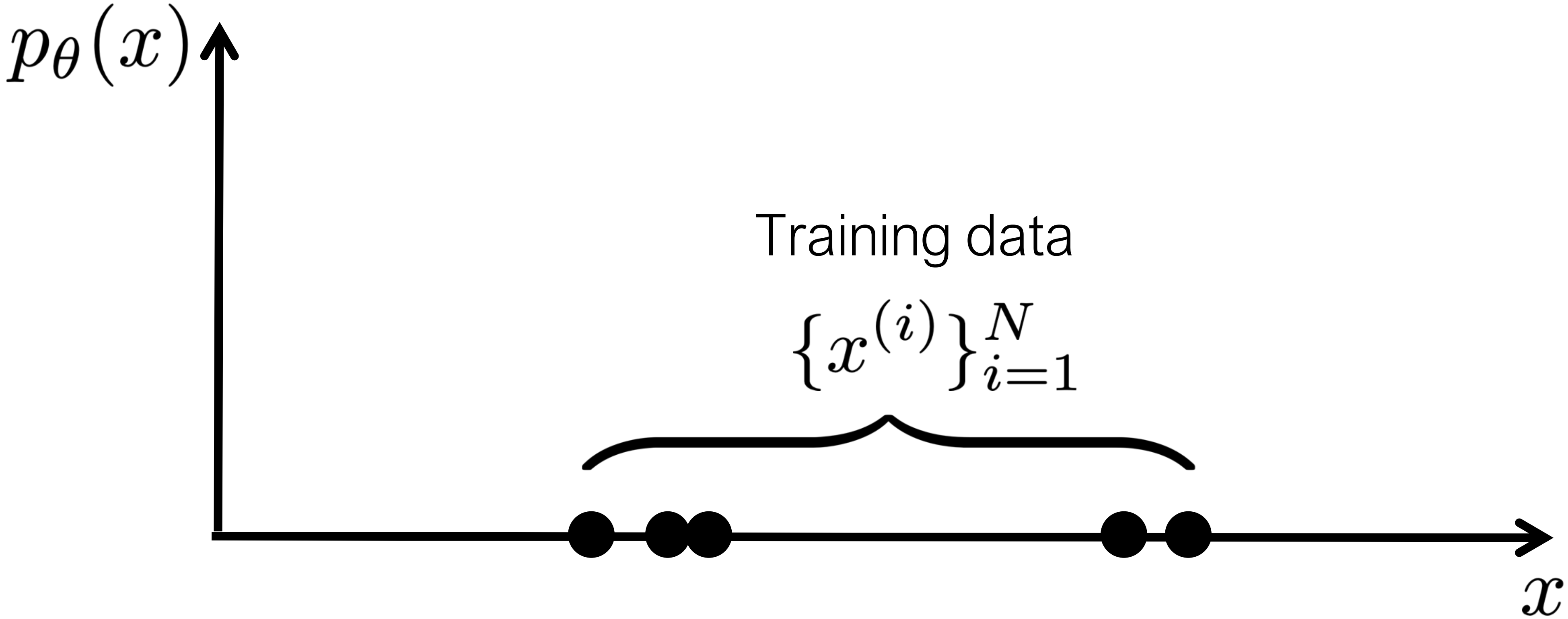
Explicit Density Models



Slides credit: Phillip Isola

Explicit Density Models

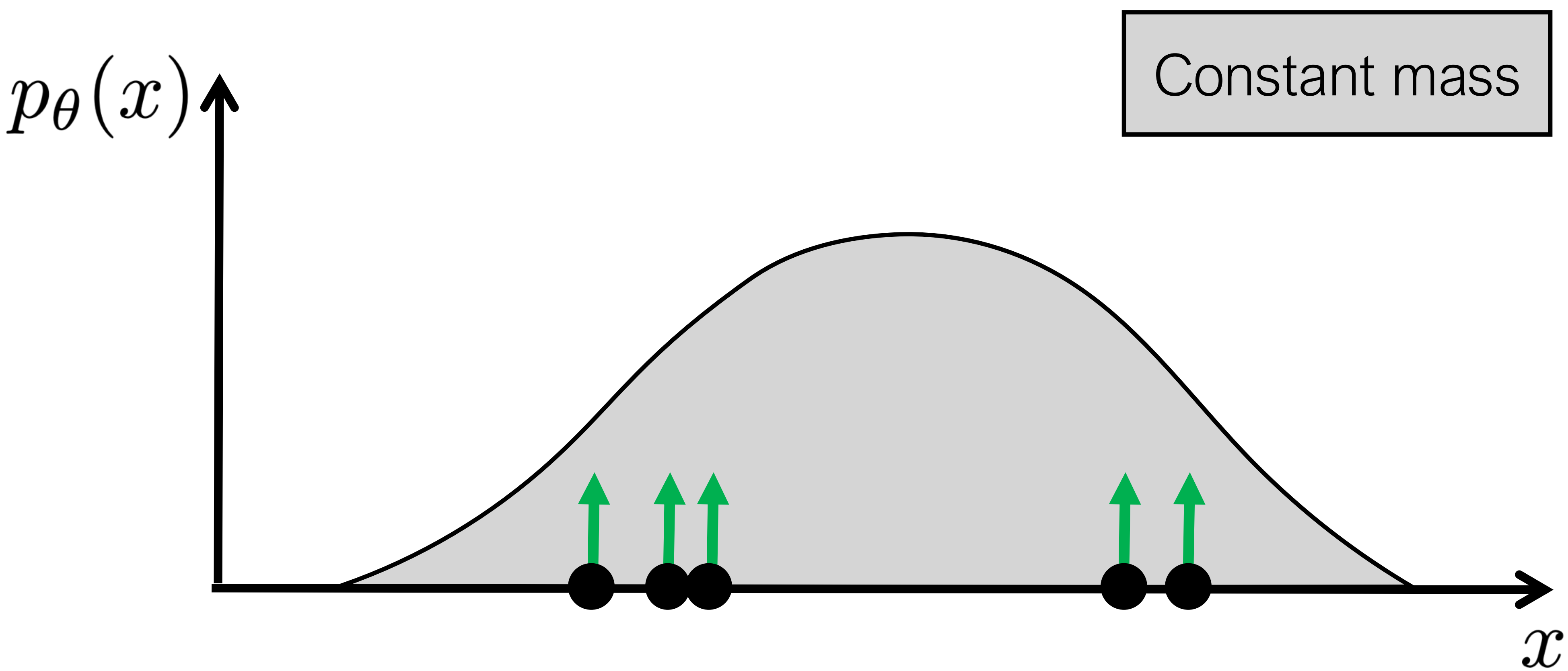
$$p_\theta : \mathcal{X} \to [0, \infty)$$



Slides credit: Phillip Isola

Explicit Density Models

$$p_\theta : \mathcal{X} \rightarrow [0, \infty)$$

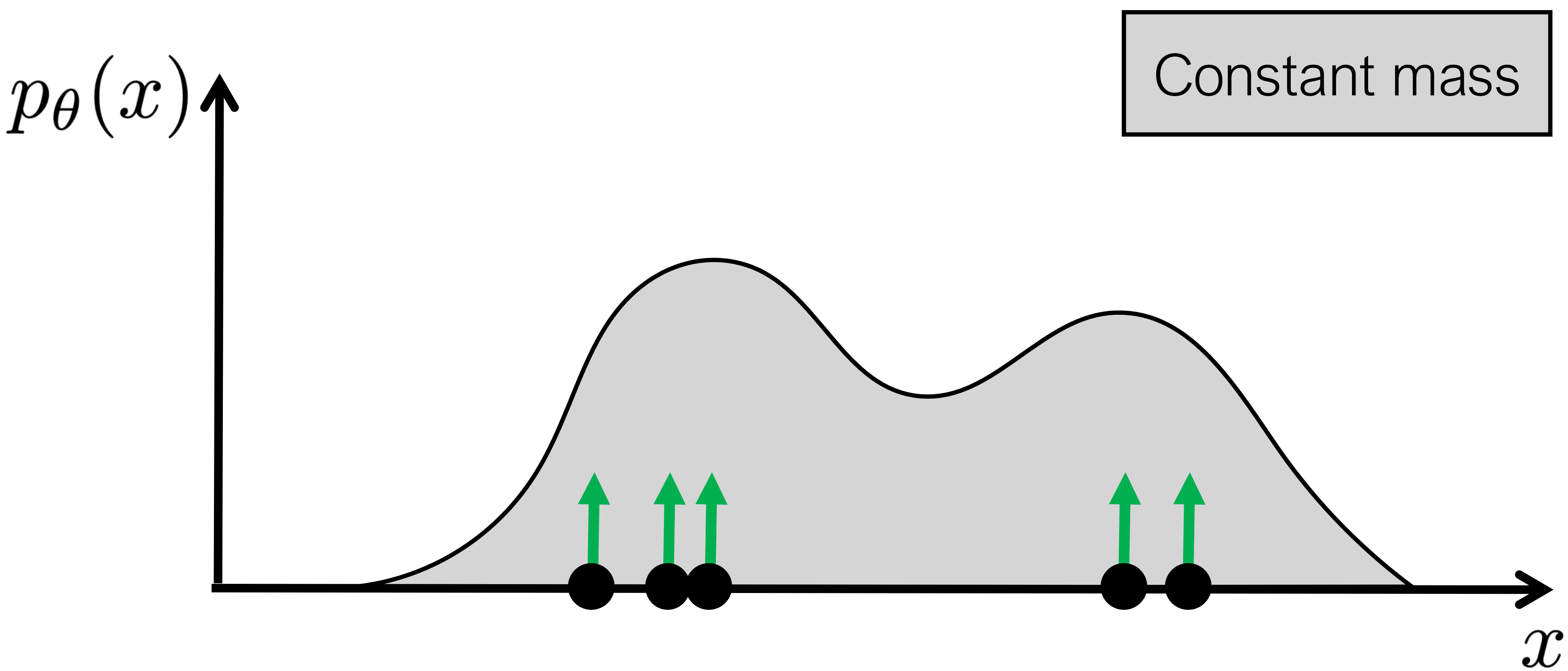


$$\int_x p_\theta(x)dx = 1$$

Slides credit: Phillip Isola

Explicit Density Models

$$p_\theta : \mathcal{X} \to [0, \infty)$$

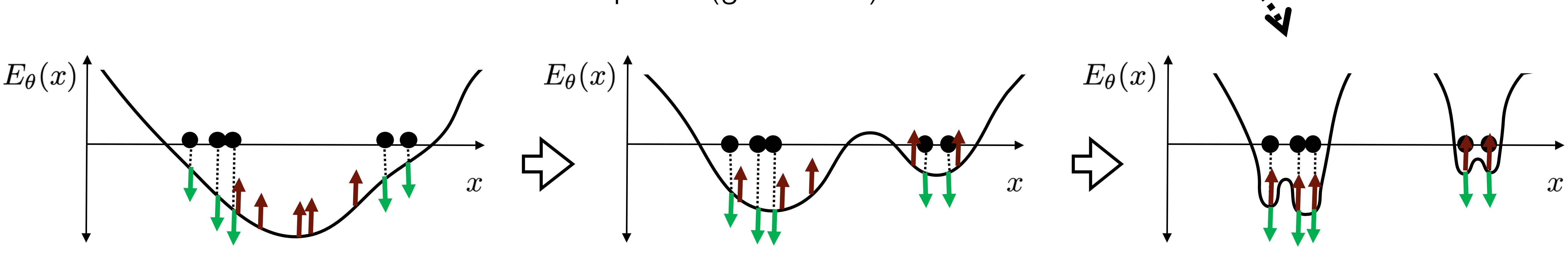


$$\int_x p_\theta(x)dx = 1$$

Slides credit: Phillip Isola

Energy-based models

At convergence, green (data) and red (model) samples are identical and model update (green-red) cancels out



Slides credit: Phillip Isola

Energy-based models — learning algorithm

$$\nabla_\theta \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log p_\theta(\mathbf{x})] = \nabla_\theta \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log \frac{e^{-E_\theta(\mathbf{x})}}{Z(\theta)}]$$

$$= \boxed{-\mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\nabla_\theta E_\theta(\mathbf{x})]} - \underbrace{\boxed{\nabla_\theta \log Z(\theta)}}_{\text{How to measure this?}}$$

Slides credit: Phillip Isola

Energy-based models — learning algorithm

$$\nabla_\theta \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log p_\theta(\mathbf{x})] = \nabla_\theta \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log \frac{e^{-E_\theta(\mathbf{x})}}{Z(\theta)}]$$

$$= \boxed{-\mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\nabla_\theta E_\theta(\mathbf{x})]} - \boxed{\nabla_\theta \log Z(\theta)}$$

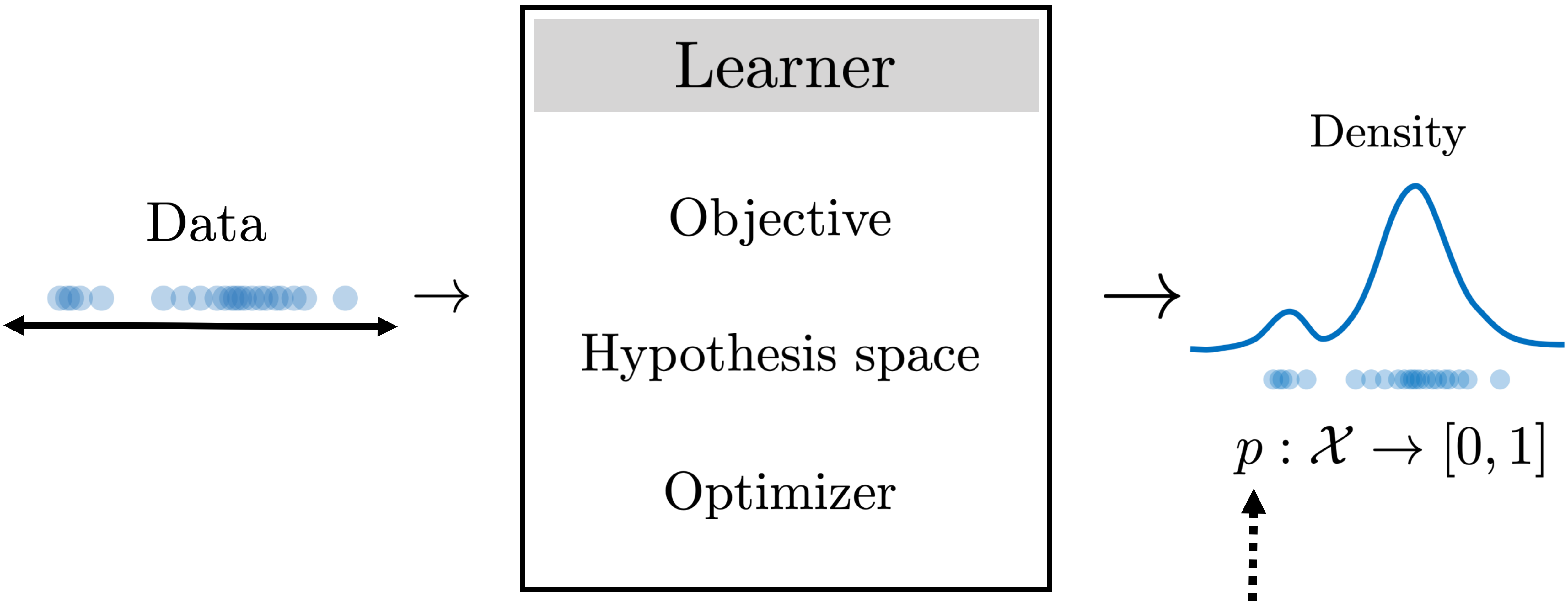
$$= \boxed{-\mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\nabla_\theta E_{\theta(\mathbf{x})}]} + \boxed{\mathbb{E}_{\mathbf{x}\sim p_\theta}[\nabla_\theta E_\theta(\mathbf{x})]}$$

$$\approx \boxed{-\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta E_\theta(\mathbf{x}^{(i)})} + \boxed{\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta E_\theta(\hat{\mathbf{x}}^{(i)})}$$

$\mathbf{x}^{(i)} \sim p_{\texttt{data}}$ $\quad$ $\hat{\mathbf{x}}^{(i)} \sim p_\theta$

Slides credit: Phillip Isola

Learning an Explicit Density Model



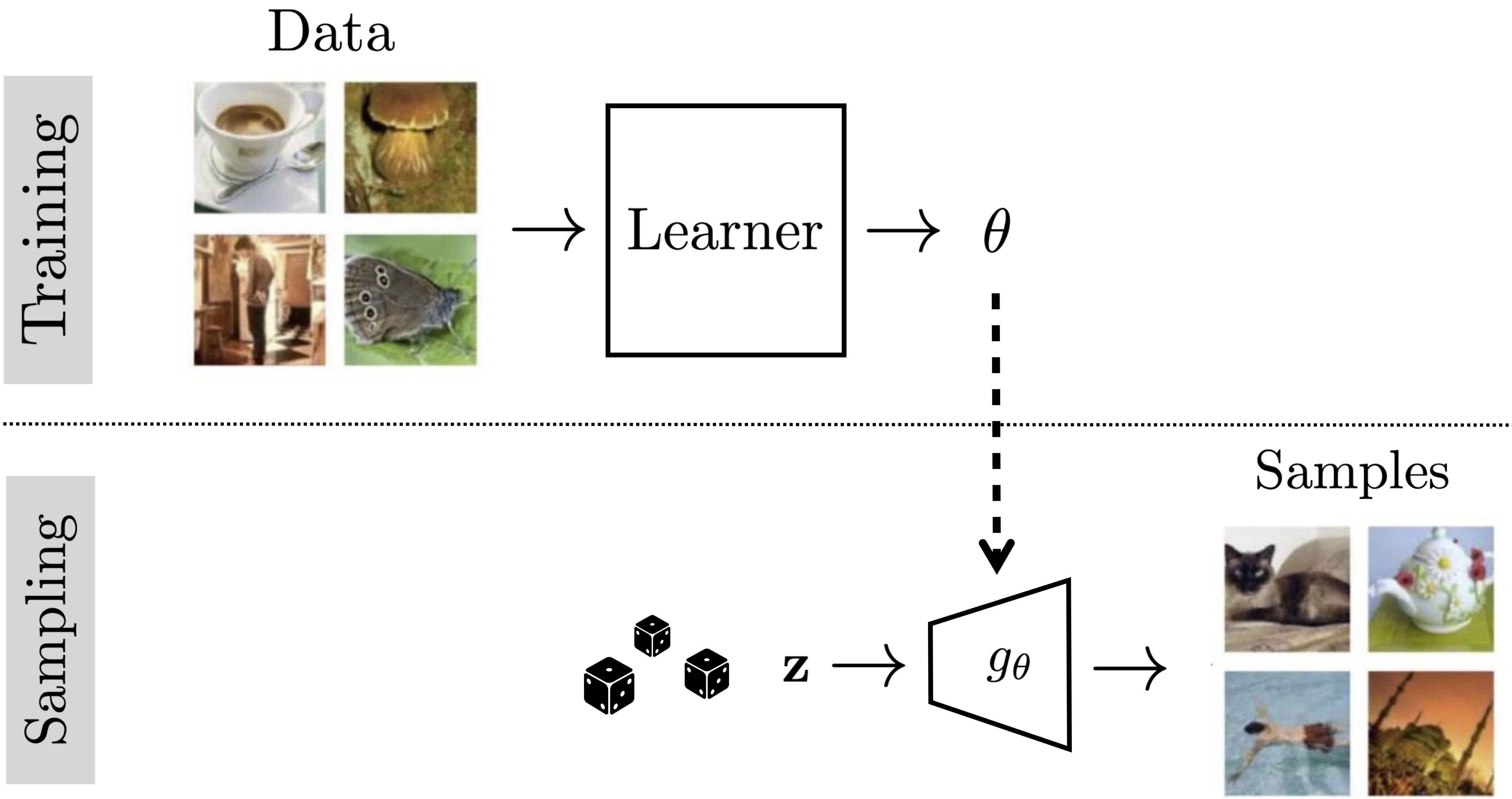
Integral of probability density function needs to be 1 ⟶ Normalized distribution
(some models output unnormalized *energy functions*)

[figs modified from: http://introtodeeplearning.com/materials/2019_6S191_L4.pdf]

Useful for abnormality/outlier detection (detect unlikely events)

Slides credit: Phillip Isola

Implicit Generative Models



Slides credit: Phillip Isola

Data

$\{x^{(i)}\}_{i=1}^{N} \rightarrow$ **Generative modeling** $\rightarrow$

Density function

$p_\theta : \mathcal{X} \rightarrow [0, \infty)$

Energy function

$E_\theta : \mathcal{X} \rightarrow \mathbb{R}$

Generator

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

You can represent the data generating process directly or indirectly

Slides credit: Phillip Isola

Case study #1: Fitting a Gaussian to data

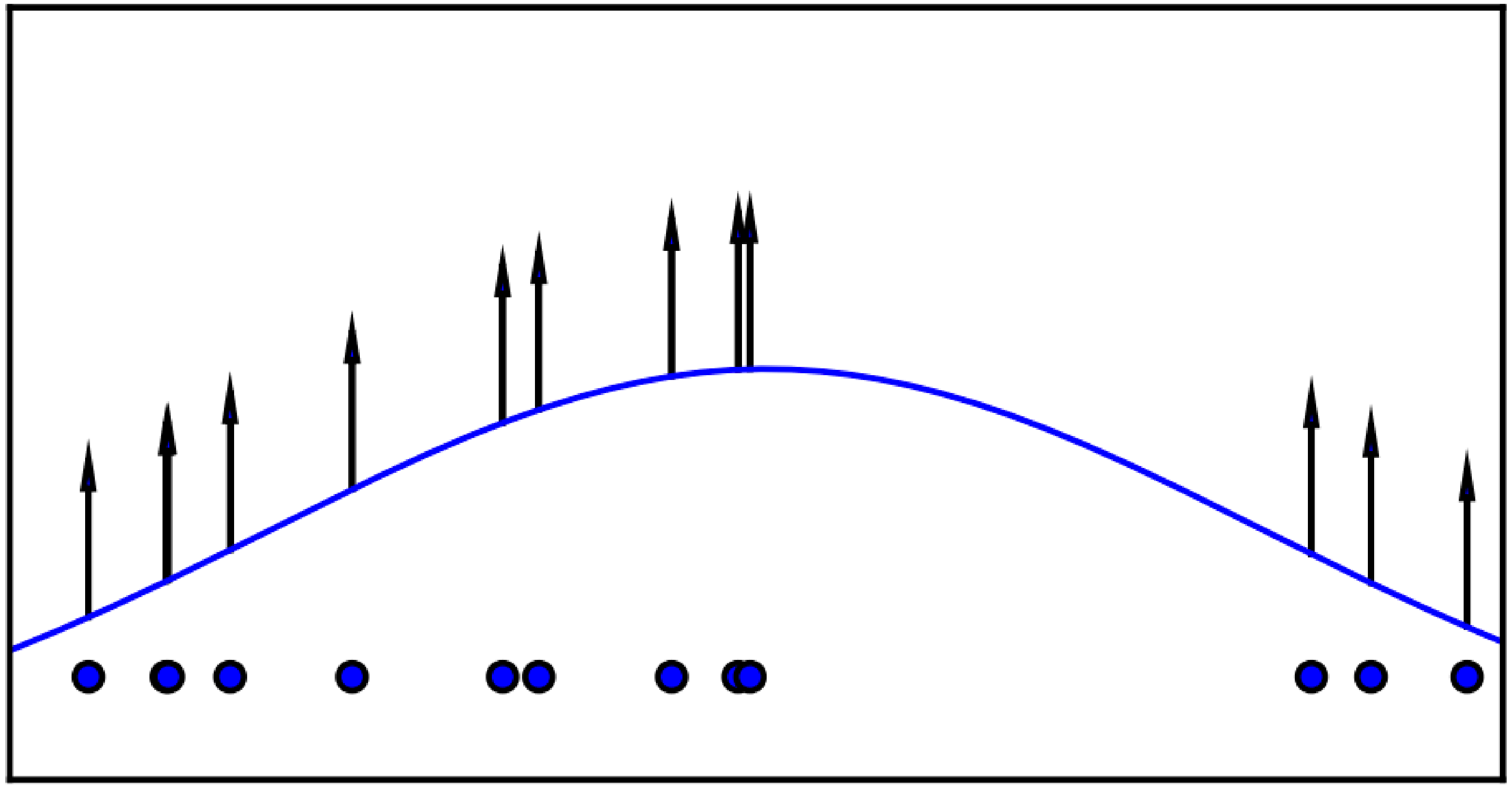


fig from [Goodfellow, 2016]

Max likelihood objective

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

Considering only Gaussian fits

$$p_\theta(x) = \mathcal{N}(x; \mu, \sigma)$$
$$\theta = [\mu, \sigma]$$

Closed form optimum:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Case study #2: Generative Adversarial Network

Data $\rightarrow$

Learner

Objective

$$\min_G \max_D \mathbb{E}_z[\log(1 - D(G(z))] + \mathbb{E}_x[\log D(x)]$$

Hypothesis space

Deep nets G and D

Optimizer

Alternating SGD on G and D

$\rightarrow$

Critic

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

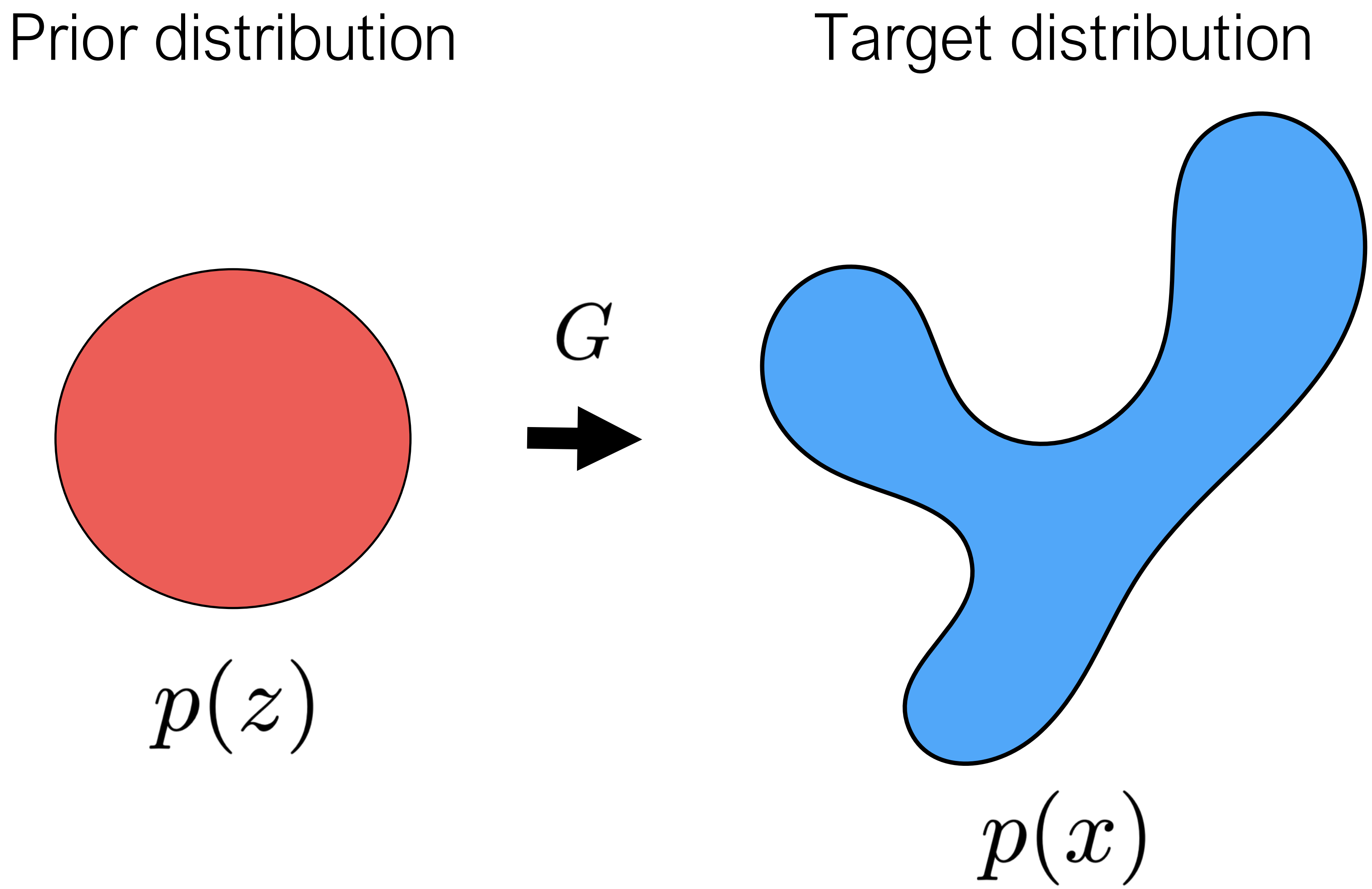
Sampler

$G : \mathcal{Z} \rightarrow \mathcal{X}$

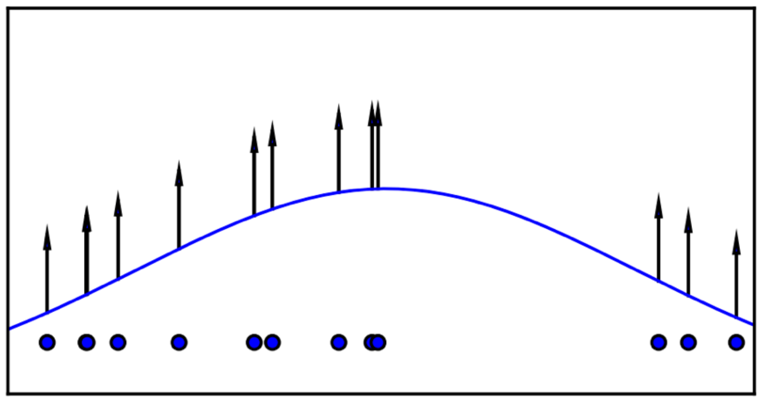
Variational Autoencoder (VAE)

Variational Autoencoders (VAEs)

[Kingma & Welling, 2014; Rezende, Mohamed, Wierstra 2014]

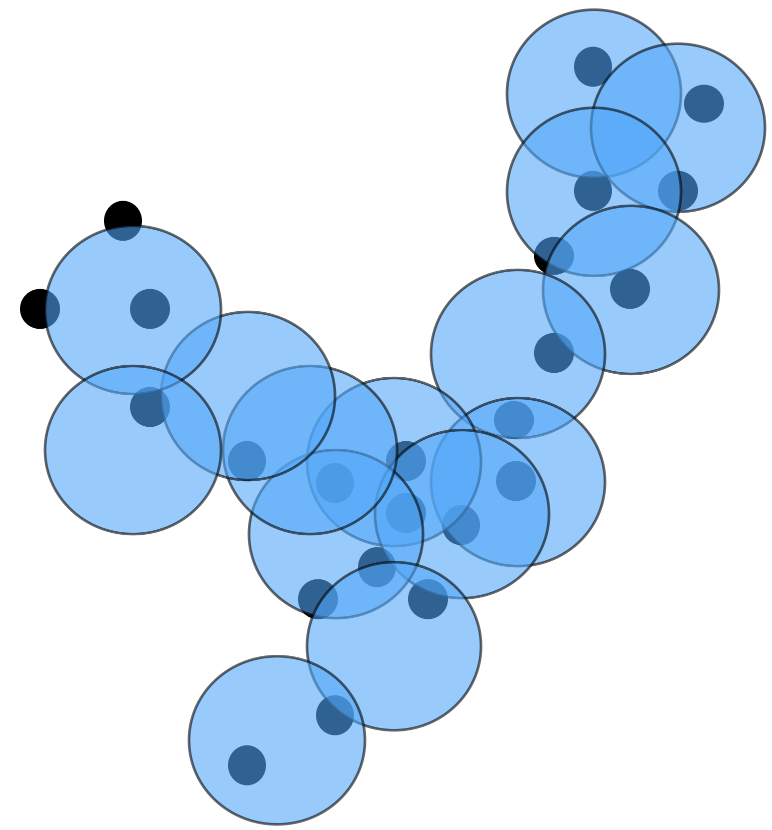


Mixture of Gaussians



$$p_\theta(x) = \sum_{i=1}^{k} w_i \mathcal{N}(x; u_i, \Sigma_i)$$

Target distribution

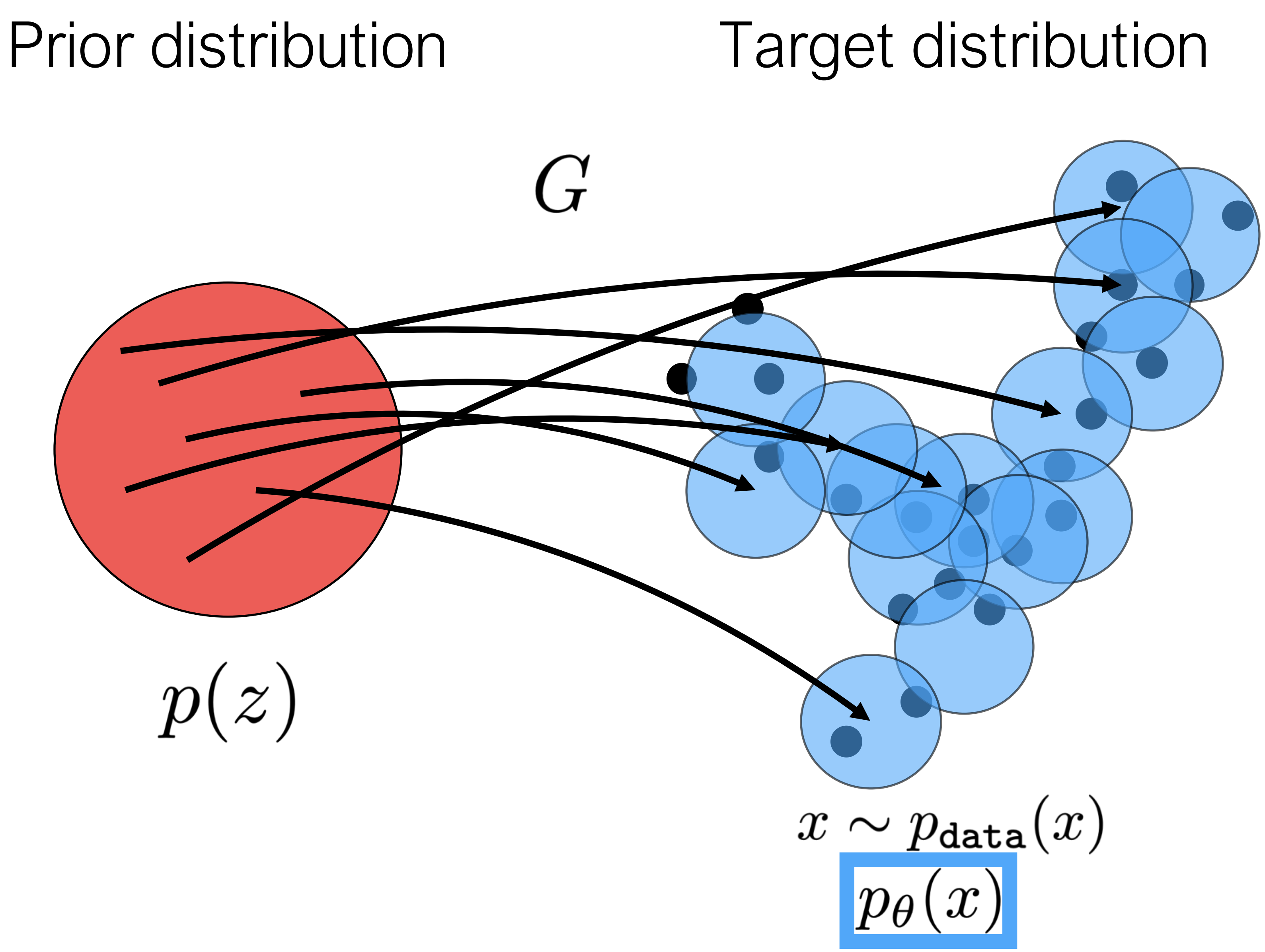


$x \sim p_{\mathtt{data}}(x)$

$\boxed{p_\theta(x)}$

Variational Autoencoders (VAEs)

[Kingma & Welling, 2014; Rezende, Mohamed, Wierstra 2014]



Density model:

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

$$p(x|z;\theta) \sim \mathcal{N}(x; G^\mu_\theta(z), G^\sigma_\theta(z))$$

Sampling:

$$z \sim p(z) \quad \epsilon \sim \mathcal{N}(0,1)$$

$$x = G^\mu_\theta(z) + G^\sigma_\theta(z)\epsilon$$

Variational Autoencoder (VAE)

Data $\rightarrow$

Learner

Objective

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

Hypothesis space

$$\cancel{p_\theta(x) = \int p(x|z;\theta)p(z)dz}$$

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

$\rightarrow$

~~Density~~

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

Sampler

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

Variational Autoencoders (VAEs)

Fitting a model to data requires computing $p_\theta(x)$

How to compute $p_\theta(x)$ efficiently?

$$p_\theta(x) = \int p(x|z;\theta)p(z)dz \longleftarrow$$ almost all terms are near zero

Train "inference network" $q_\psi(z|x)$
to give distribution over the z's that are likely to produce x

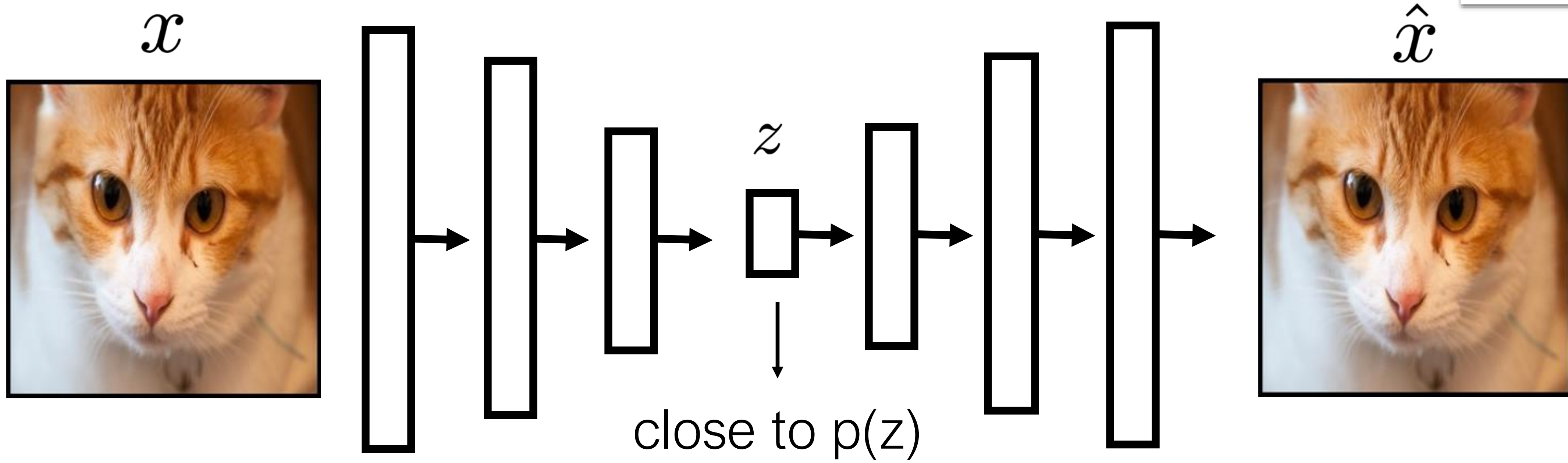
Approximate $p_\theta(x)$ with $\mathbb{E}_{q_\psi(z|x)}[p_\theta(x|z)]$

[Kingma and Welling, 2014]

Tutorial on VAEs [Doersch, 2016]

Variational Autoencoders (VAEs)

encoder $q_\psi(z|x)$ $\quad z = E^{\mu}_{\psi}(x) + E^{\sigma}_{\psi}(x) \cdot \epsilon_z \qquad$ generator $p_\theta(x|z)$ $\quad \hat{x} = G^{\mu}_{\theta}(z)$



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

Multi-variate Gaussian

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

reconstruction loss

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

KLD loss

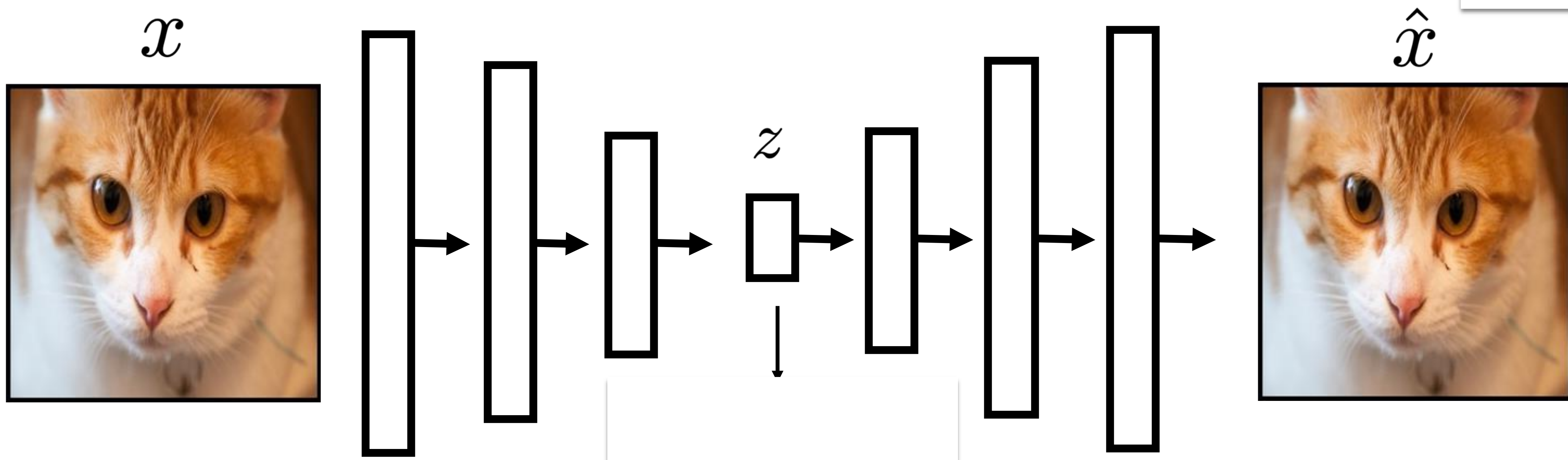
$$\mathrm{KLD}(\mathcal{N}(E^{\mu}_{\psi}(x), E^{\sigma}_{\psi}(x)) \mid \mathcal{N}(0, I))$$

[Kingma and Welling, 2014]

Autoencoders (AEs)

encoder $q_\psi(z|x)$ $z = E^{\mu}_{\psi}(x)$

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



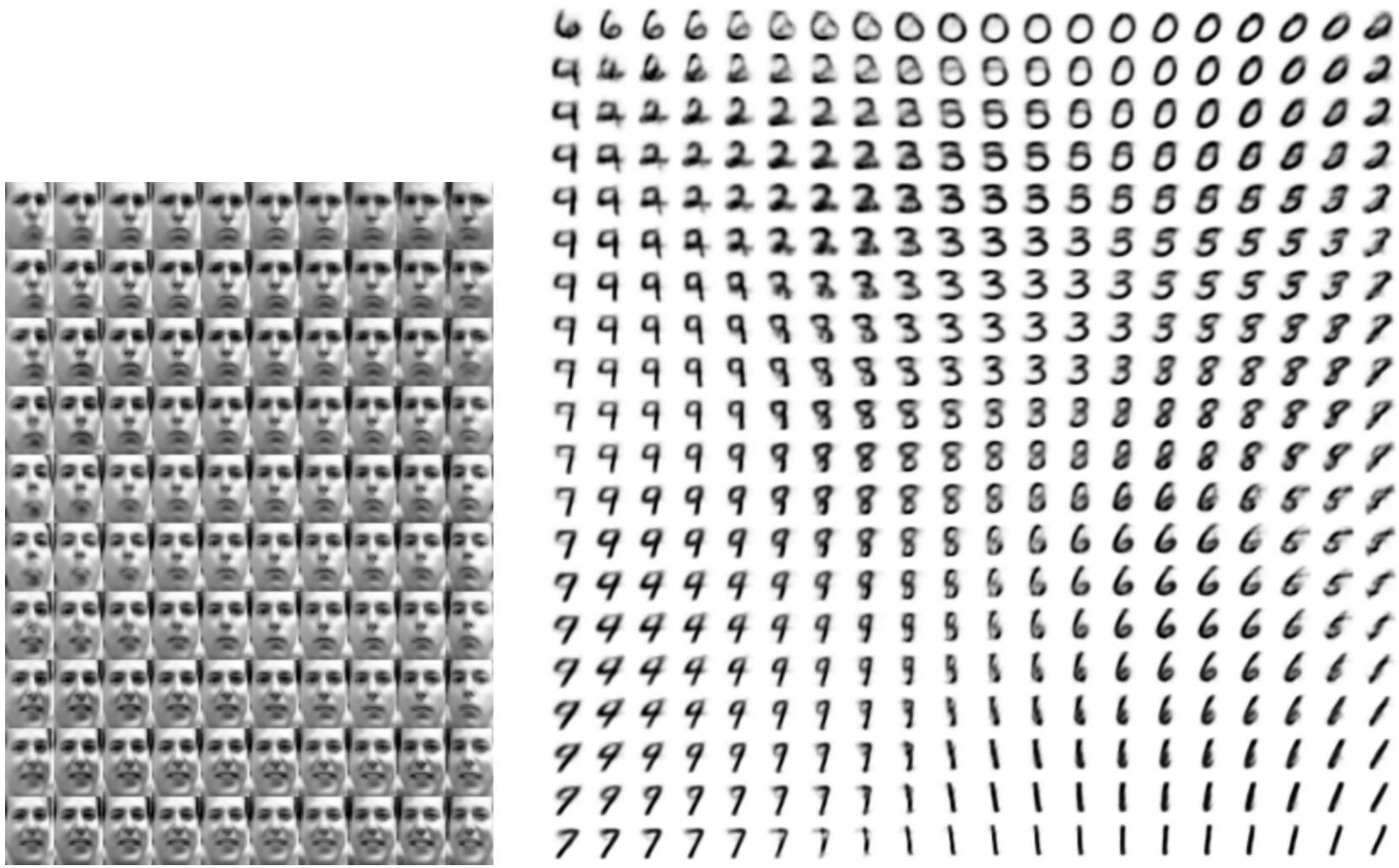
$$\max_{\theta,\psi} \mathbb{E}_{x_i \sim p_{\text{data}}}[\mathbb{E}_{q_\psi(z|x_i)}[p_\theta(x|z)]]$$

reconstruction loss

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

[Hinton and Salakhutdinov, Science 2006]

Variational Autoencoders (VAEs)



VAE with two-dimensional latent space

[Kingma and Welling, 2014]

How to improve VAE?

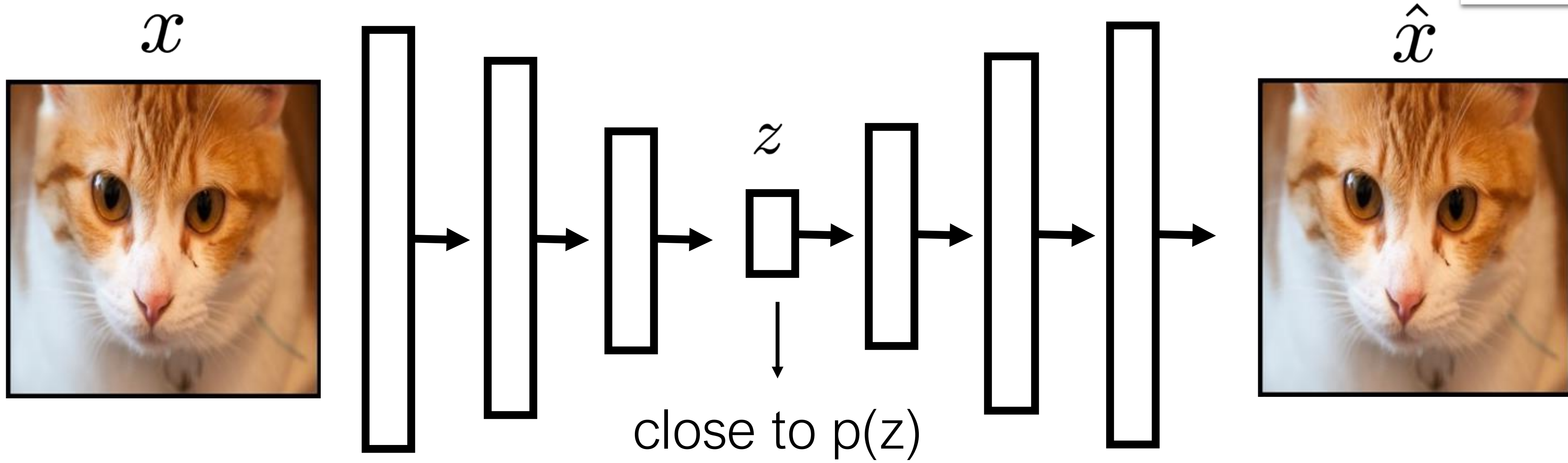
- Why are the results blurry?
  - L2 reconstruction loss?
  - Lower bound might not be tight?

- How can we further improve results?

VAE + Perceptual Loss

encoder $q_\psi(z|x)$ $z = E^{\mu}_{\psi}(x) + E^{\sigma}_{\psi}(x) \cdot \epsilon_z$

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



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

Multi-variate Gaussian

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

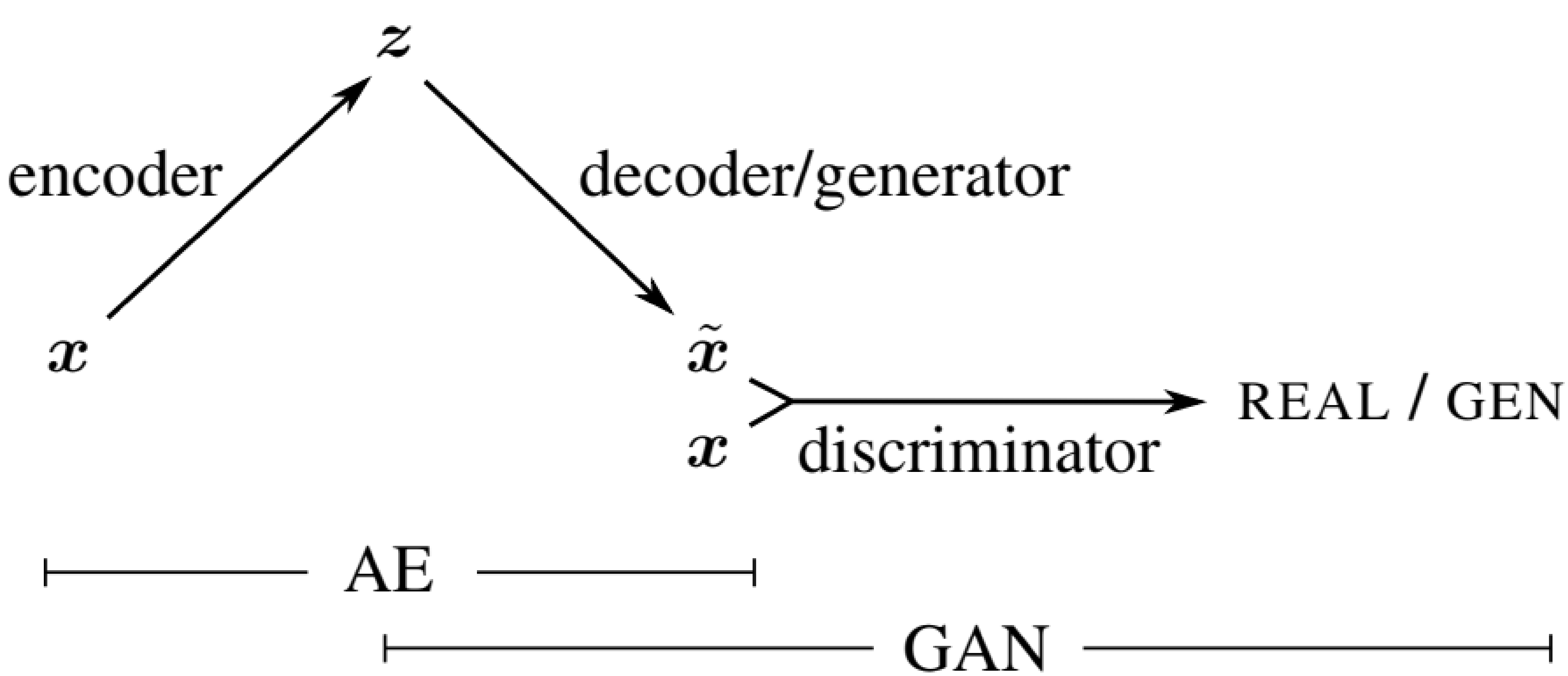
Perceptual loss

$$||F(x) - F(\hat{x})||_2$$

KLD loss

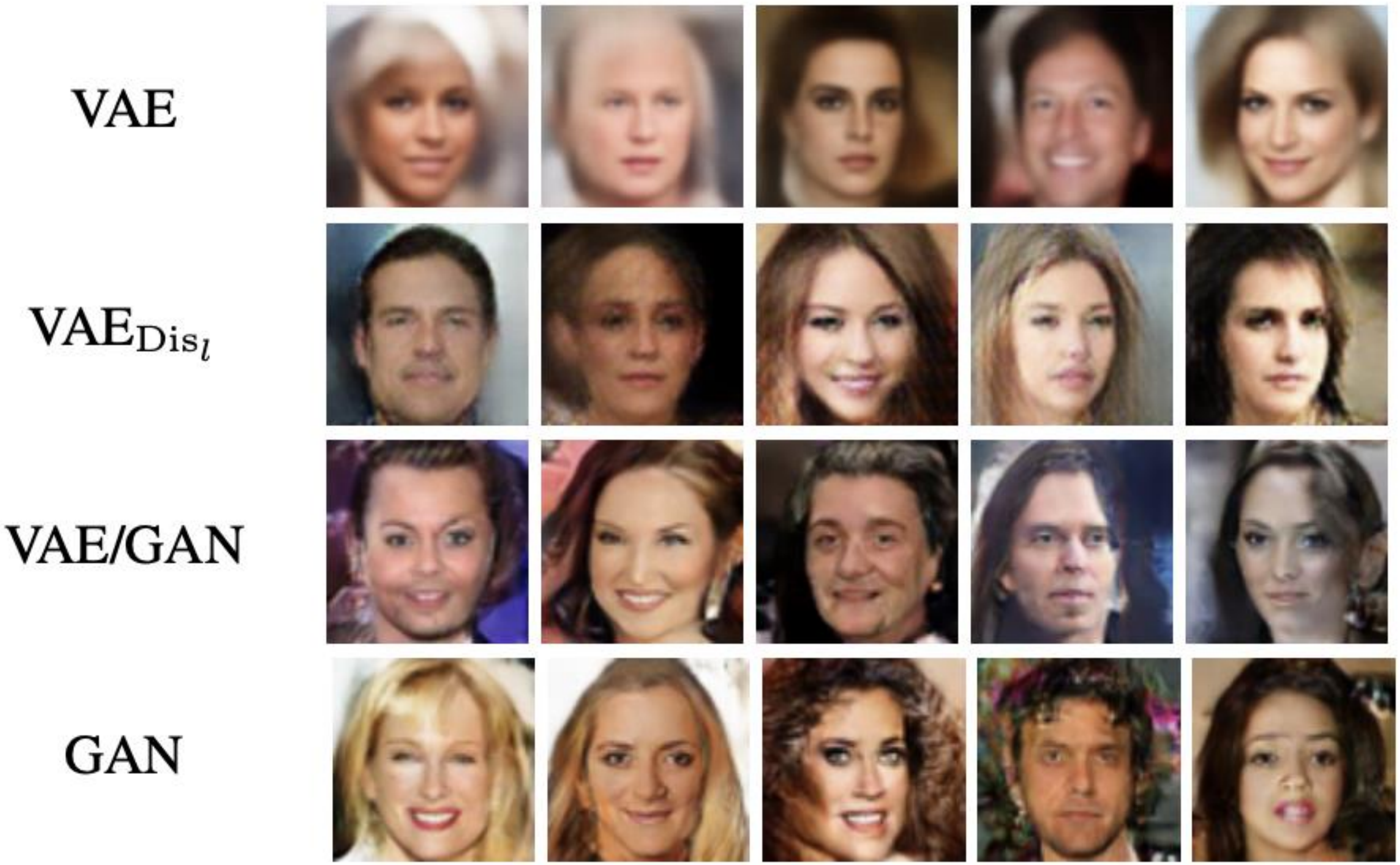
$$\mathrm{KLD}(\mathcal{N}(E^{\mu}_{\psi}(x), E^{\sigma}_{\psi}(x)) \mid \mathcal{N}(0, I))$$

VAE + GANs



Autoencoding beyond pixels using a learned similarity metric [Larsen et al. 2015]

VAE + GANs



VAE(Disl) = VAE + feature matching loss

[Larsen et al. 2015]

Variational Autoencoder (VAE)

Data $\rightarrow$

Learner

Objective

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

Hypothesis space

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

Optimizer

Variational Bayes

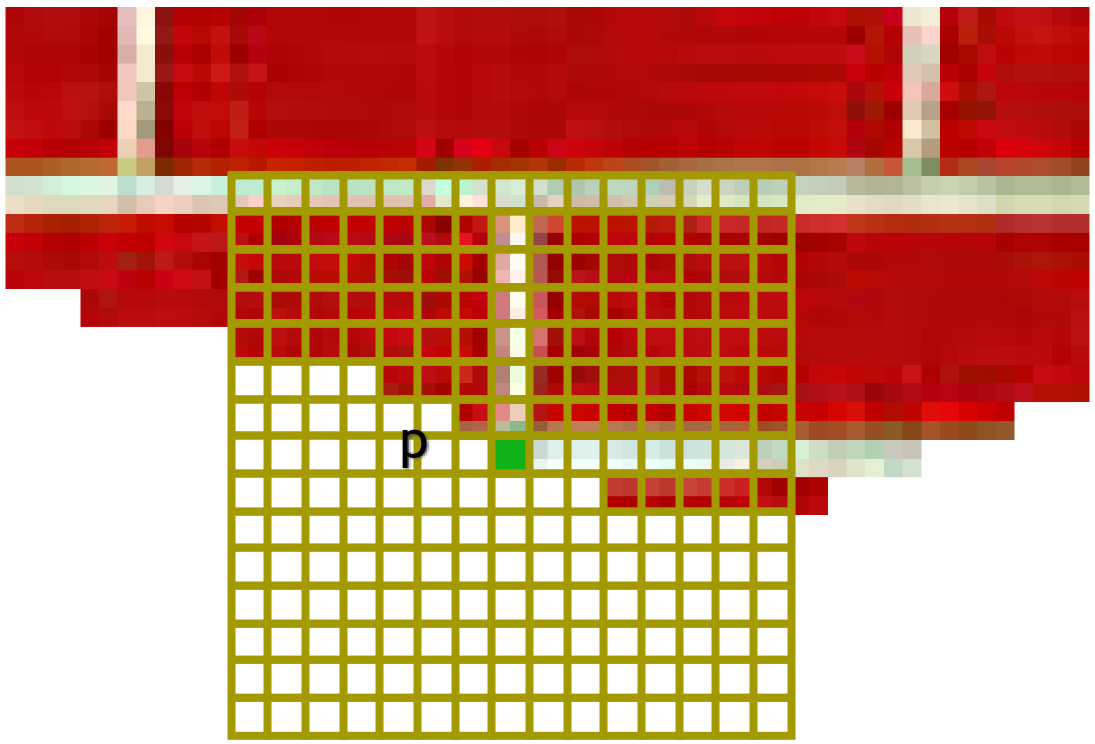
$\rightarrow$ Sampler

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

Autoregressive Model

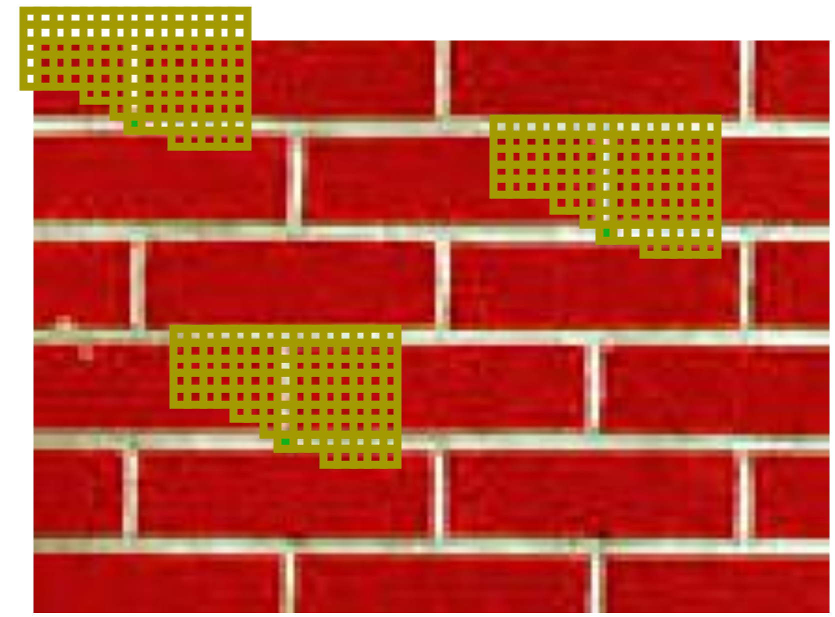
Texture synthesis by non-parametric sampling

[Efros & Leung 1999]



Synthesizing a pixel

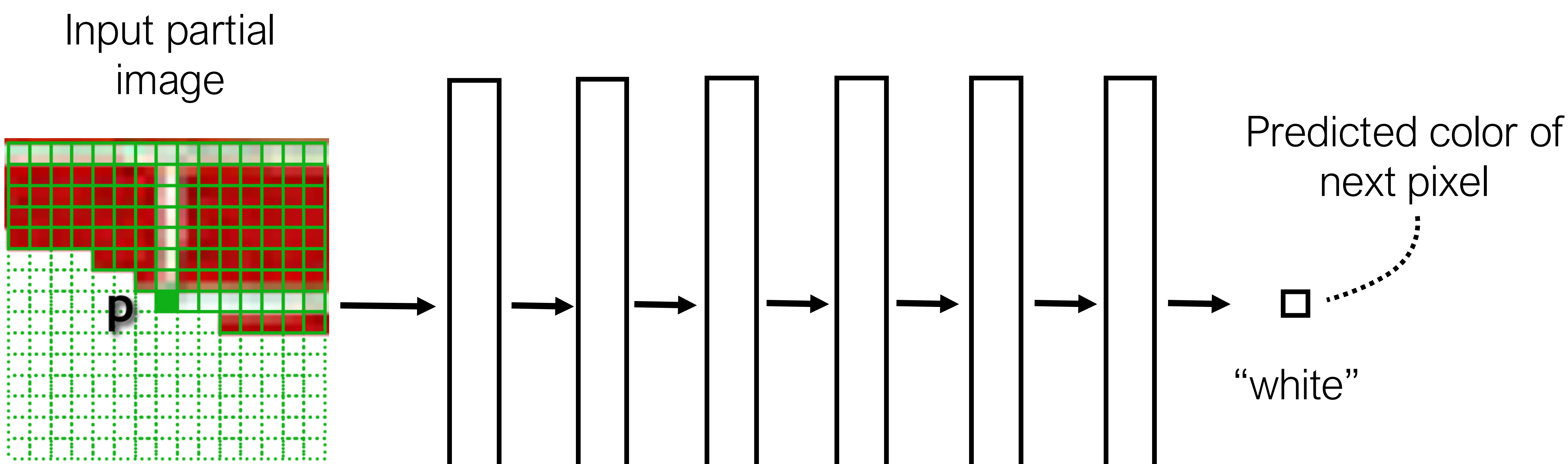
non-parametric
sampling



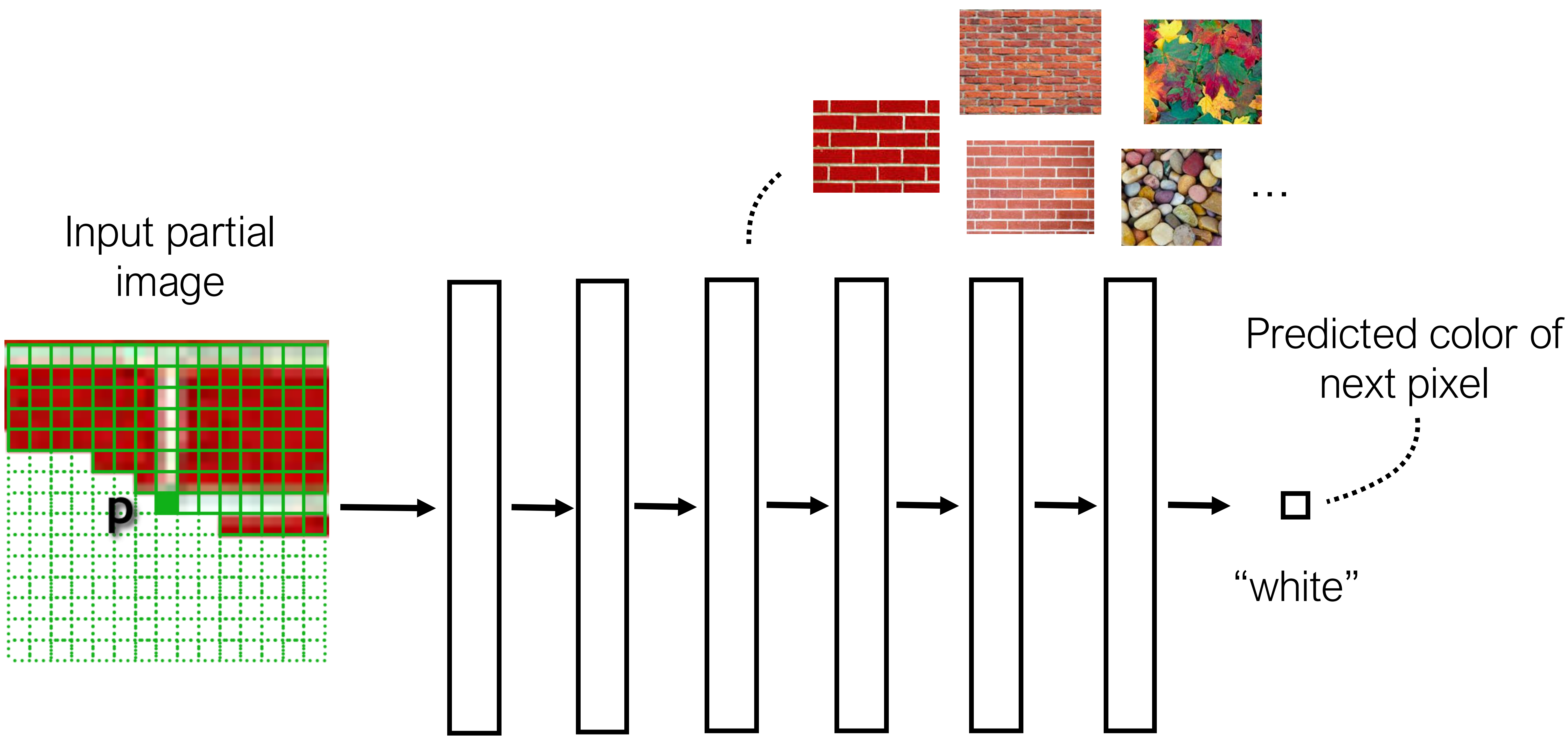
Input image

Models $P(p|N(p))$

Autoregressive image synthesis



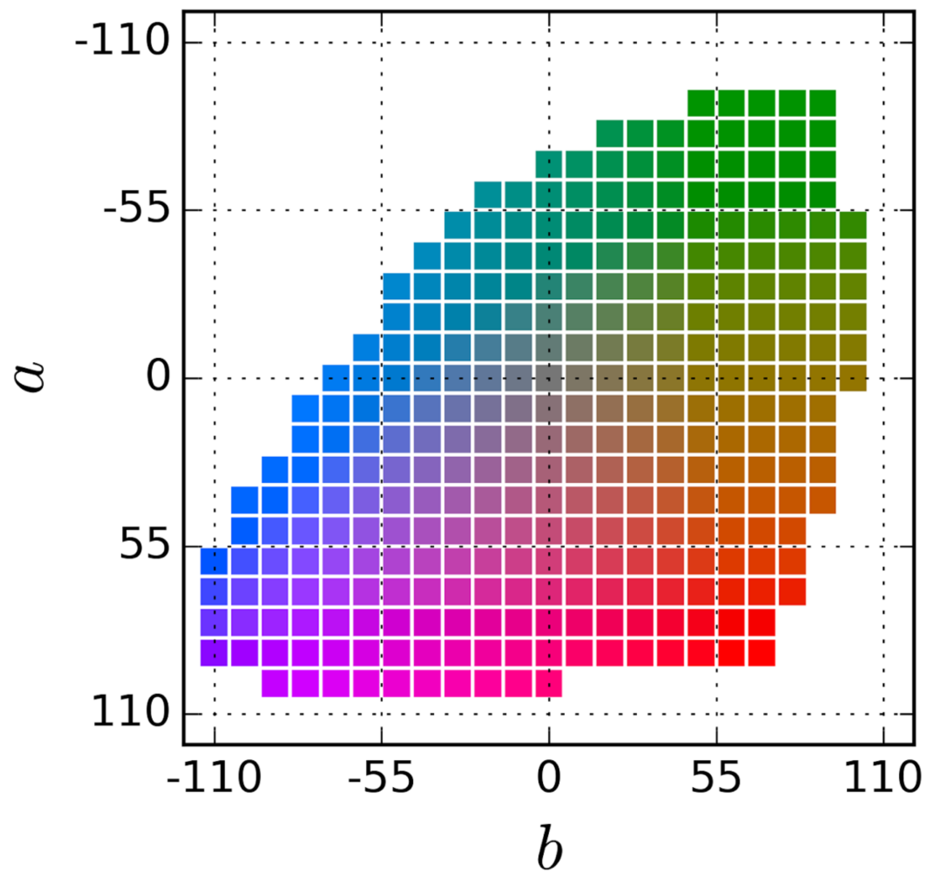
[PixelRNN, PixelCNN, van der Oord et al. 2016]



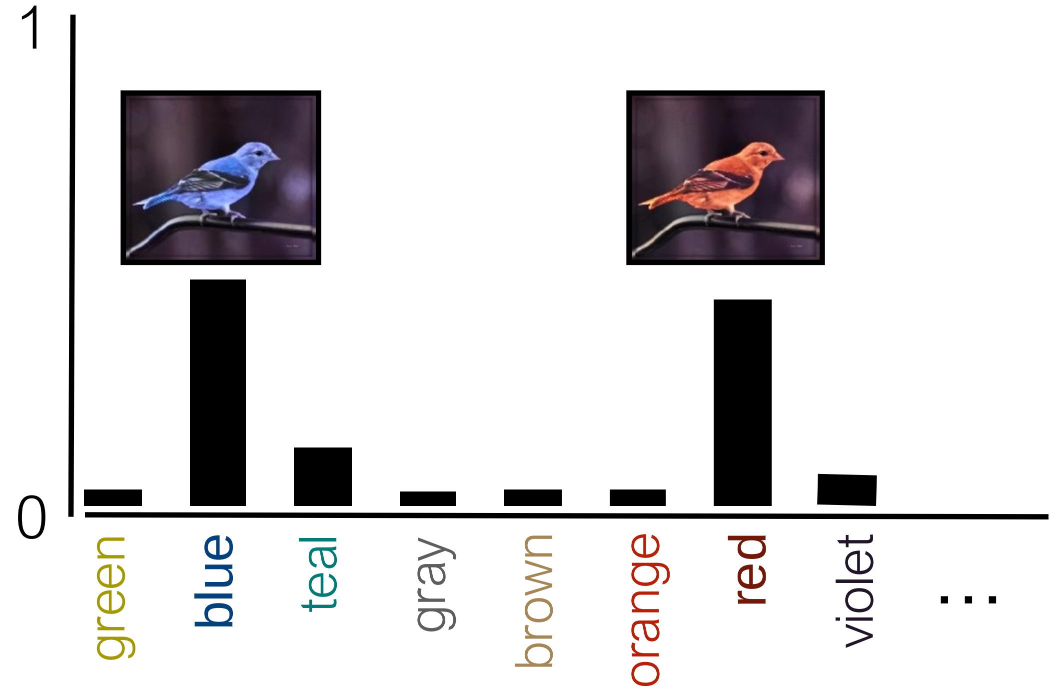
[PixelRNN, PixelCNN, van der Oord et al. 2016]

Recall: we can represent colors as discrete classes

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



Prediction for a single pixel i,j



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

And we can interpret the learner as modeling P(next pixel | previous pixels):

Softmax regression (a.k.a. multinomial logistic regression)

$$\hat{\mathbf{y}} \equiv [P_\theta(Y=1|X=\mathbf{x}), \dots, P_\theta(Y=K|X=\mathbf{x})]$$

predicted probability of each class given input **x**

One-hot vector

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

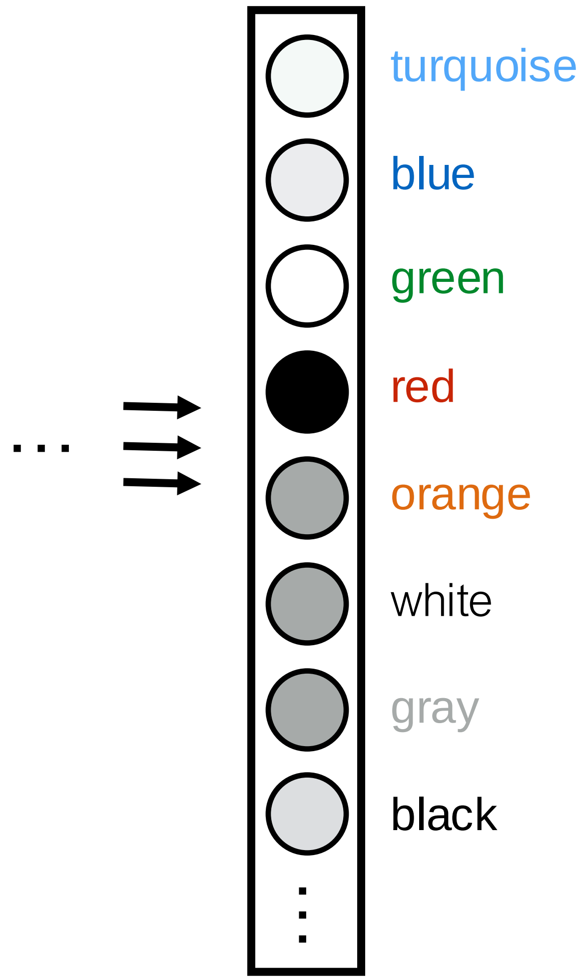
picks out the -log likelihood of the ground truth class under $\mathbf{y}$ the model prediction $\hat{\mathbf{y}}$

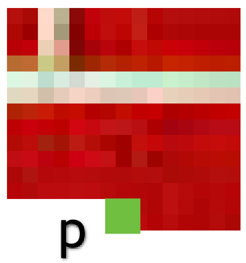
$$f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} H(\mathbf{y}_i, \hat{\mathbf{y}}_i)$$

max likelihood learner!

Cross-entropy loss

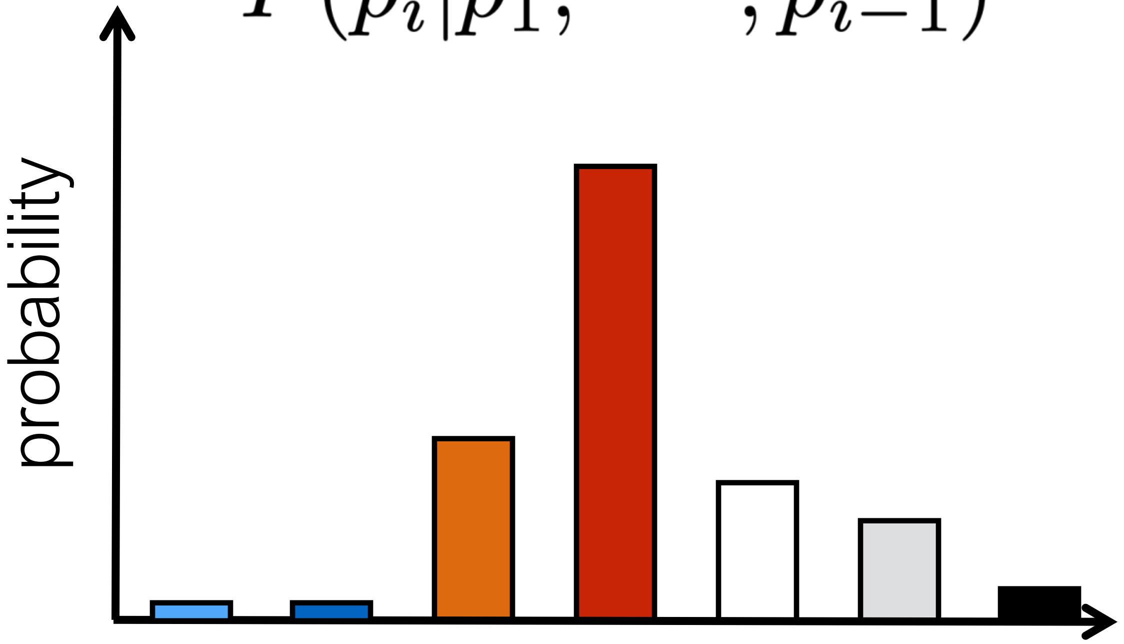
Network output

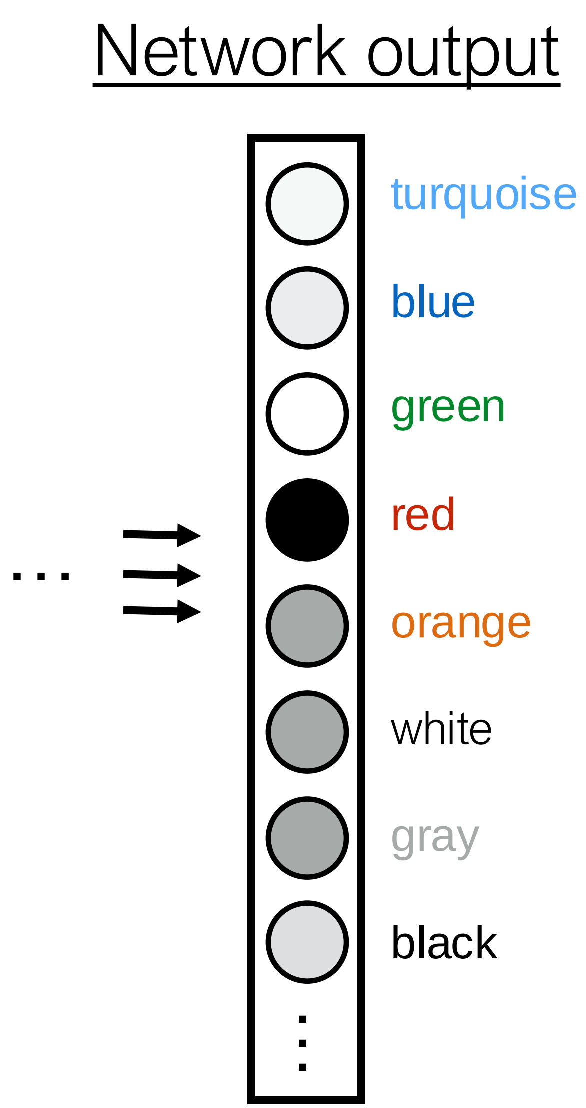


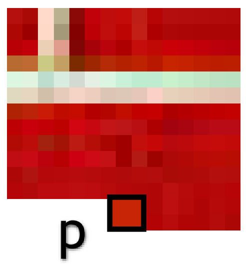


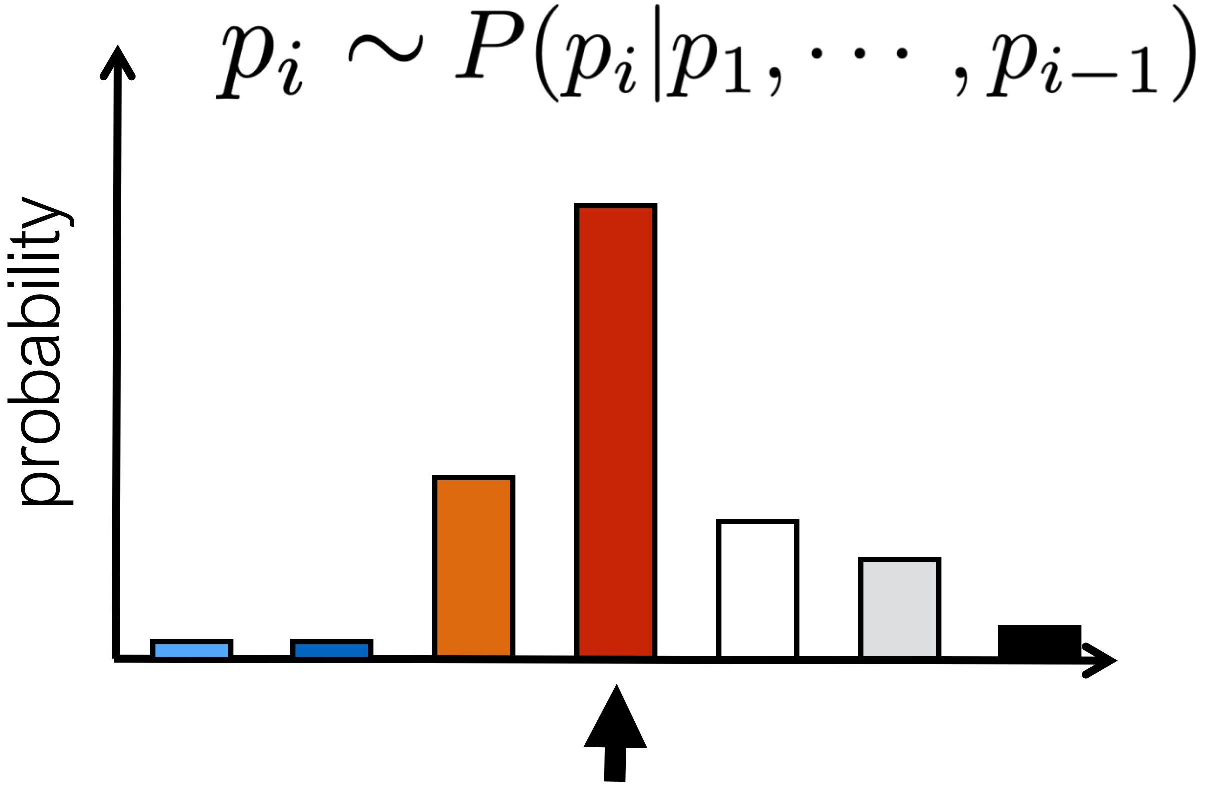
P(next pixel | previous pixels)

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

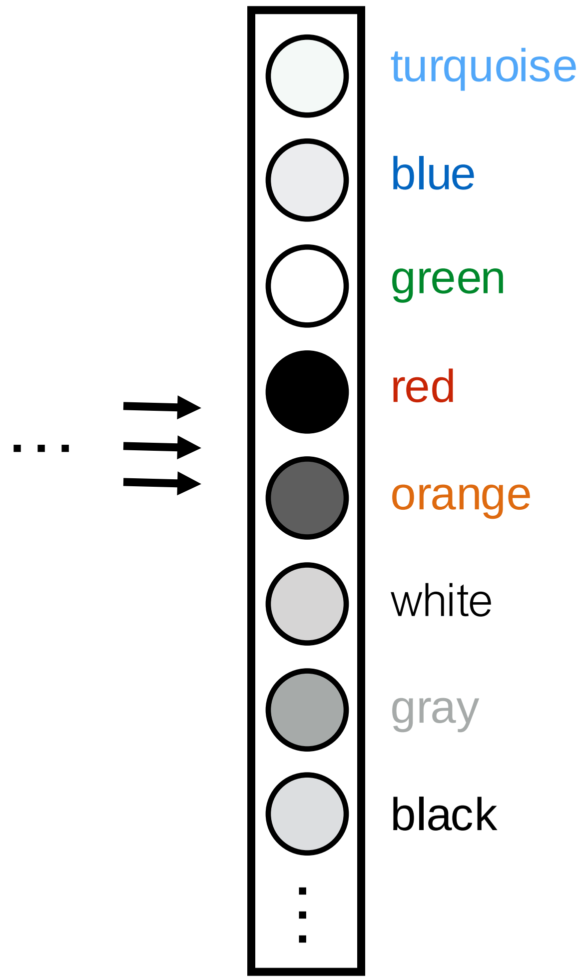


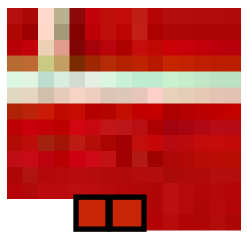

Network output
turquoise
blue
green
red
orange
white
gray
black


p

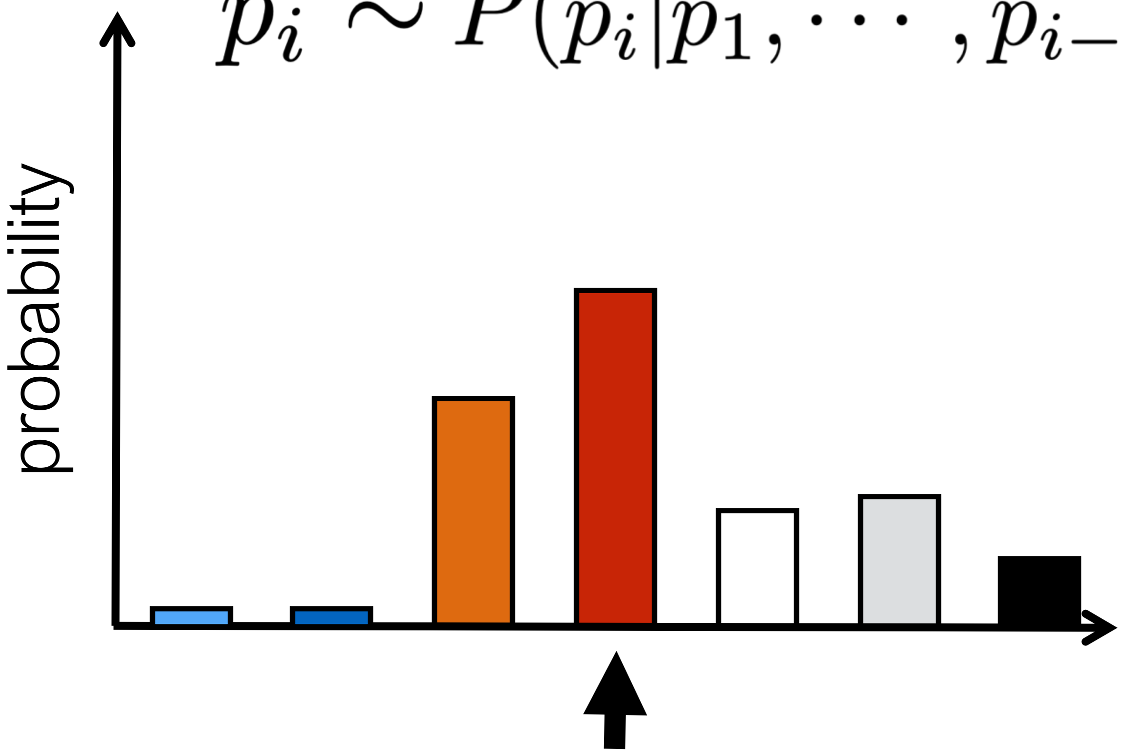

$p_i \sim P(p_i|p_1, \cdots, p_{i-1})$
probability

Network output

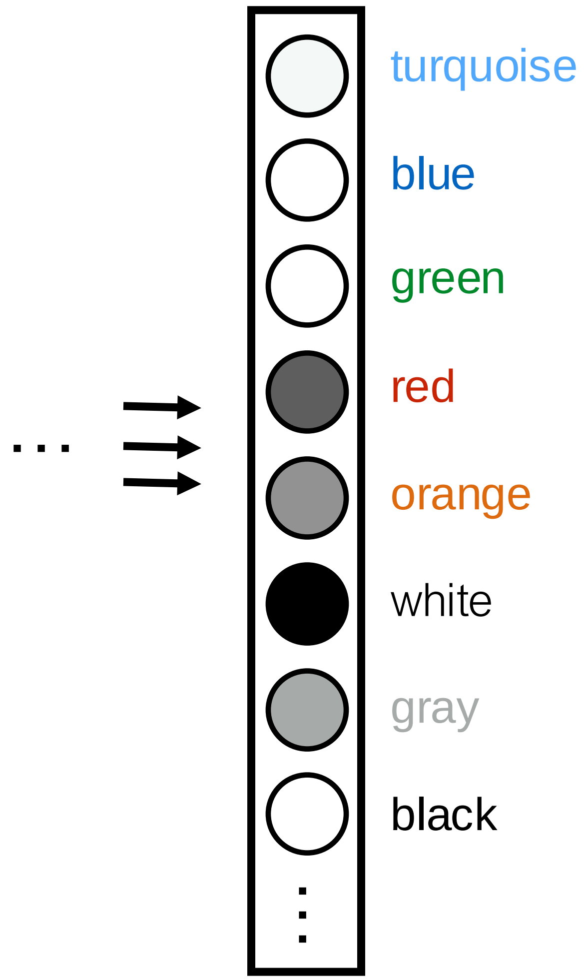


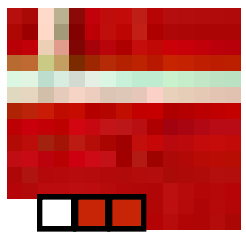


$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$

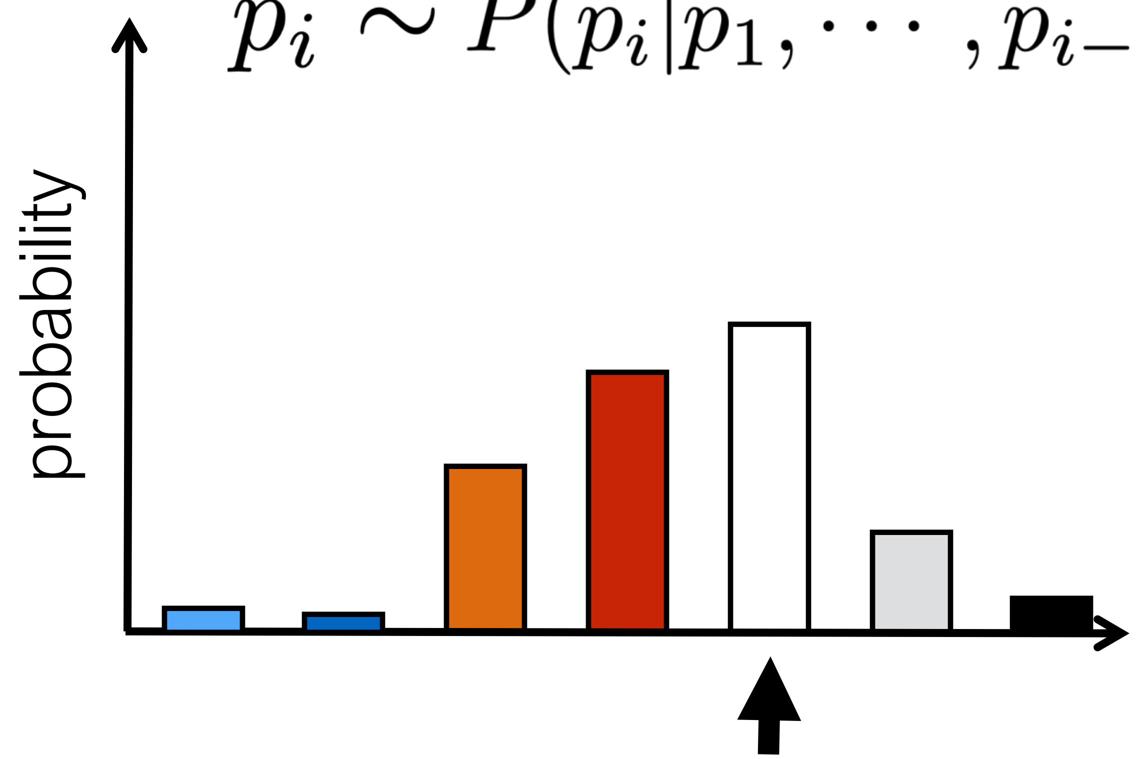


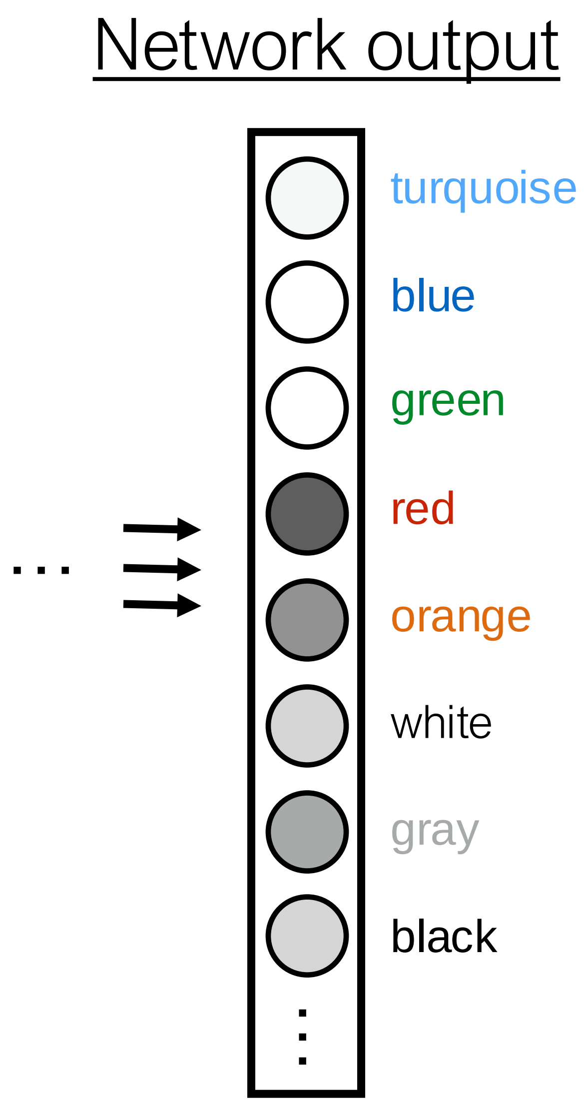
Network output

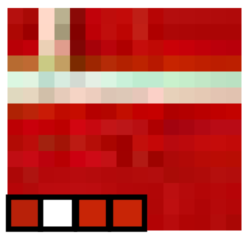




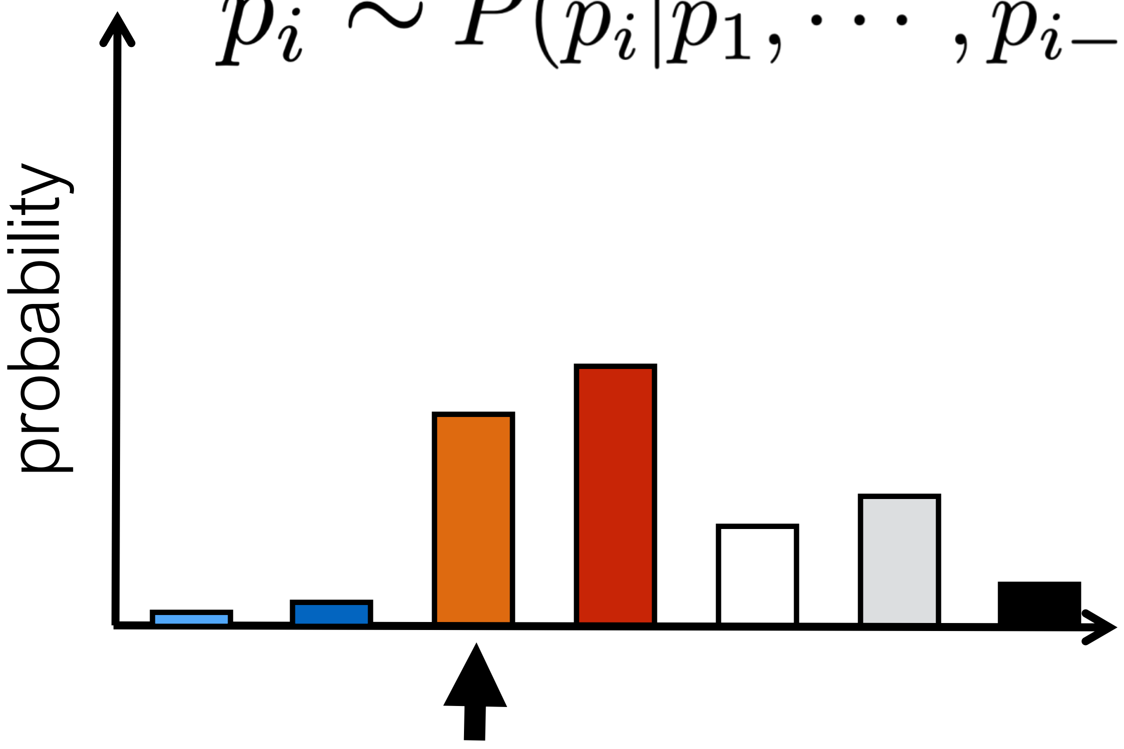
$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$

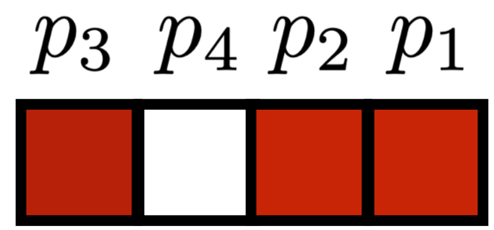






$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$





$$p_1 \sim P(p_1)$$

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

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

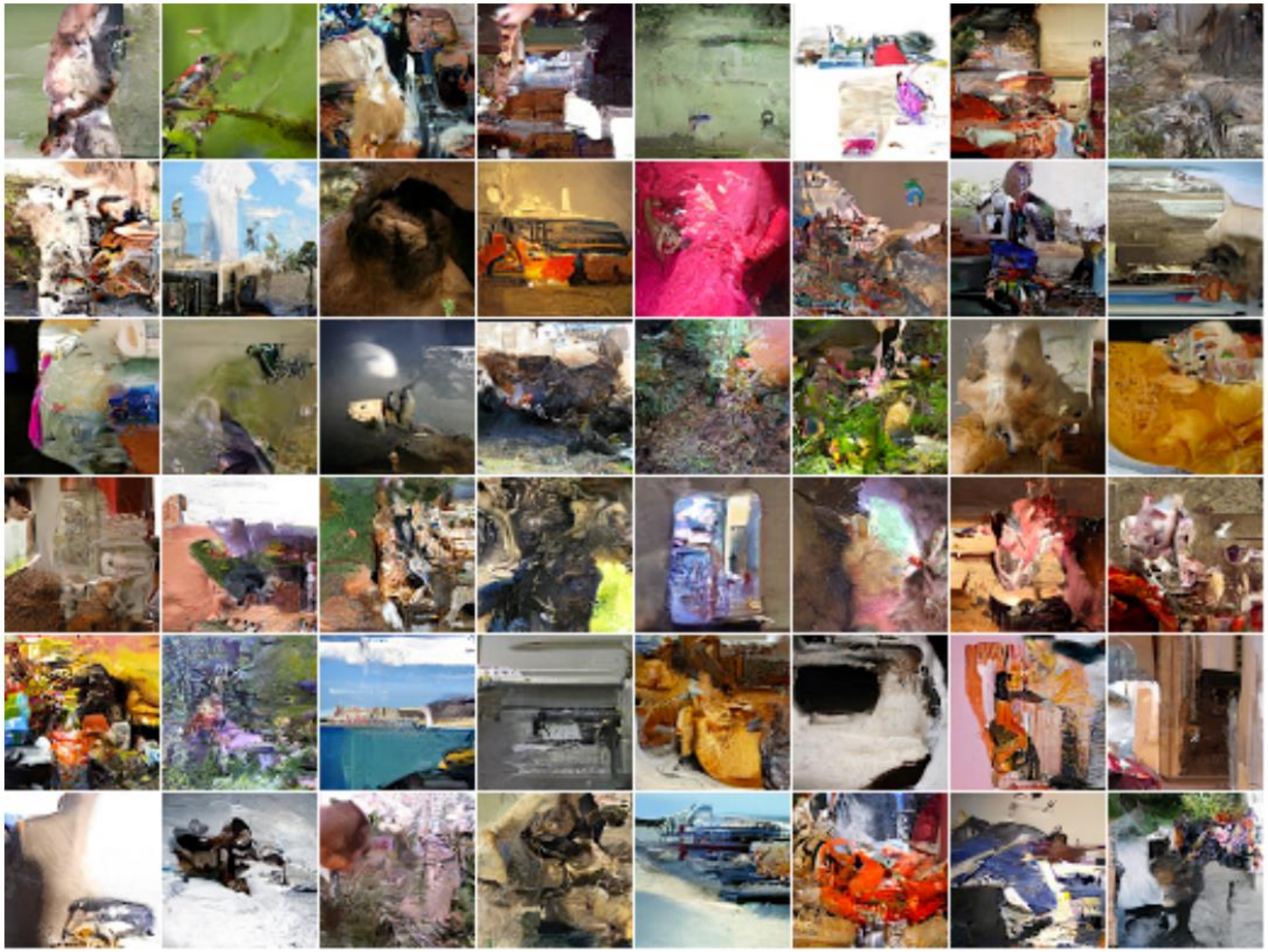
$$p_4 \sim P(p_4|p_1, p_2, p_3)$$

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

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

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

Samples from PixelRNN

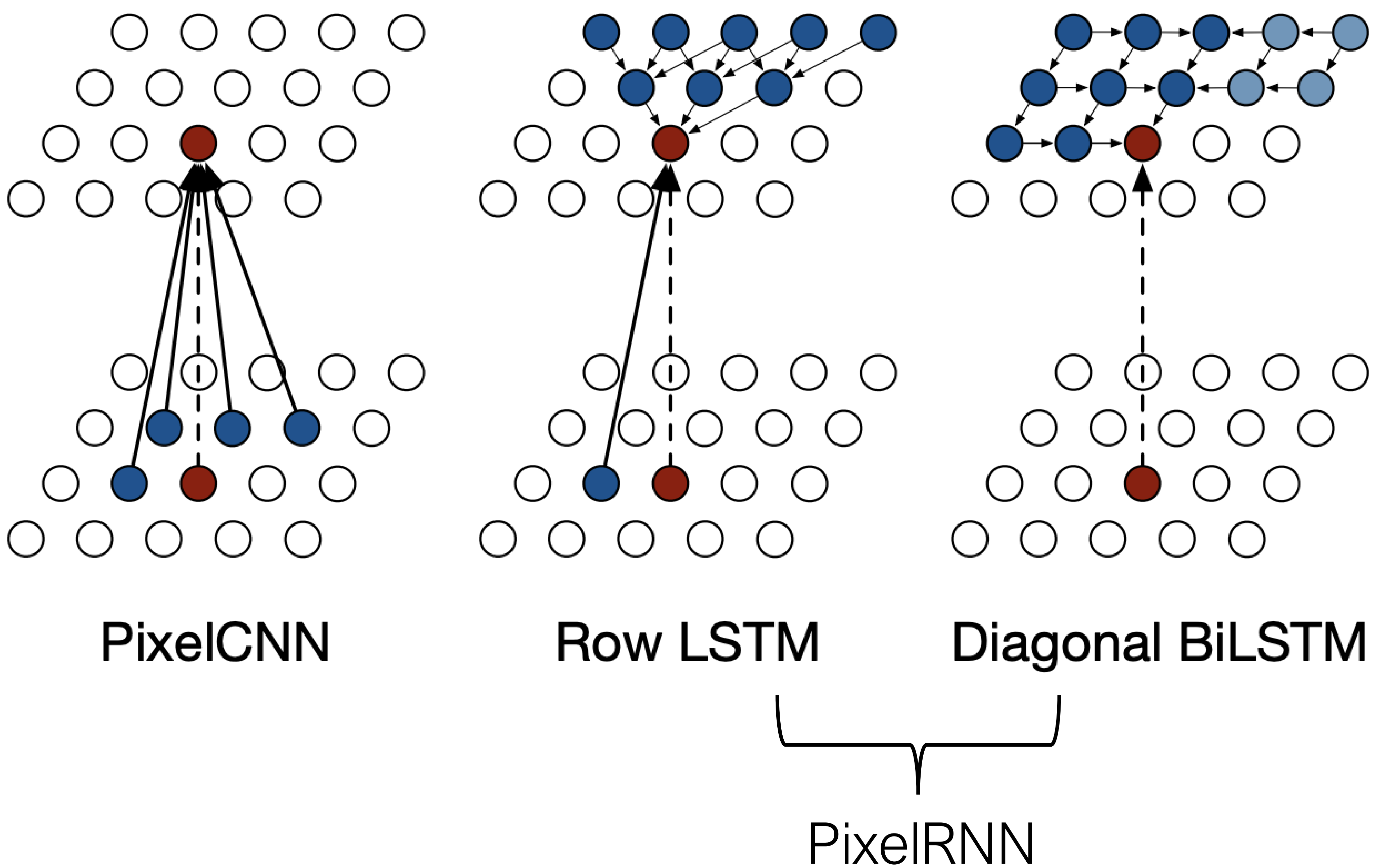


[PixelRNN, van der Oord et al. 2016]

Image completions (conditional samples) from PixelRNN

[PixelRNN, van der Oord et al. 2016]

PixelCNN vs. PixelRNN



Checkout PixelCNN++ [Salimans et al.,2017] (+ coarse-to-fine, ResNet, whole pixels, etc.)

How to improve PixelCNN?

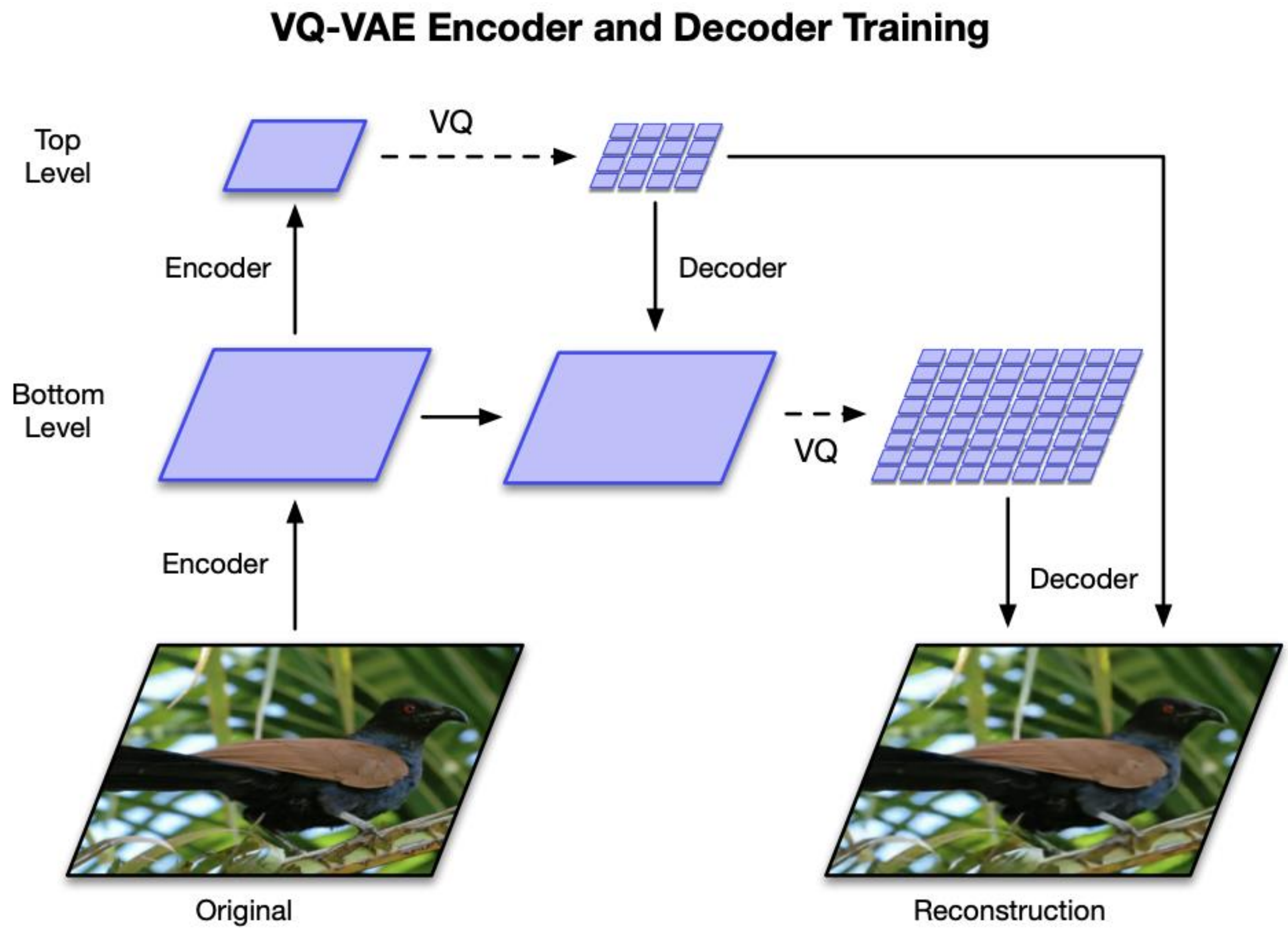
- What are the limitations of PixelCNN/RNNs?
  - Slow sampling time.
  - May accumulate errors over multiple steps. (might not be a big issue for image completion)
- How can we make it faster?
- How can we further improve results?

VQ-VAE-2: VAE+PixelCNN

VQ (Vector quantization) maps continuous vectors into discrete codes
Common methods: clustering (e.g., k-means)

Generating Diverse High-Fidelity Images with VQ-VAE-2 [Razavi et al., 2019]

VQ-VAE-2: VAE+PixelCNN

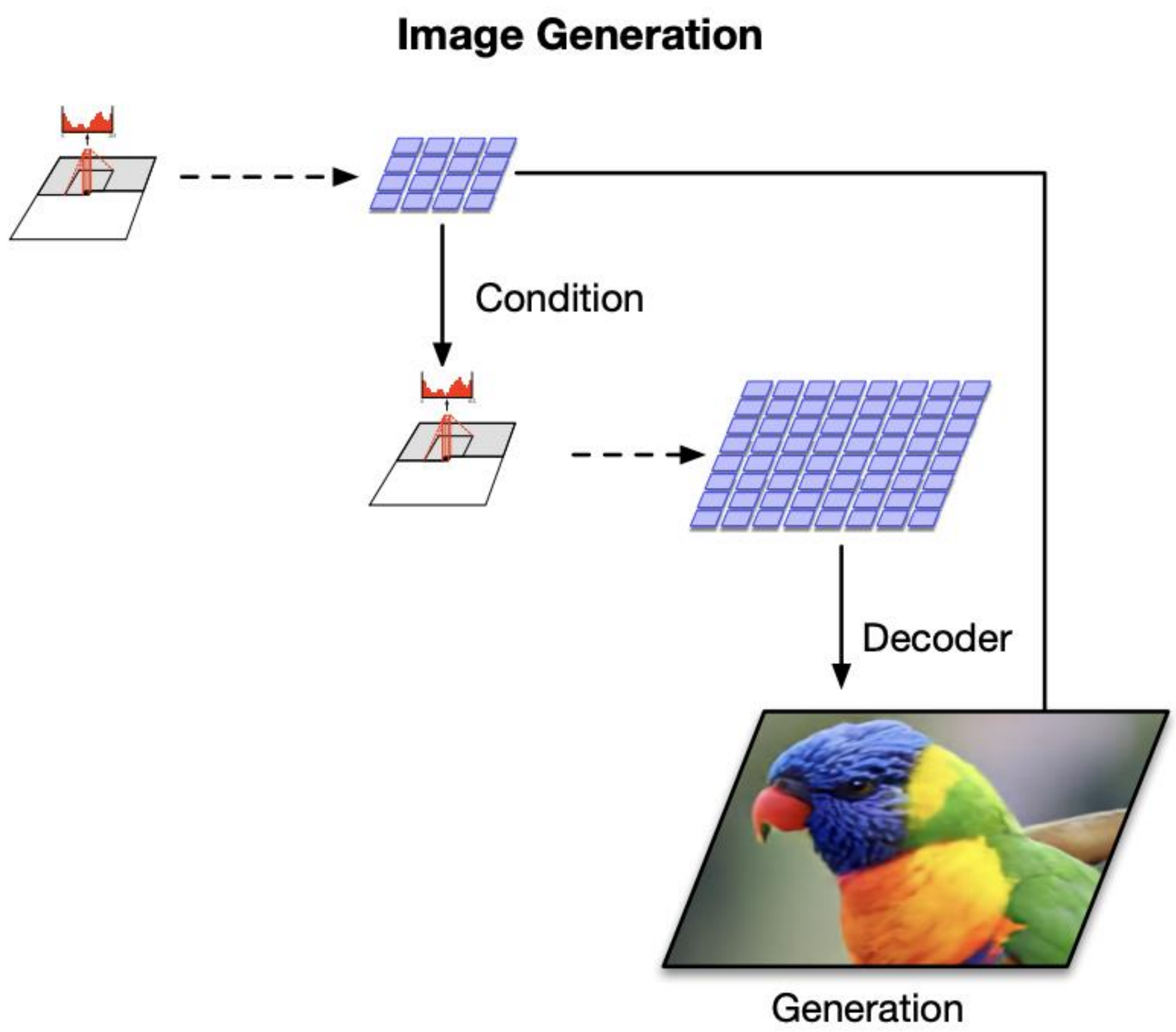


VAE+VQ: learn a more compact codebook for PixelCNN (instead of pixels)

PixelCNN: use a more expressive bottleneck for VAE (instead of Gaussian)

[Razavi et al., 2019]

VQ-VAE-2: VAE+PixelCNN



VAE+VQ: learn a more compact codebook for PixelCNN (instead of pixel colors)
PixelCNN: use a more expressive bottleneck for VAE (instead of Gaussian prior)

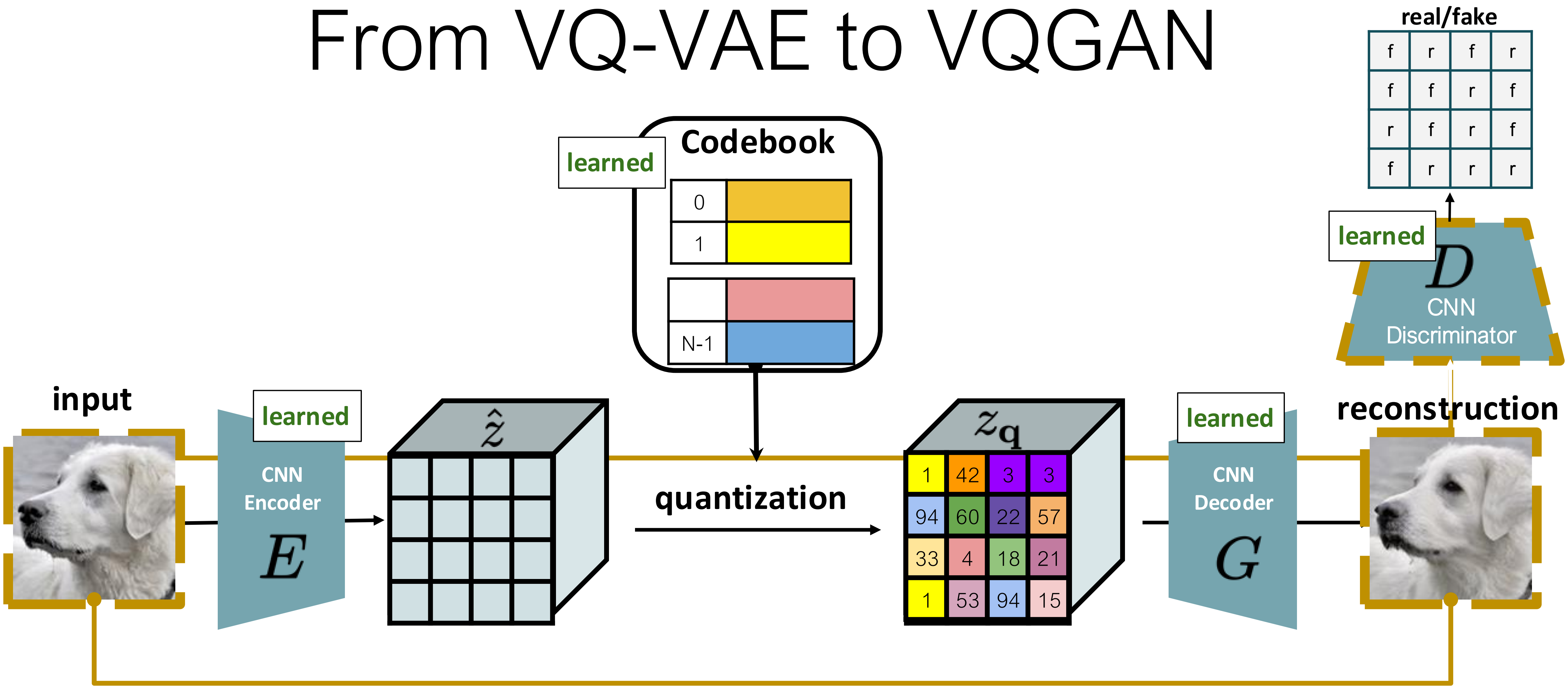
[Razavi et al., 2019]

How to Improve further?

- Better architectures
- Better loss functions for encoder-decoder

[Razavi et al., 2019]

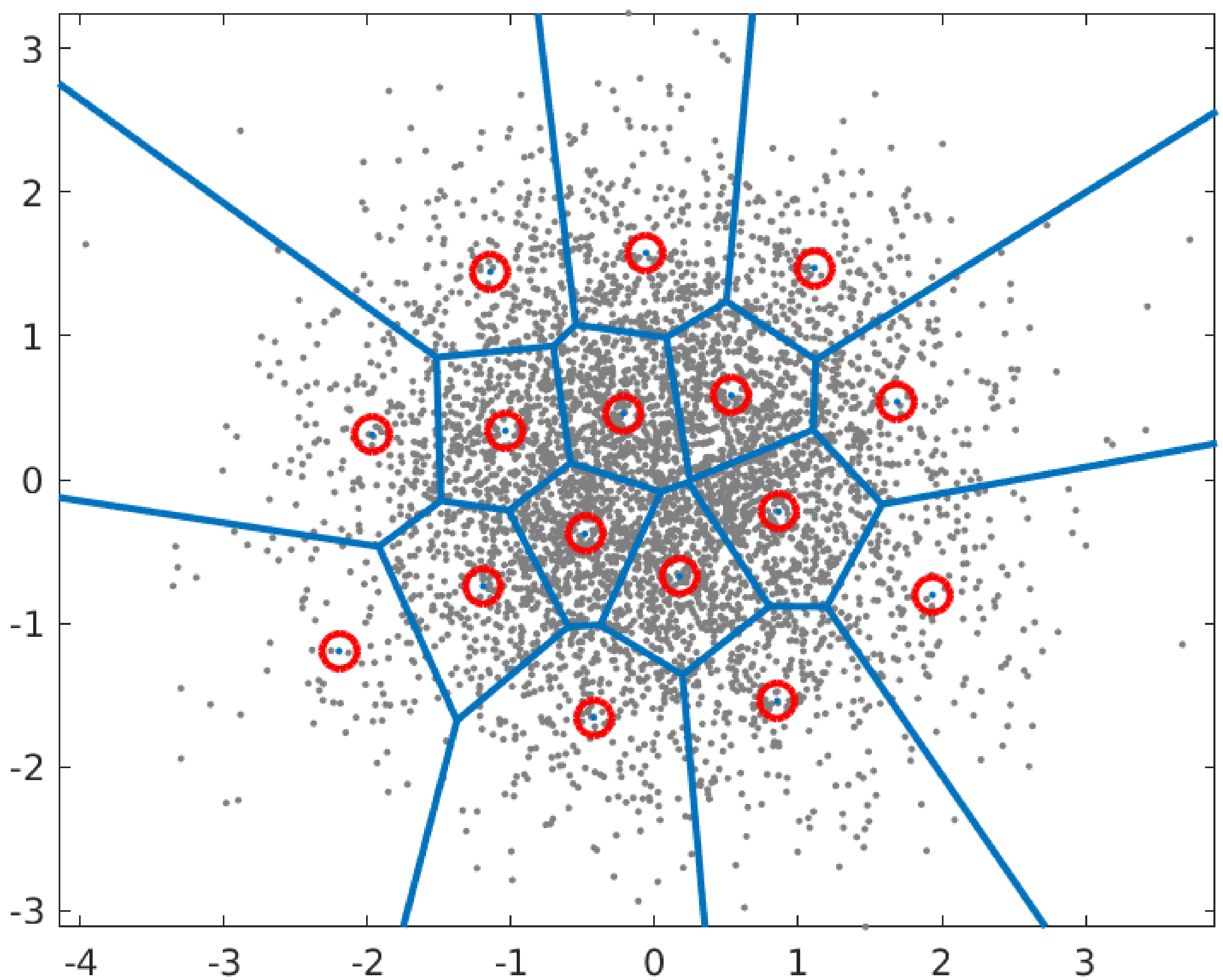
From VQ-VAE to VQGAN



1) replace L2/L1 rec. loss with Perceptual loss (includes pixel-level)
2) add Discriminator to favor realism over per-pixel reconstruction
3) use transformer rather than CNN

Slide credit: Robin Rombach

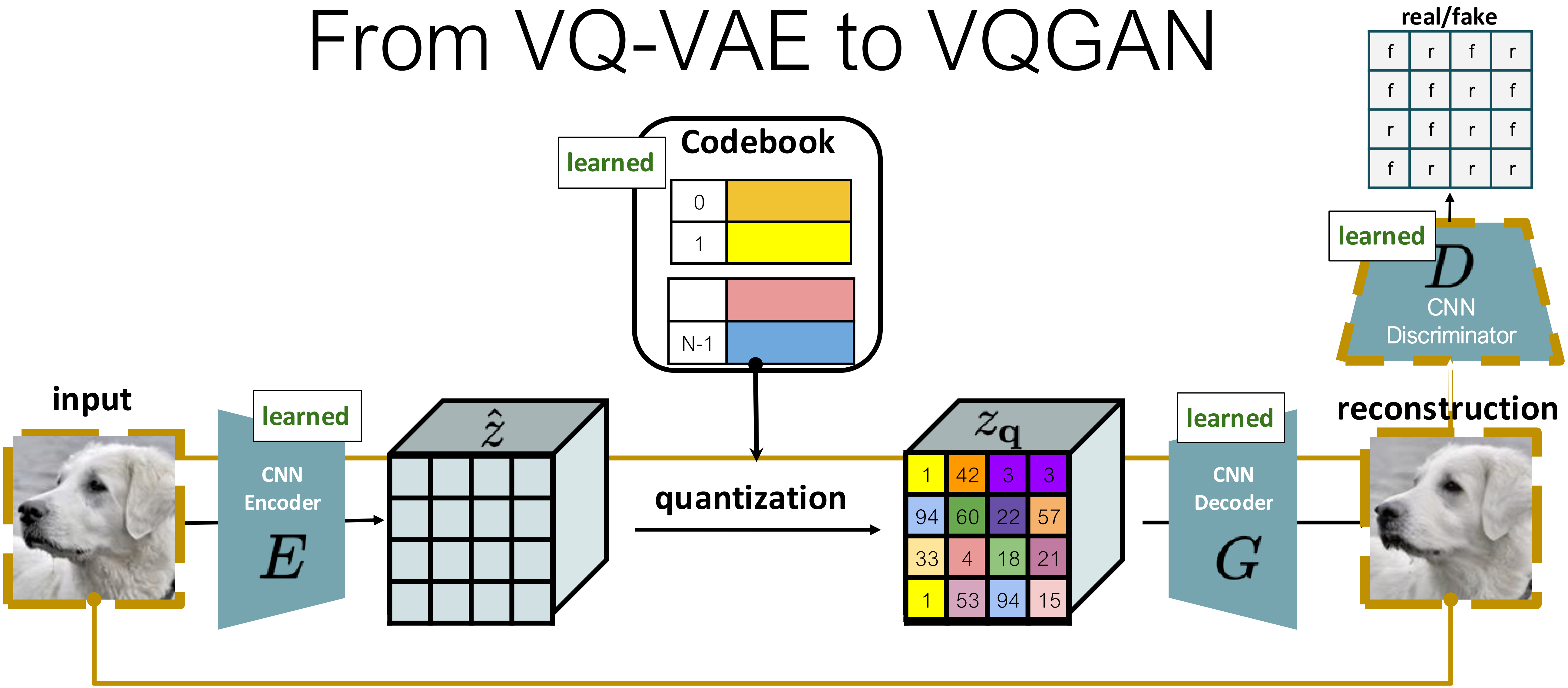
Vector Quantization (VQ)



K-means, EM (GMM), end-to-end learning

https://wiki.aalto.fi/pages/viewpage.action?pageId=149883153

From VQ-VAE to VQGAN



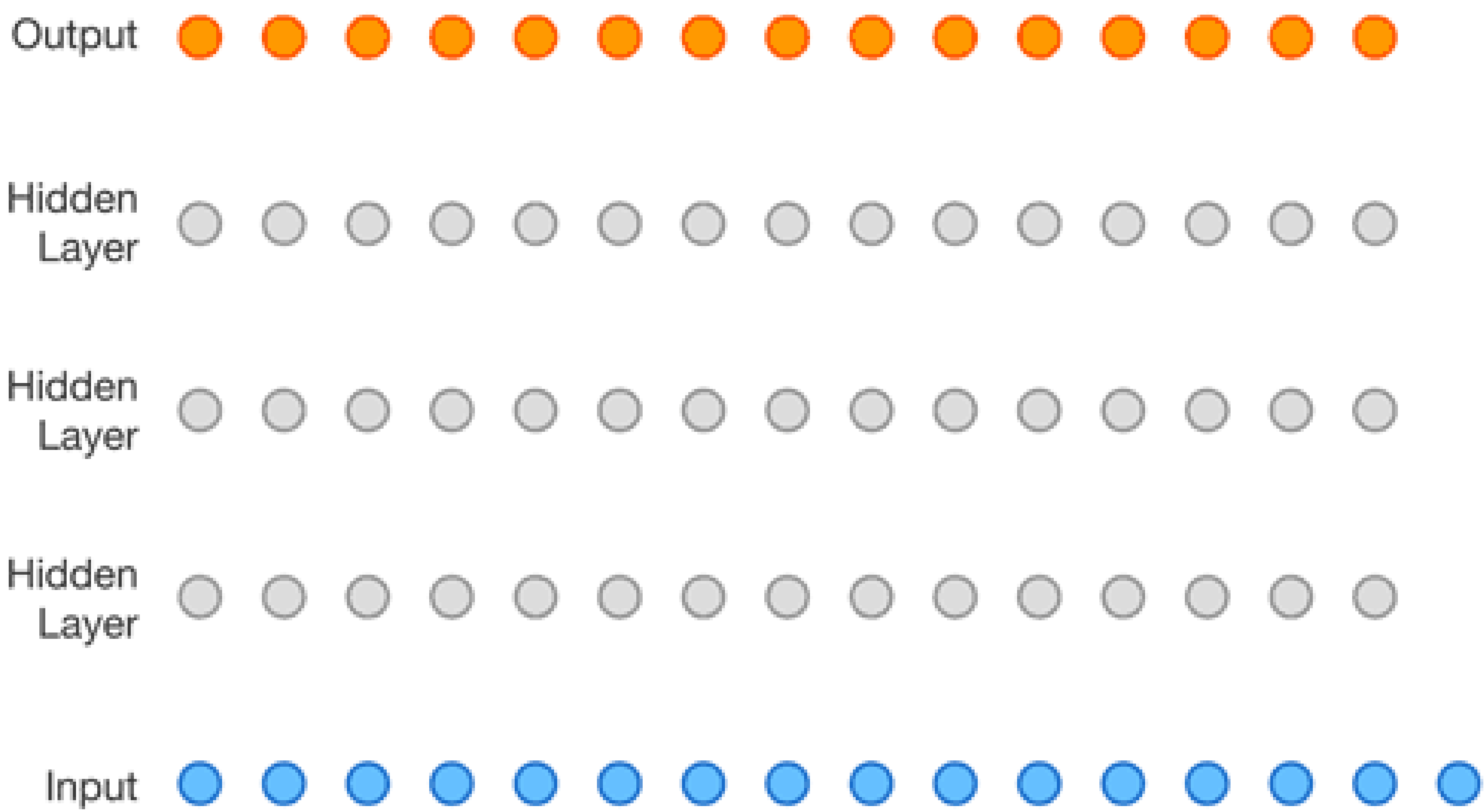
$$\mathcal{L}_{\text{total}} = \mathcal{L}_{rec} + \mathcal{L}_{VQ} + \lambda \mathcal{L}_{GAN} \quad \text{where} \quad \lambda = \frac{\nabla_{G_L}[\mathcal{L}_{rec}]}{\nabla_{G_L}[\mathcal{L}_{GAN}] + \delta}$$

Slide credit: Robin Rombach

Slide credit: Robin Rombach

Slide credit: Robin Rombach

WaveNet



[**Wavenet**, https://deepmind.com/blog/wavenet-generative-model-raw-audio/]

Auto-regressive models works extremely well for audio/music data.

Autoregressive Model

Data $\rightarrow$

Learner

Objective

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

Hypothesis space

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

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

Autoregressive probability model

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

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

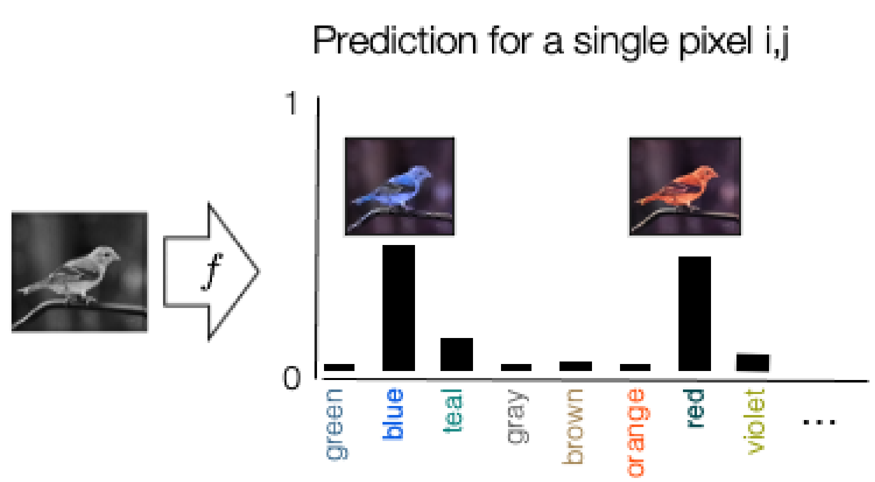
General product rule

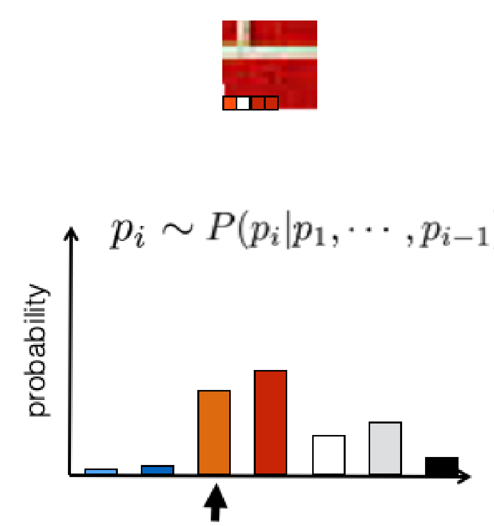
The sampling procedure we defined above takes exact samples from the learned probability distribution (pmf).

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

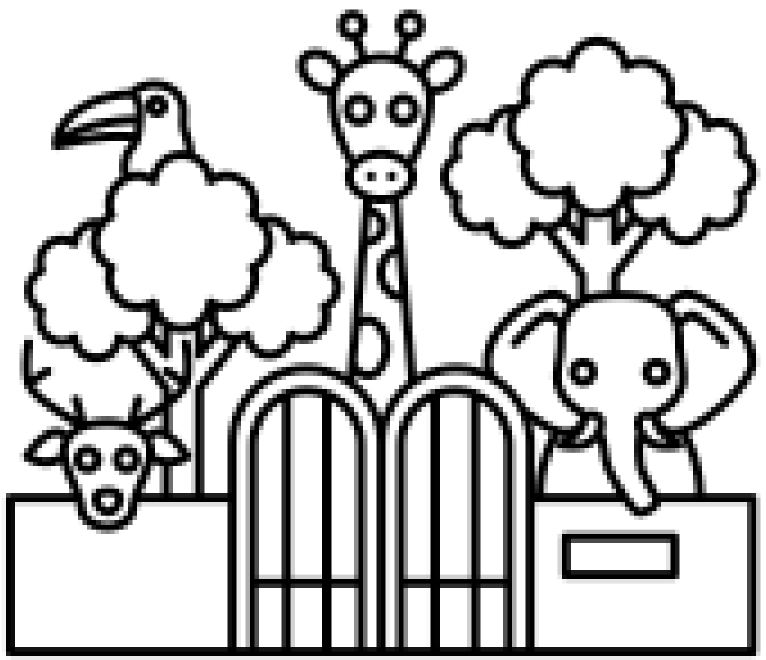
Per-pixel classification vs. Autoregressive

- Image colorization: per-pixel classification loss
- PixelCNN, VQ-VAE2: autoregressive model
- Key idea: only produce discrete representations (e.g., VQ codebook, 313 ab color bins)
- Differences: Independent vs. sequential prediction





Thank You!



16-726 Learning-based Image Synthesis

https://learning-image-synthesis.github.io/