HW2

sources/destinations

cloning

seamless cloning

Student Presentation (Generative Models)

paper titles	venue	speakers
A Style-Based Generator Architecture for Generative Adversarial Networks (StyleGAN)	CVPR 2019	
Large Scale GAN Training for High Fidelity Natural Image Synthesis (BigGAN)	ICLR 2019	
Generating Diverse High-Fidelity Images with VQ-VAE-2 (VQ-VAE-2)	NeurIPS 2019	
Conditional Image Generation with PixelCNN Decoders (PixelCNN)	NeurIPS 2016	
Glow: Generative Flow with Invertible 1x1 Convolutions (Glow)	NeurIPS 2018	
Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)	CVPR 2020	
Denoising Diffusion Probabilistic Models (DDPM)	NeurIPS 2020	
Denoising Diffusion Implicit Models (DDIM)	ICLR 2021	
Large scale adversarial representation learning (BigBiGAN)	ICLR 2019	
Alias-Free Generative Adversarial Networks (StyleGAN3)	NeurIPS 2021	
SinGAN: Learning a Generative Model from a Single Natural Image (SinGAN)	ICCV 2019	
Score-Based Generative Modeling through Stochastic Differential Equations (SDE)	ICLR 2021	

What has driven GAN progress?

- Loss functions:

  cross-entropy, least square, Wasserstein loss, gradient penalty, Hinge loss, …
- Network architectures (G/D)

  Conv layers, Transposed Conv layers, modulation layers (AdaIN, spectral norm)
  mapping networks, …
- Training methods

  1. coarse-to-fine progressive training
  2. using pre-trained classifiers (multiple classifiers, random projection)
- Data

  data alignment, differentiable augmentation
- GPUs

  bigger GPUs = bigger batch size (stable training) + higher resolution

Generative Model Zoo

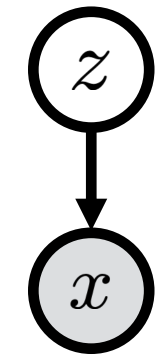
Jun-Yan Zhu

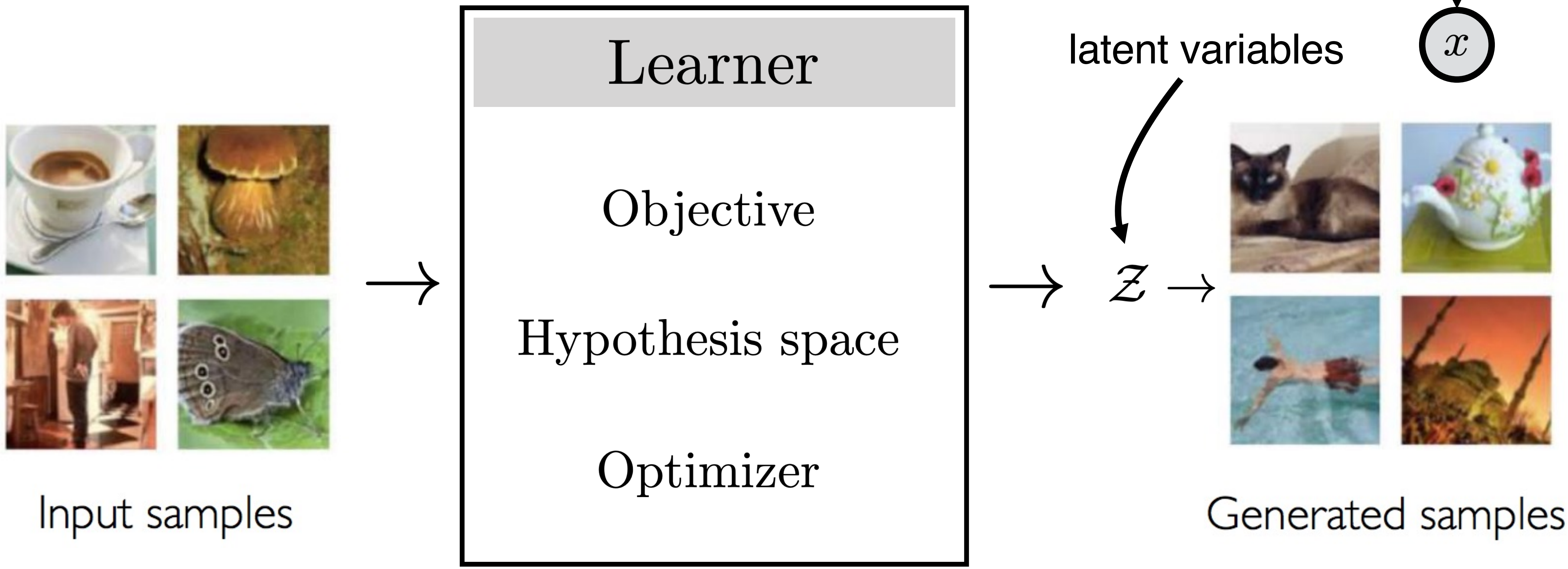
16-726 Learning-based Image Synthesis, Spring 2022

many slides from Phillip Isola, Richard Zhang, Alyosha Efros

© ATOM

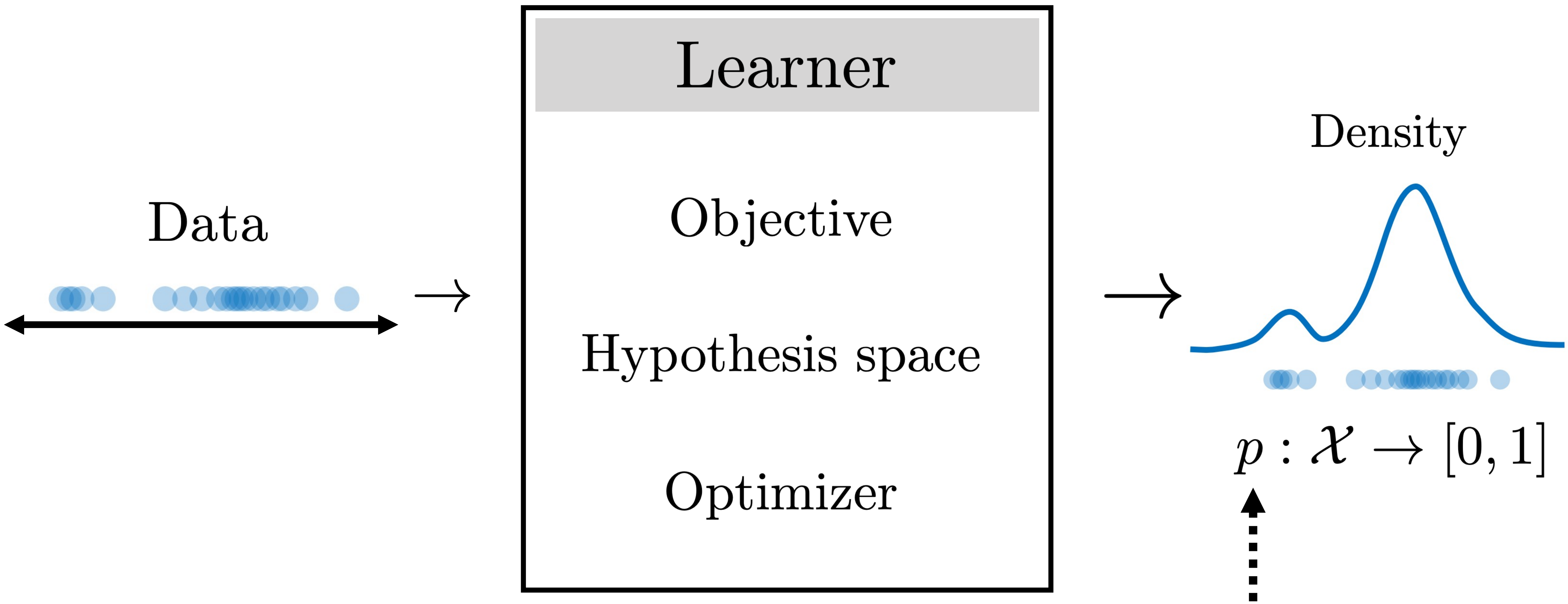
Learning a generative model





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

Learning a density model



Integral of probability density function needs to be 1 → Normalized distribution

(some models output unormalized ***energy functions***)

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

Useful for abnormality/outlier detection (detect unlikely events)

Case study #1: Fitting a Gaussian to data

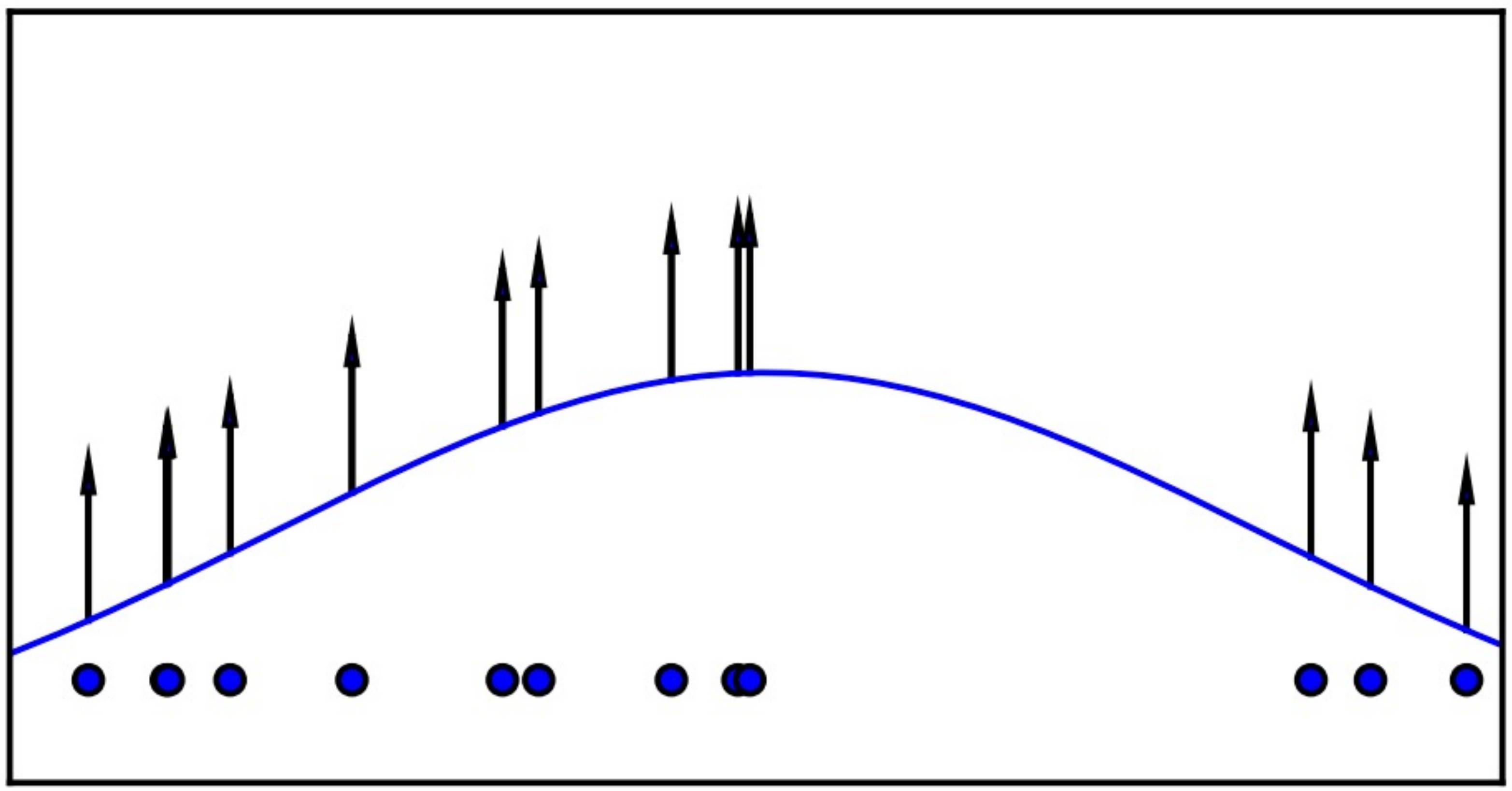


fig from [Goodfellow, 2016]

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:

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

Maximum log likelihood=minimize KLD

KLD (Kullback–Leibler divergence): $\mathcal{KL}(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx$

JSD (Jensen–Shannon divergence): $\mathcal{JSD}(p \parallel q) = \frac{1}{2}\mathcal{KL}(p \parallel \frac{p+q}{2}) + \frac{1}{2}\mathcal{KL}(q \parallel \frac{p+q}{2})$

$$\mathbb{E}_{x \sim p_{\text{data}}(x)}[\log p_\theta(x)] = \int_x p_{\text{data}}(x) \log p_\theta(x) dx$$

$$\mathcal{KL}(p_{\text{data}}(x)||p_\theta(x)) = \int_x p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{p_\theta(x)} dx$$

$$= \int_x p_{\text{data}}(x) \log p_{\text{data}}(x) dx - \int_x p_{\text{data}}(x) \log p_\theta(x) dx$$

Constant (independent of θ)

Maximize log likelihood=minimize KLD

Case study #2: Generative Adversarial Network

Data →

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

→

Critic

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

Sampler

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

$p_g = p_{data}$ is the unique global minimizer of the GAN objective.

Proof

Optimal discriminator given fixed G

$$C(G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$
$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right]$$

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \left\| \frac{p_{\text{data}} + p_g}{2}\right.\right) + KL\left(p_g \left\| \frac{p_{\text{data}} + p_g}{2}\right.\right)$$

$$C(G) = -\log(4) + 2 \cdot \underbrace{JSD\left(p_{\text{data}} \| p_g\right)}_{\geq 0, \quad 0 \iff p_g = p_{data}} \quad \square$$

KLD (Kullback–Leibler divergence): $\mathcal{KL}(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx$

JSD (Jensen–Shannon divergence): $\mathcal{JSD}(p \parallel q) = \frac{1}{2}\mathcal{KL}(p \parallel \frac{p+q}{2}) + \frac{1}{2}\mathcal{KL}(q \parallel \frac{p+q}{2})$

Case study #3: learning a deep generative model

Data $\rightarrow$

Learner

Objective
max likelihood
/distribution matching

Hypothesis space
Deep net

Optimizer
SGD

$\rightarrow$

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

Case study #3: learning a deep generative model

Data $\rightarrow$

Learner

Objective
max likelihood
/distribution matching

Hypothesis space
Deep net

Optimizer
SGD

$\longrightarrow$

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

Sampler
$G : \mathcal{Z} \rightarrow \mathcal{X}$
$z \sim p(z)$
$x = G(z)$

Models that provide a sampler but no density are called **implicit generative models**

Case study #3: learning a deep generative model

Data $\rightarrow$

Learner

Objective

max likelihood
/distribution matching

Hypothesis space

Deep net

Optimizer

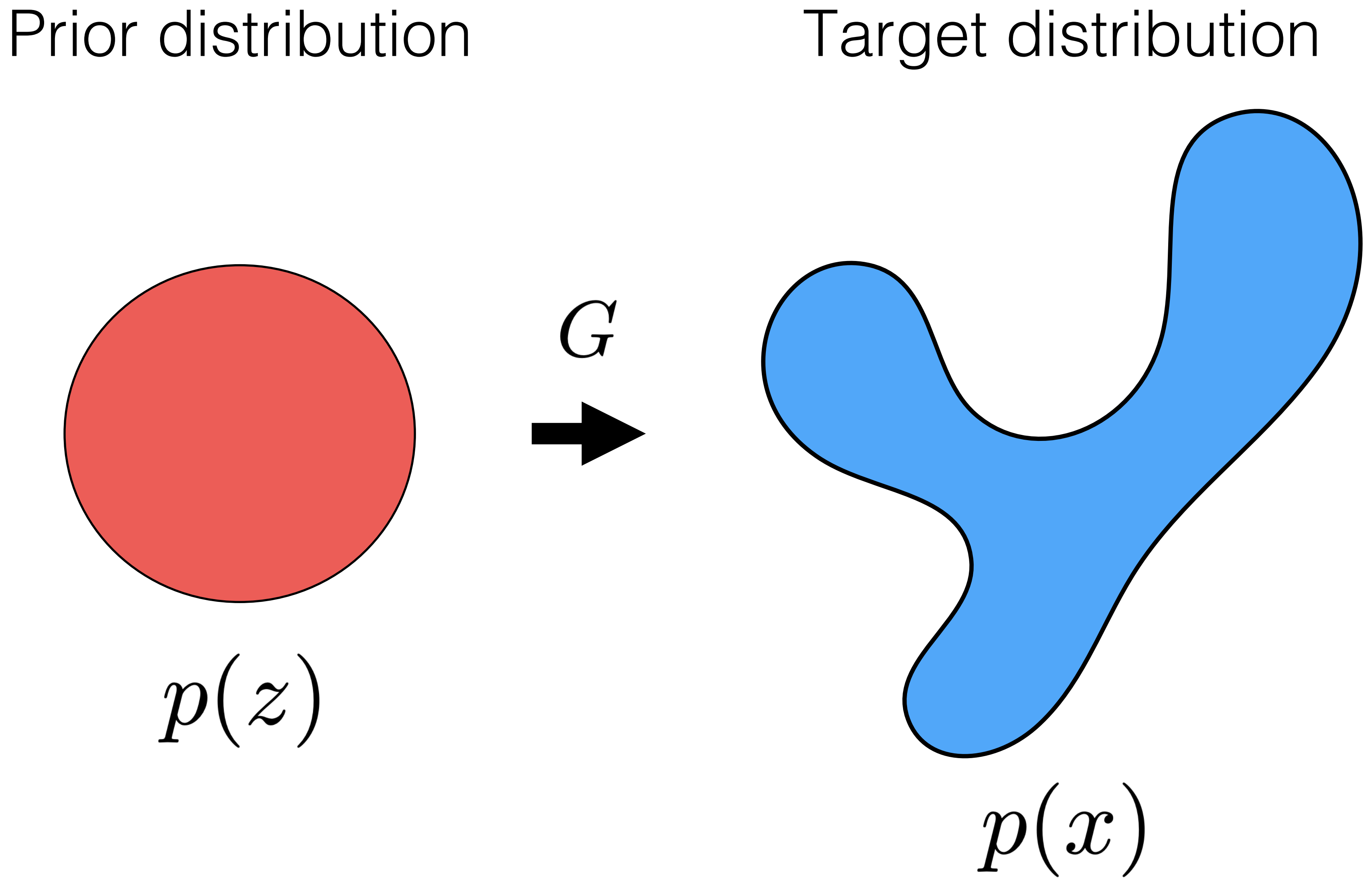
SGD

$\rightarrow$ Density

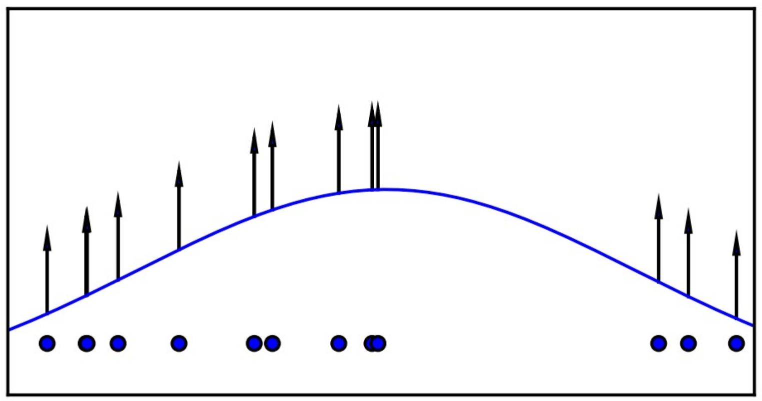
$p : \mathcal{X} \rightarrow [0, 1]$

Variational Autoencoders (VAEs)

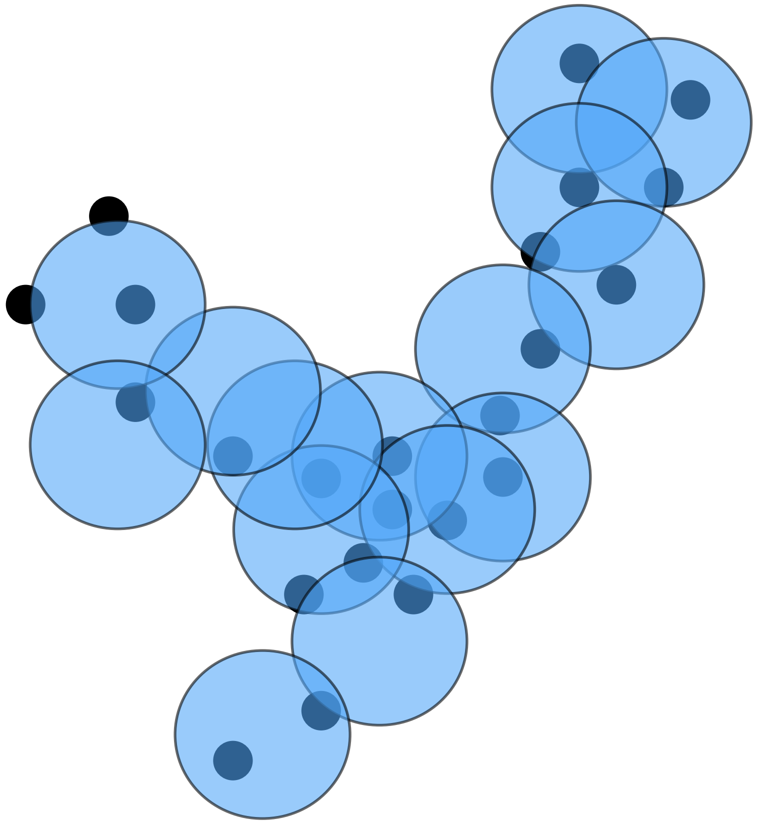
[Kingma & Welling, 2014; Rezende, Mohamed, Wierstra 2014]



Mixture of Gaussians



Target distribution



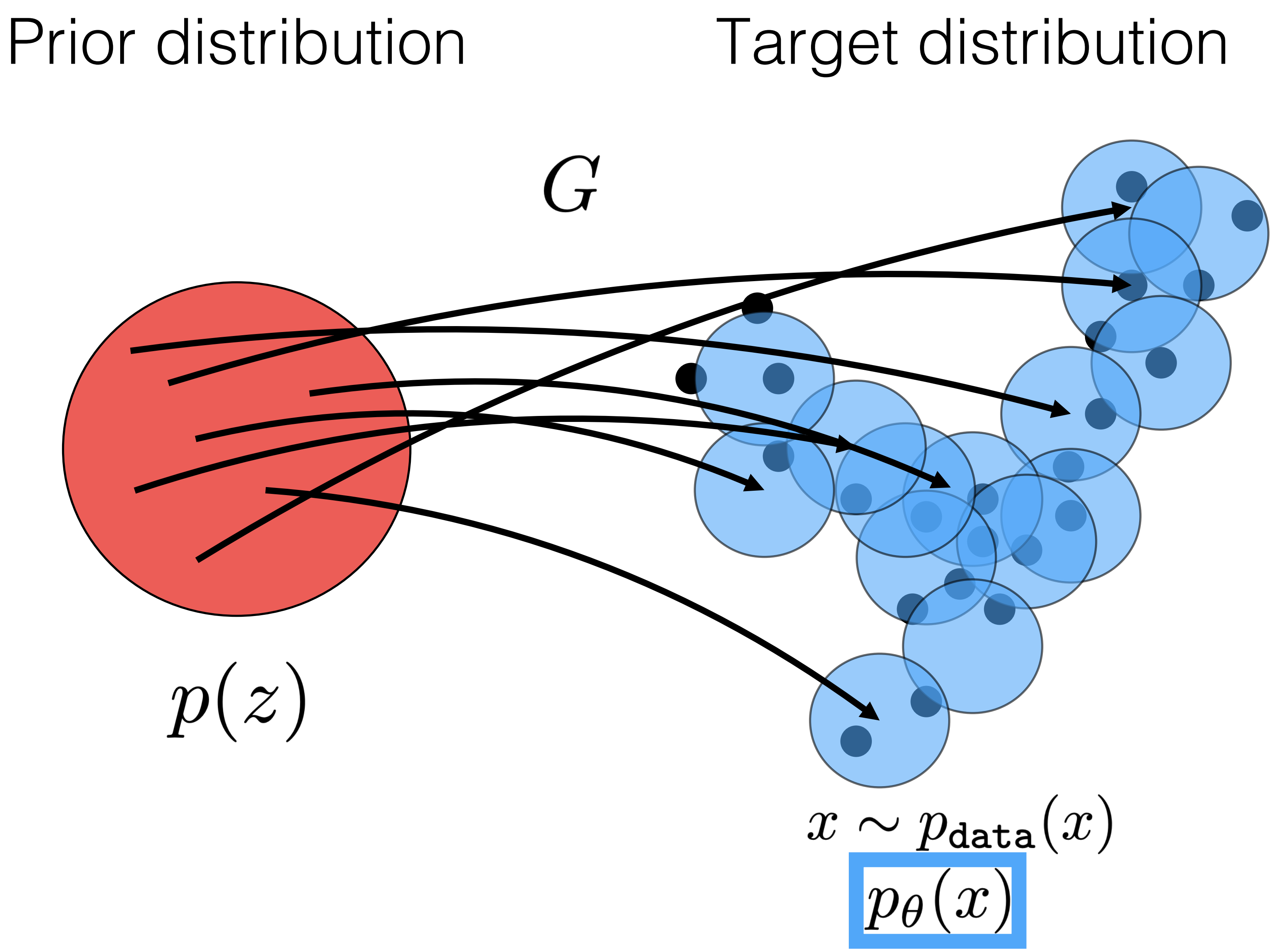
$$p_\theta(x) = \sum_{i=1}^{k} w_i \mathcal{N}(x; u_i, \Sigma_i)$$

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

$p_\theta(x)$

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 →

Learner

Objective

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

Hypothesis space

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

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

→

~~Density~~

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

Sampler

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

Variational Autoencoders (VAEs)

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

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

$$p_\theta(x) = \int p(x|z;\theta)p(z)dz$$ ⟵ 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 $\mathbb{E}_{q_\psi(z|x)}[p_\theta(x|z)]$

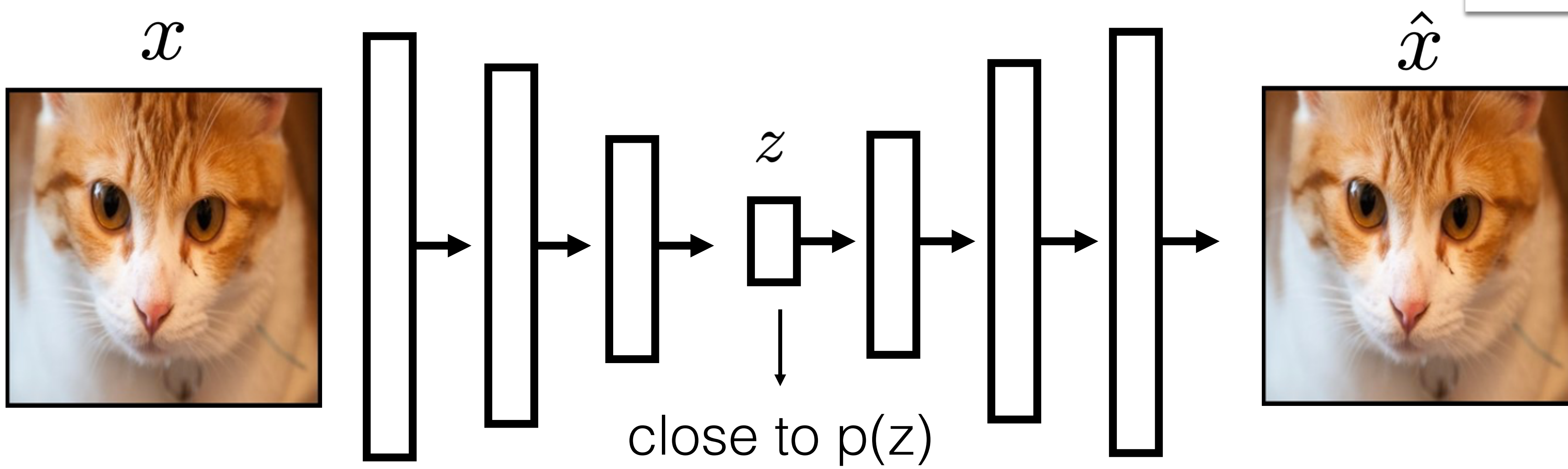
[Kingma and Welling, 2014]

Tutorial on VAEs [Doersch, 2016]

Variational Autoencoders (VAEs)

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

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



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

Multi-variate Gaussian

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

reconstruction loss

KLD loss

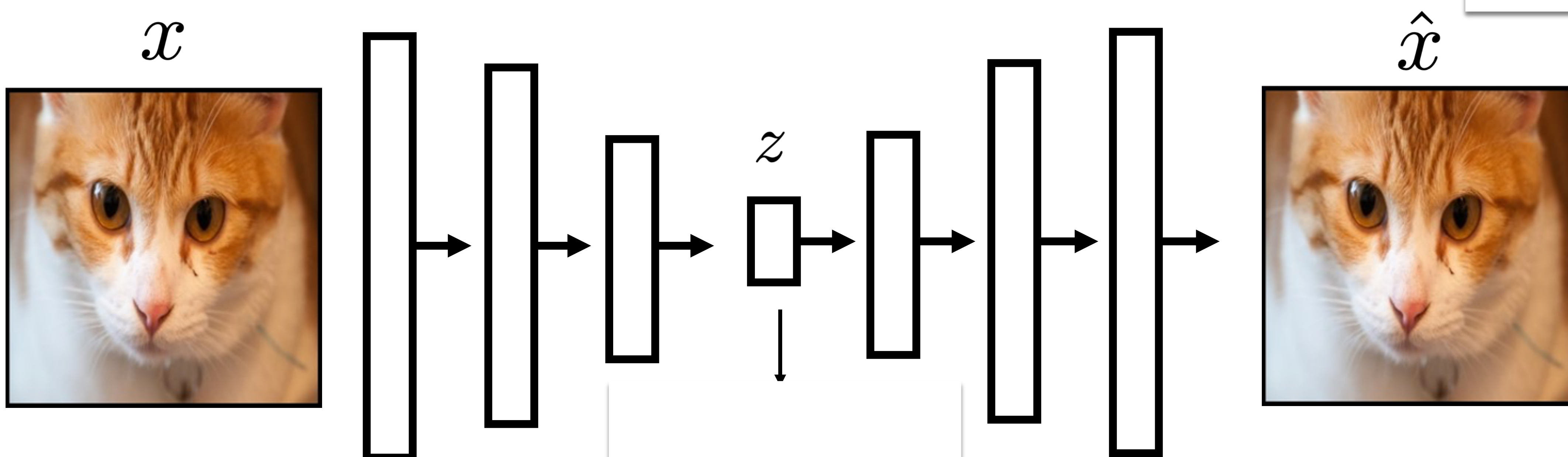
$$||x - \hat{x}||_2 \quad \text{KLD}(\mathcal{N}(E^\mu_\psi(x), E^\sigma_\psi(x)) \mid \mathcal{N}(0, I))$$

[Kingma and Welling, 2014]

Autoencoders (AEs)

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

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



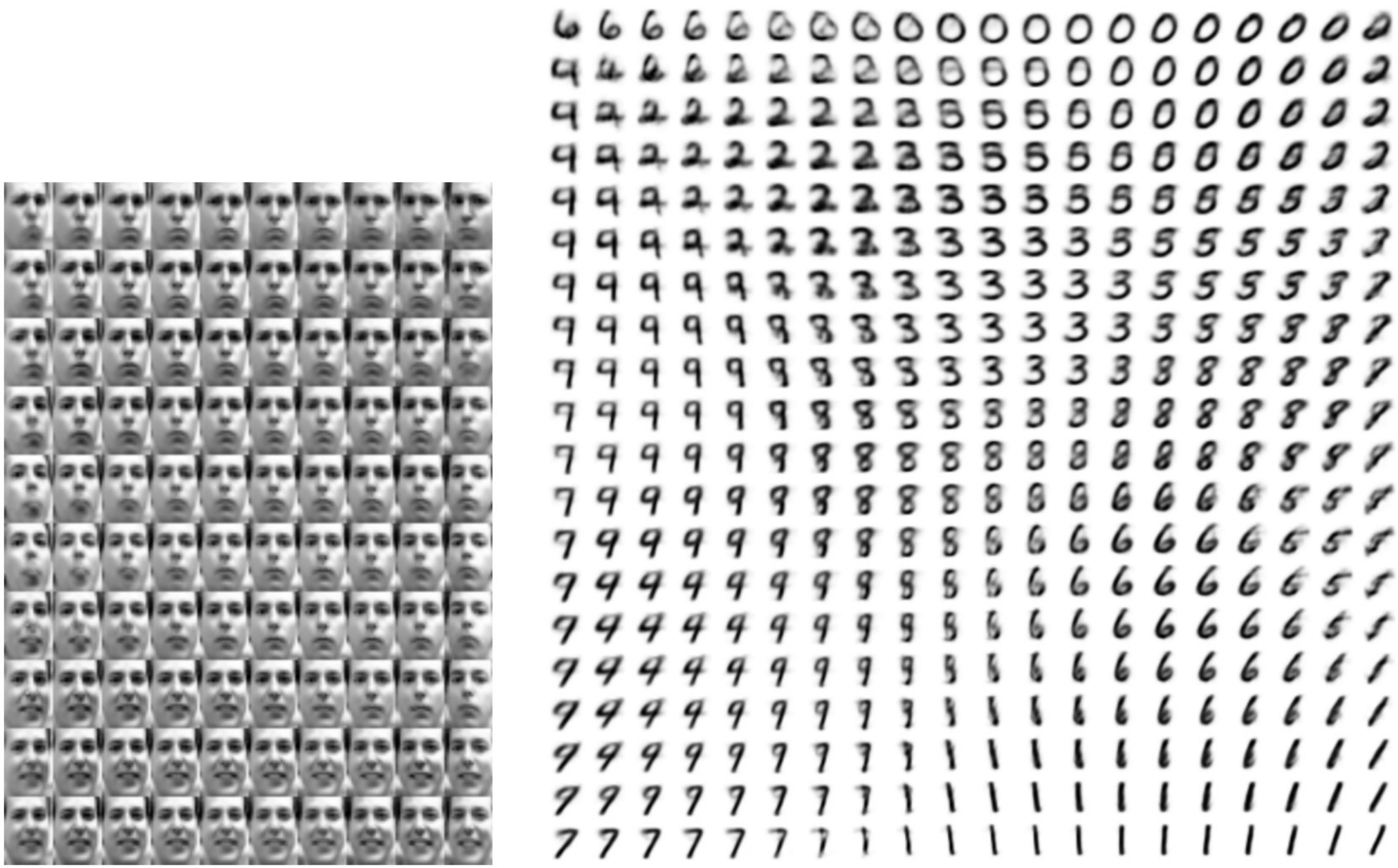
$$\max_{\theta,\psi} \mathbb{E}_{x_i \sim p_{\text{data}}}[\mathbb{E}_{q_\psi(z|x_i)}[p_\theta(x|z)]]$$

reconstruction loss

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

[Hinton and Salakhutdinov, Science 2006]

Variational Autoencoders (VAEs)



VAE with two-dimensional latent space

[Kingma and Welling, 2014]

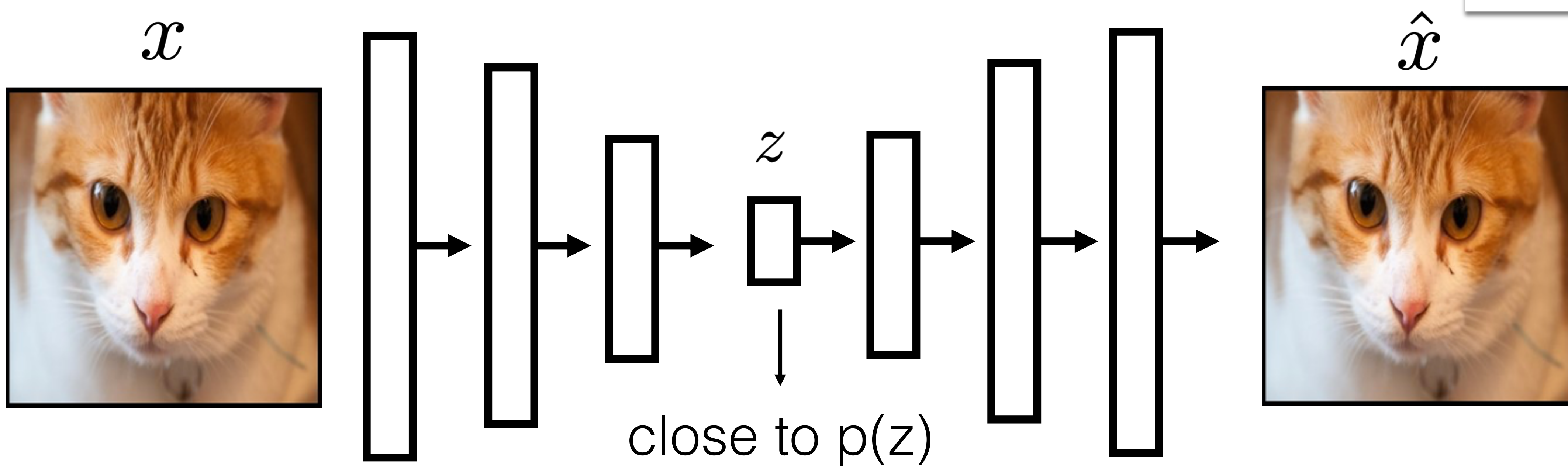
How to improve VAE?

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



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

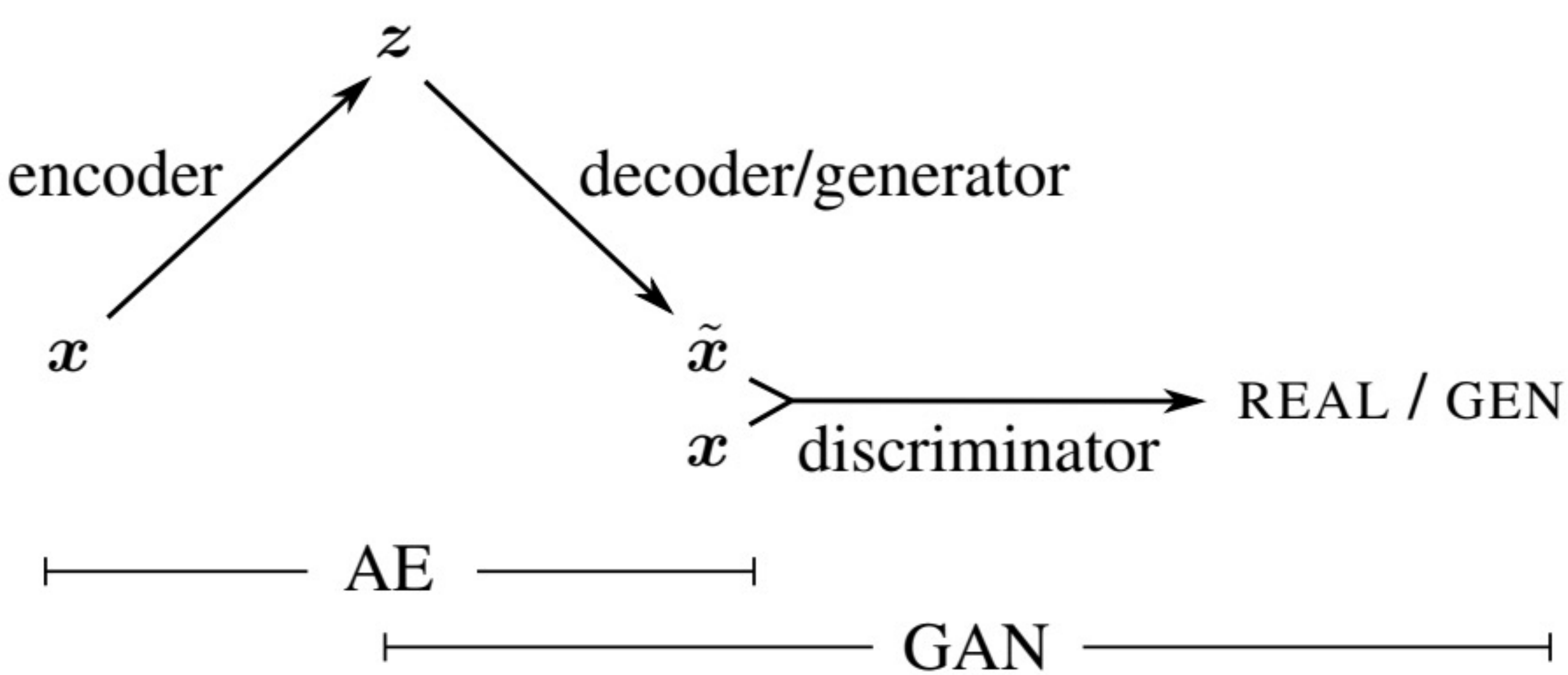
Multi-variate Gaussian

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

Perceptual loss KLD loss

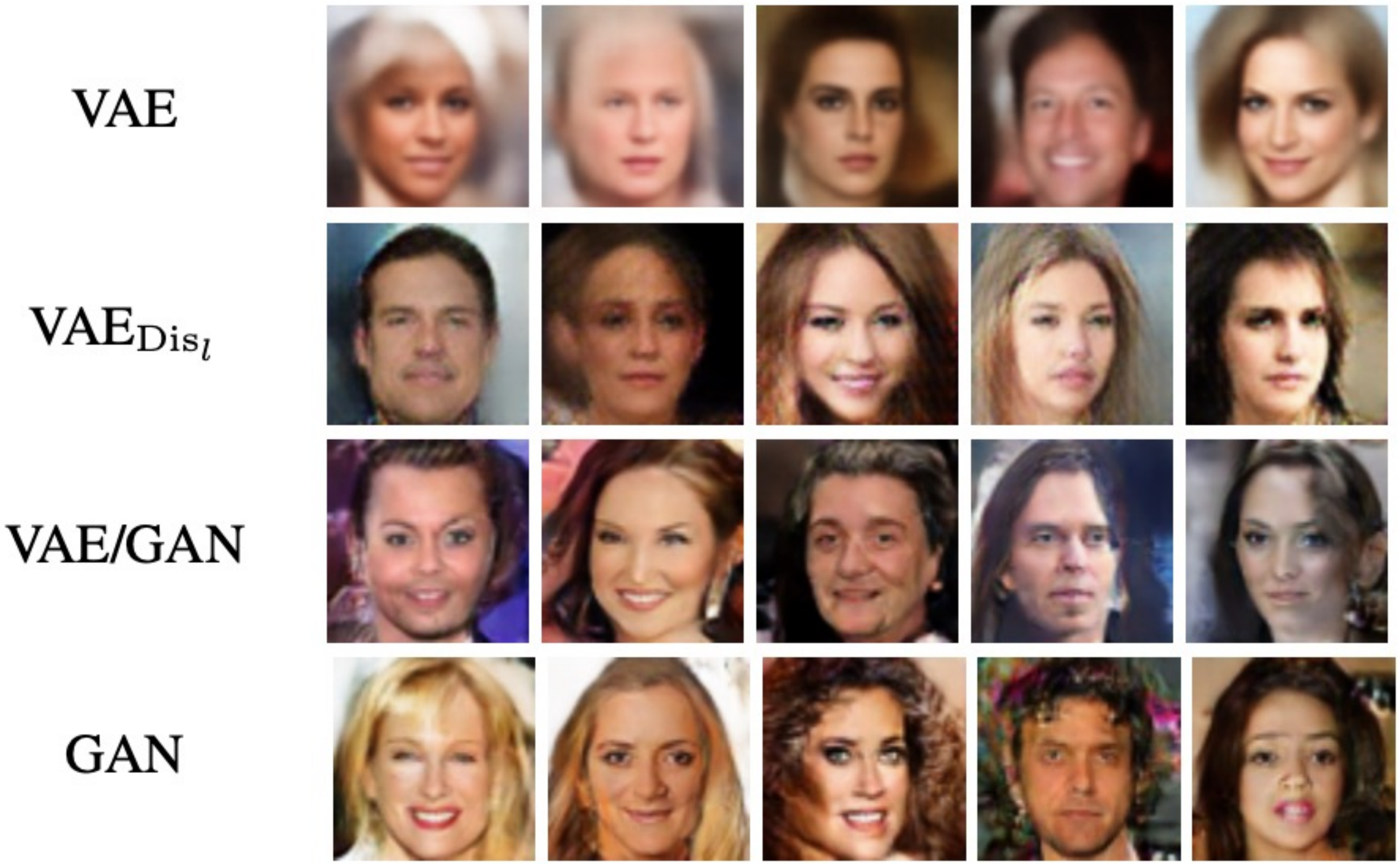
$||F(x) - F(\hat{x})||_2$ $\mathrm{KLD}(\mathcal{N}(E^\mu_\psi(x), E^\sigma_\psi(x)) \mid \mathcal{N}(0, I))$

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

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

Optimizer

Variational Bayes

$\longrightarrow$

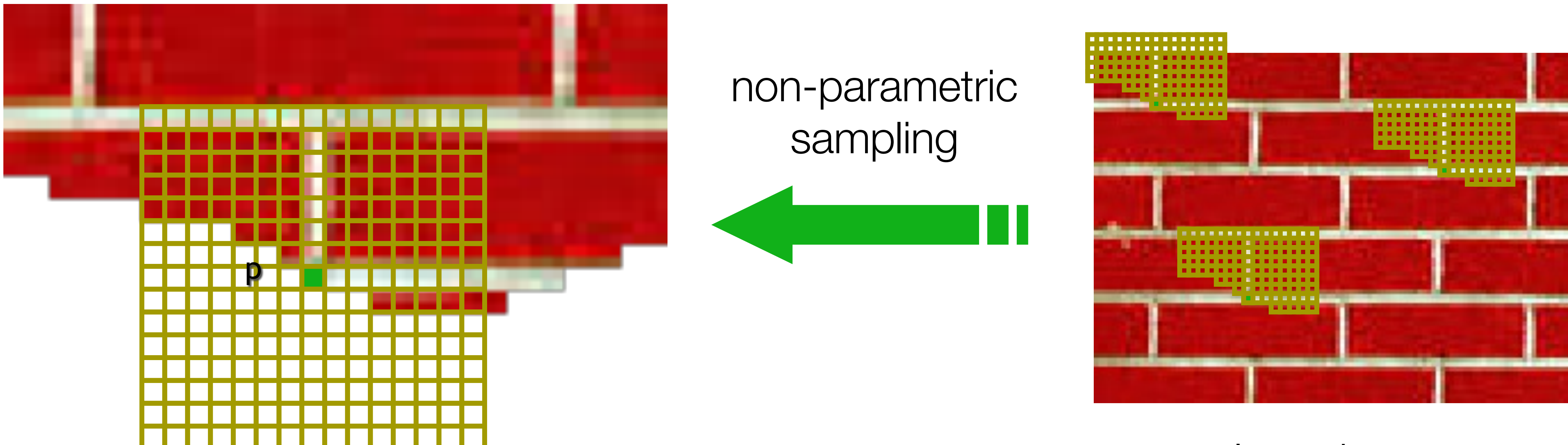
Sampler

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

Autoregressive Model

Texture synthesis by non-parametric sampling

[Efros & Leung 1999]

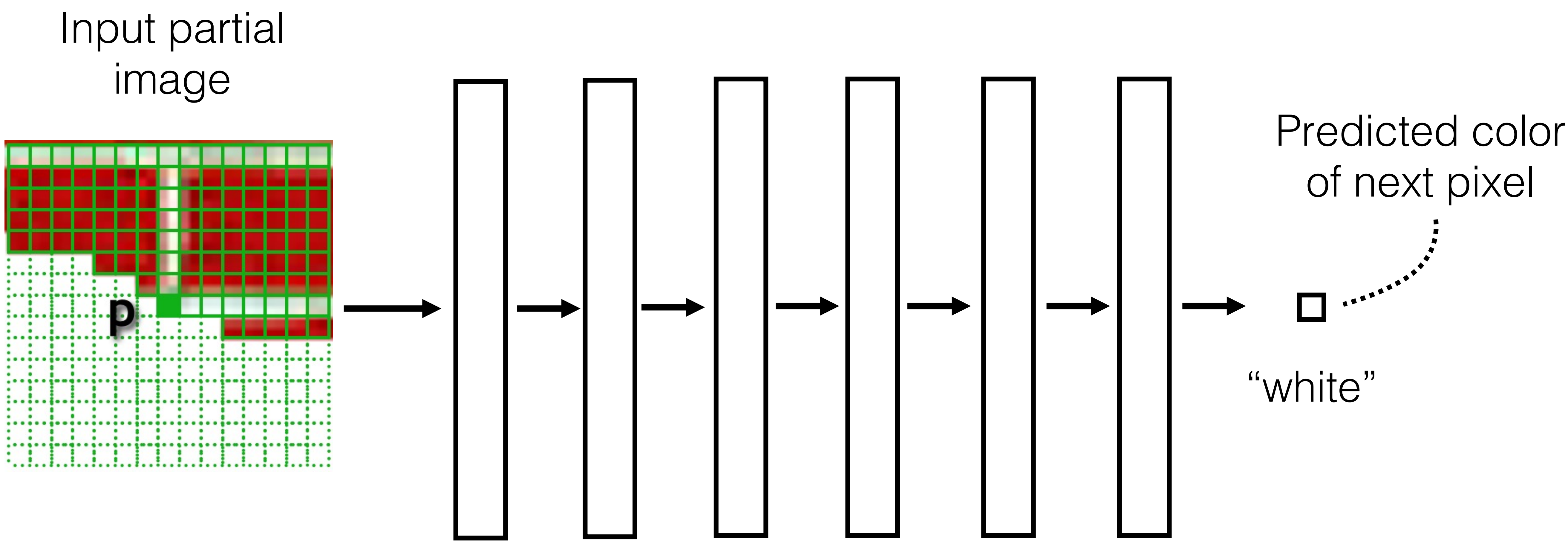


Synthesizing a pixel

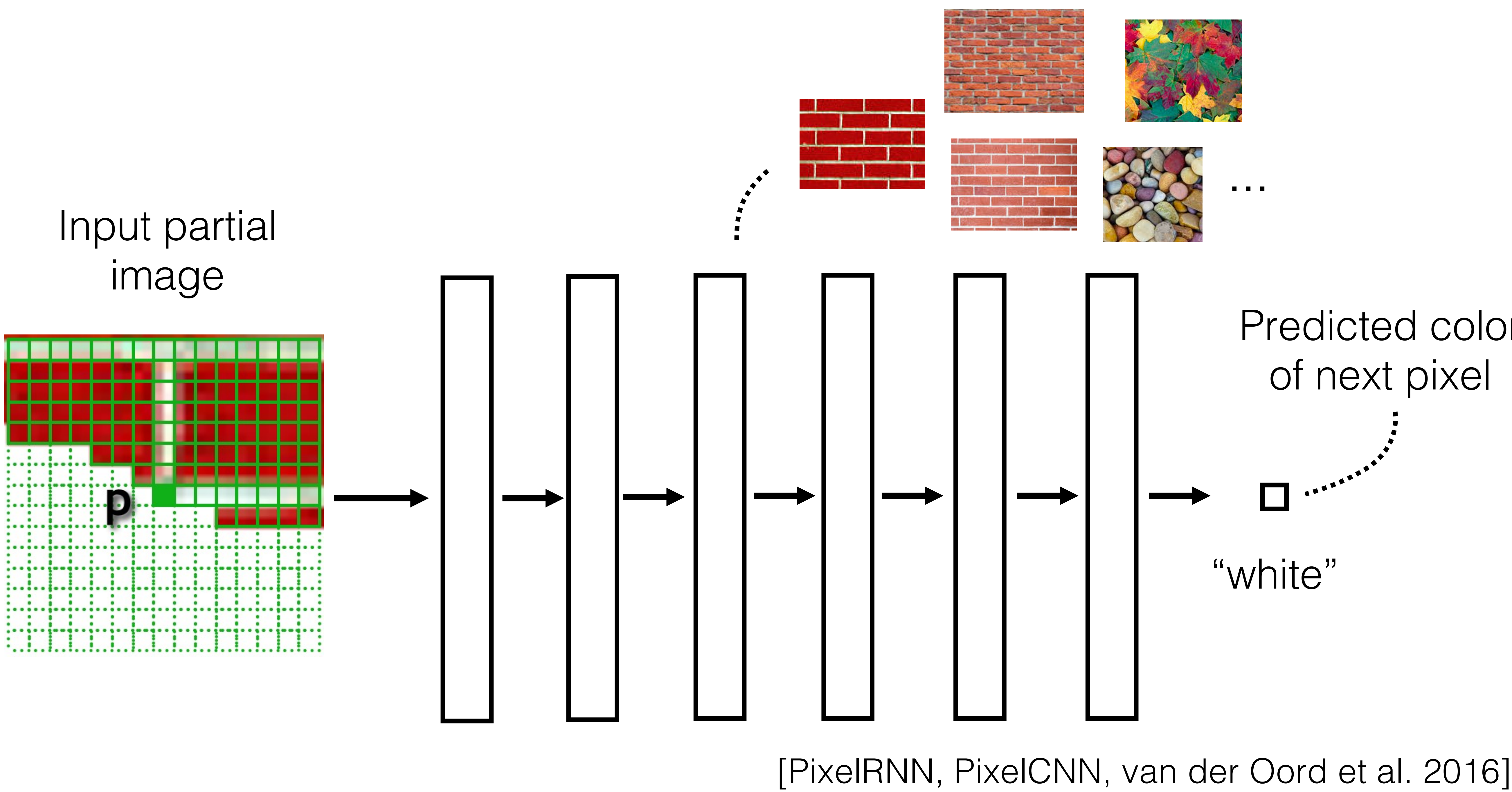
Input image

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

Autoregressive image synthesis

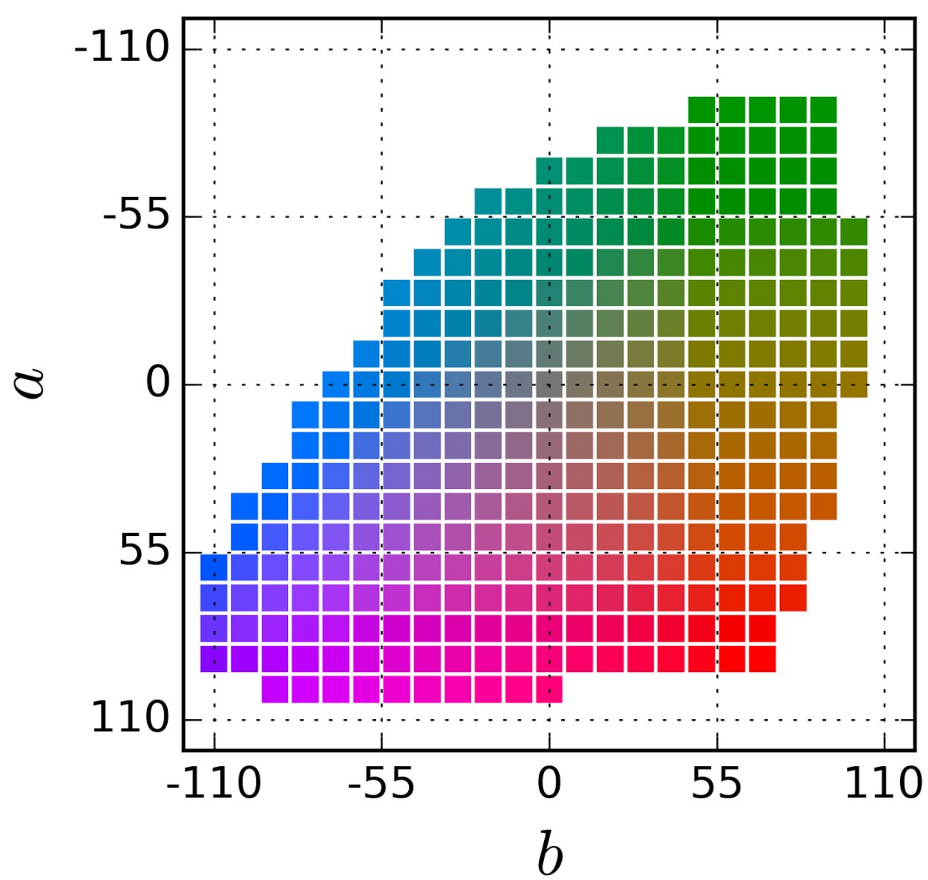


[PixelRNN, PixelCNN, van der Oord et al. 2016]

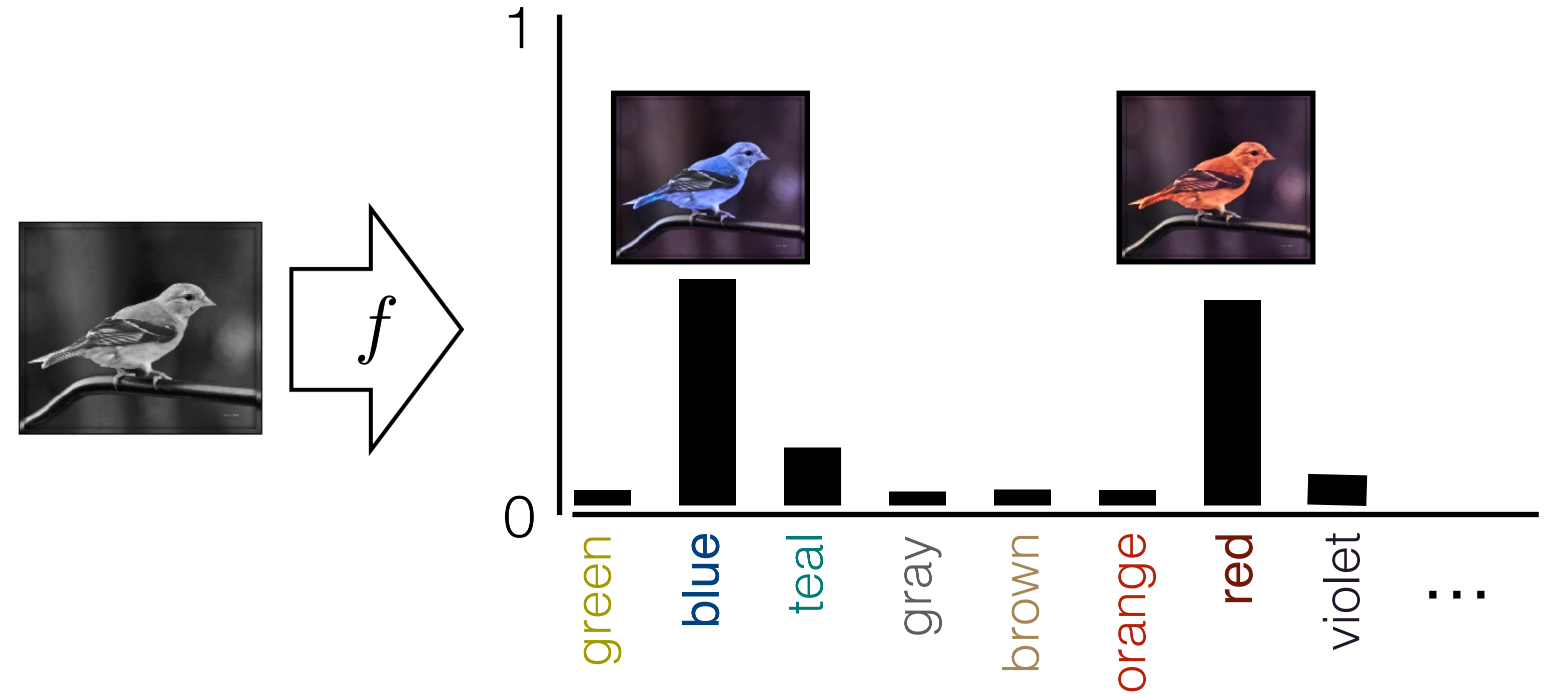

Input partial image
p
...
Predicted color of next pixel
"white"
[PixelRNN, PixelCNN, van der Oord et al. 2016]

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

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

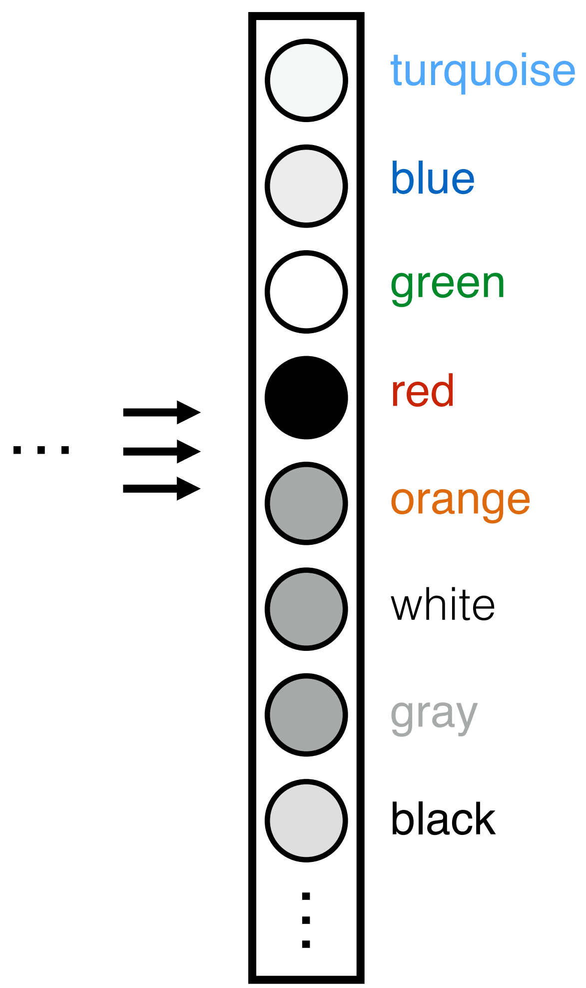
← picks out the -log likelihood of the ground truth class $\mathbf{y}$ under the model prediction $\hat{\mathbf{y}}$

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

← max likelihood learner!

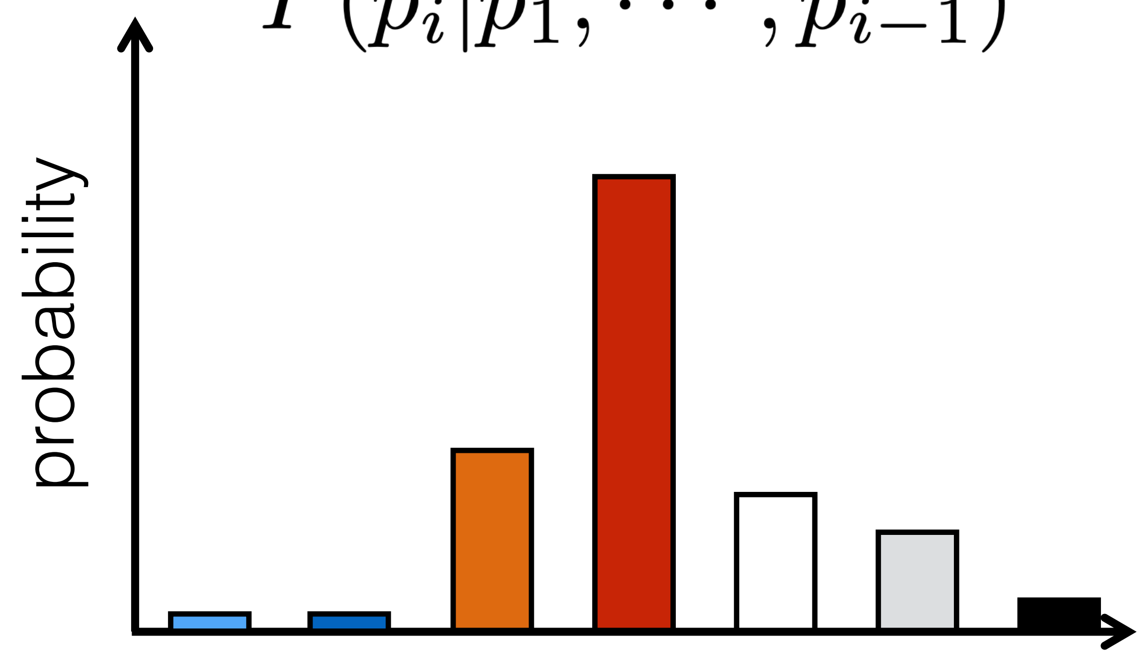
Cross-entropy loss

Network output

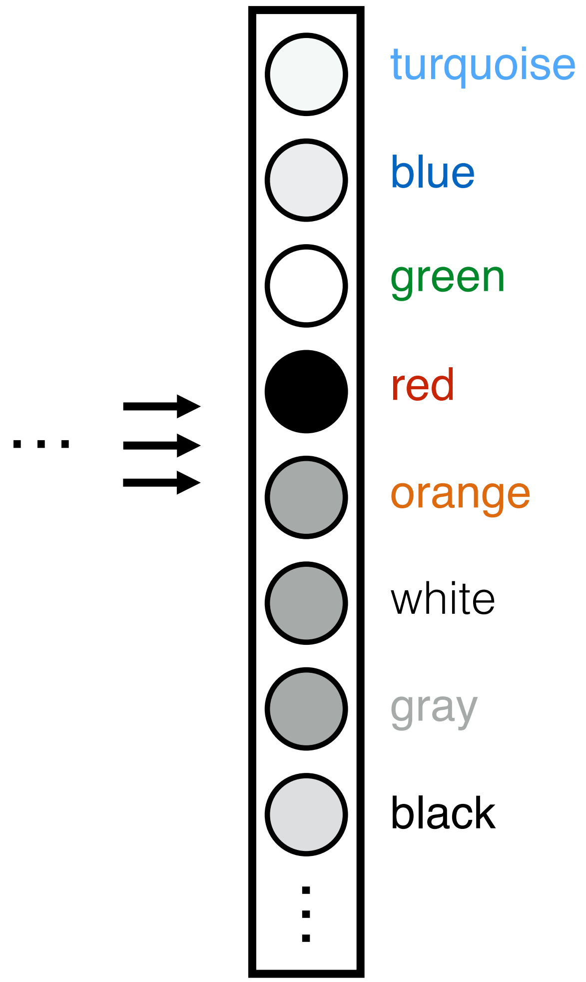


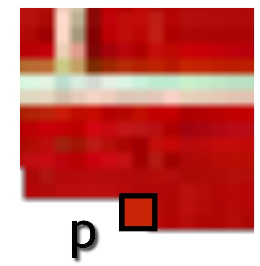
P(next pixel | previous pixels)

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

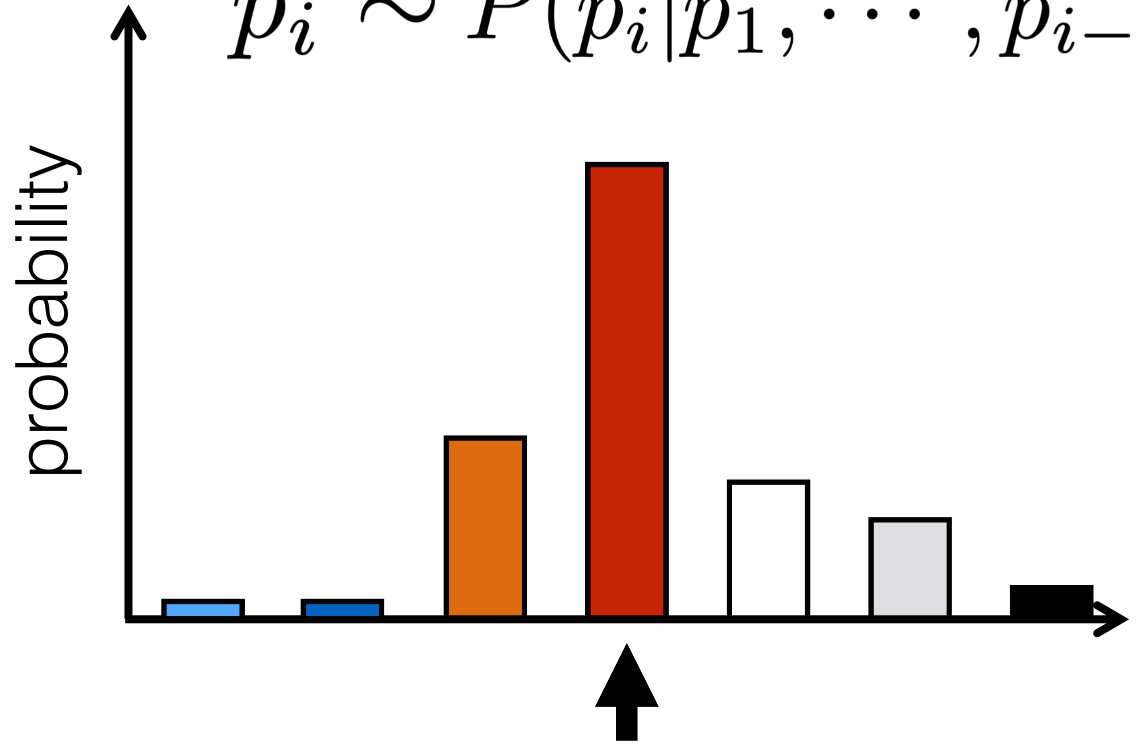


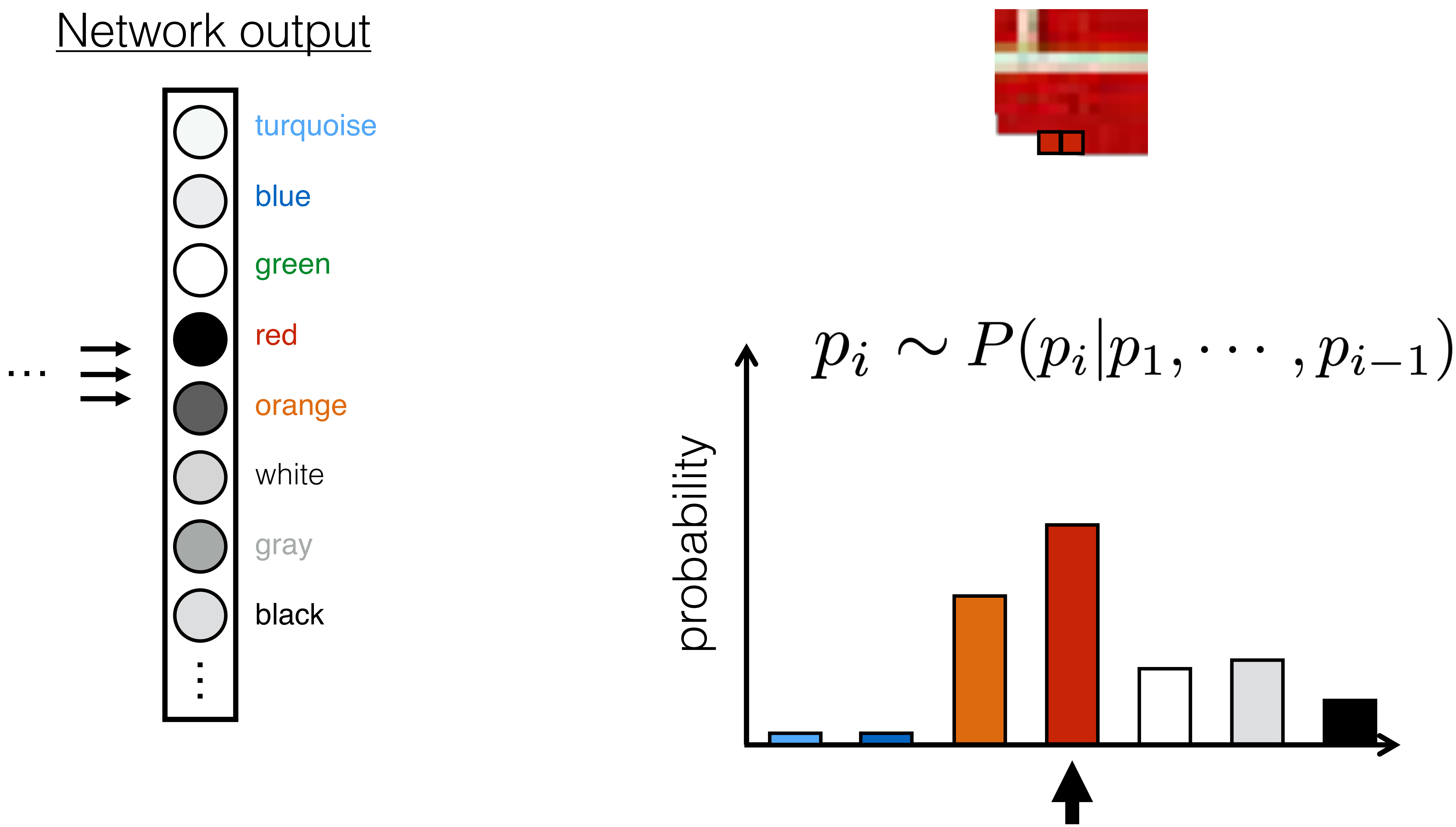
Network output



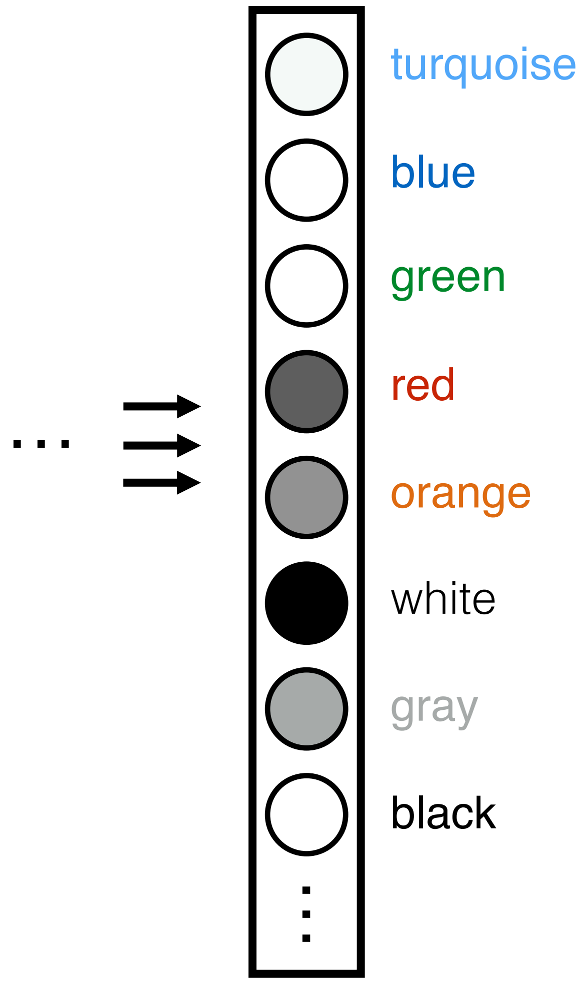


$$p_i \sim P(p_i|p_1, \cdots, p_{i-1})$$

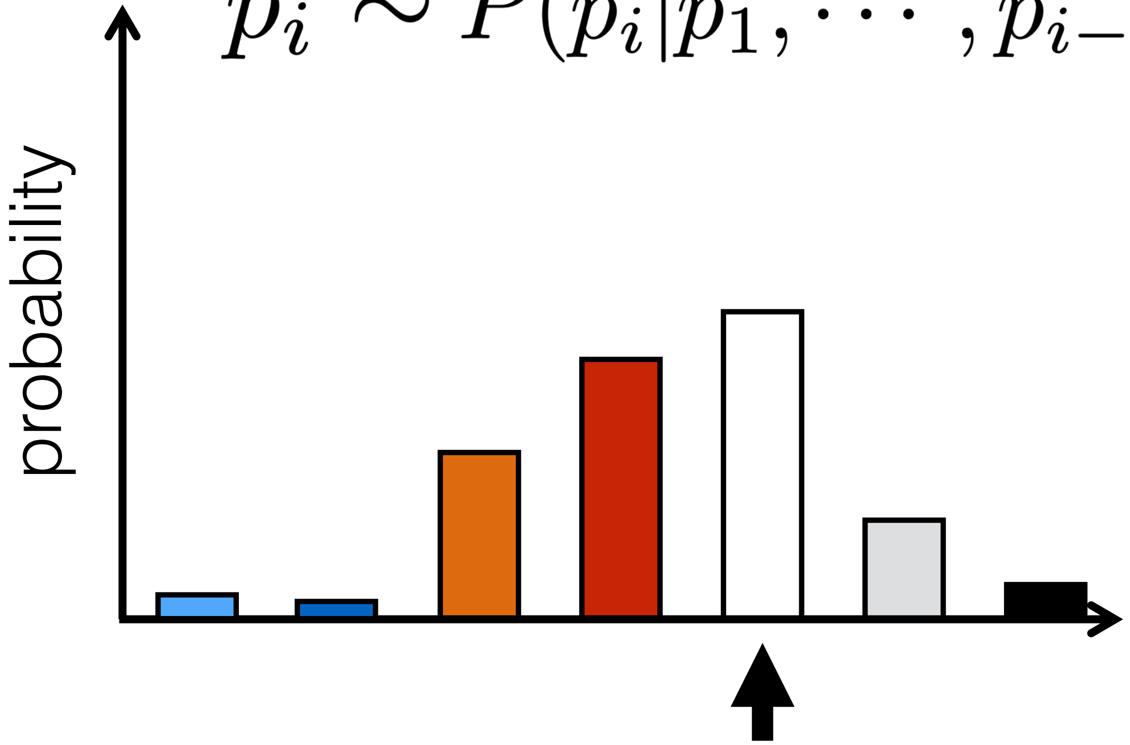



Network output
turquoise
blue
green
red
orange
white
gray
black
$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$
probability

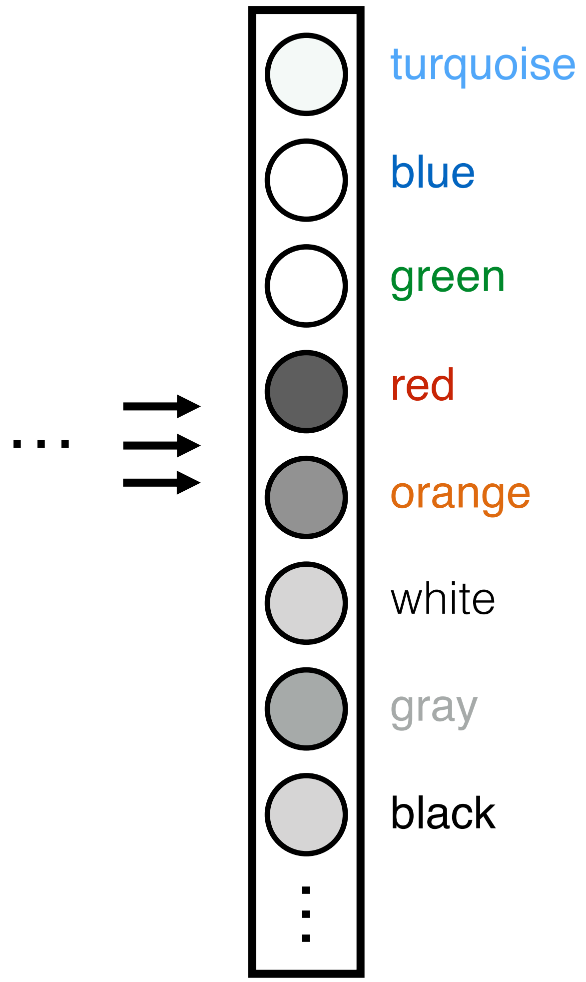
Network output

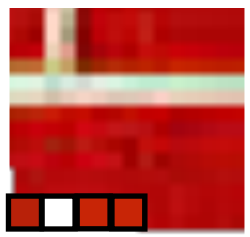


$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$

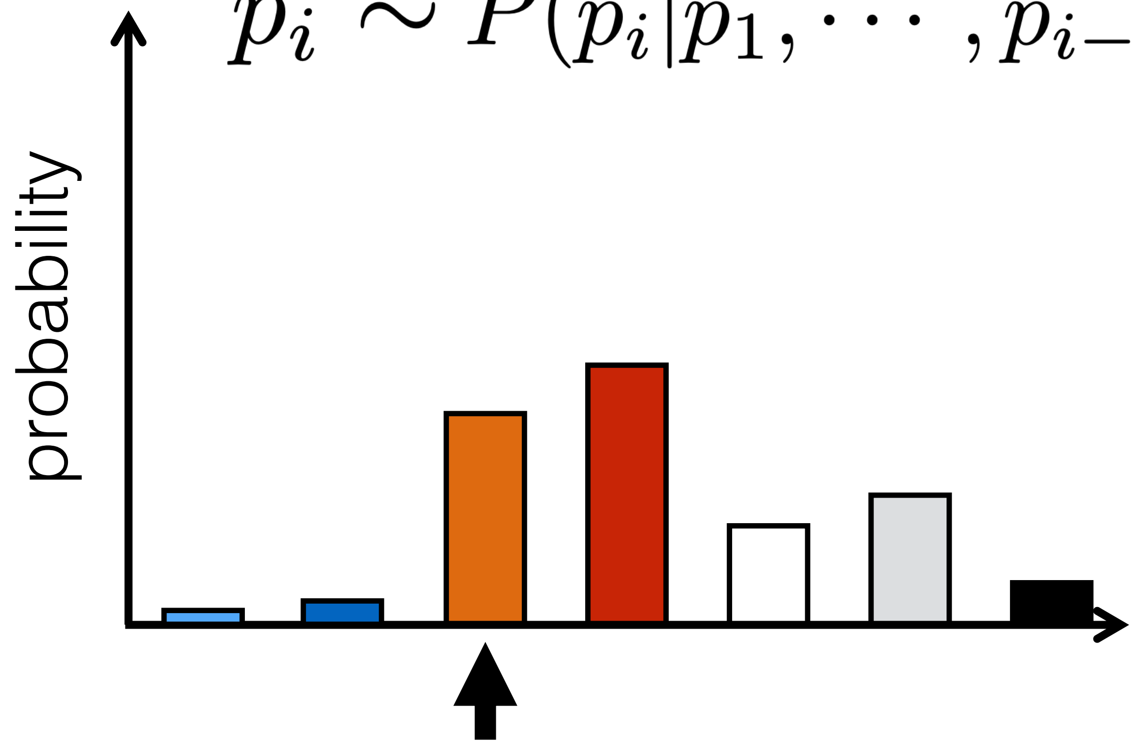


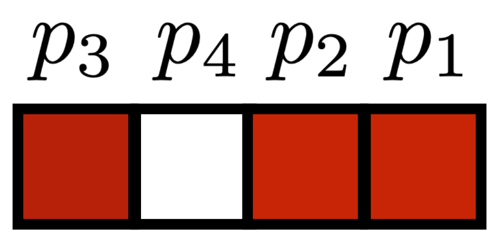
Network output





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





$$p_1 \sim P(p_1)$$

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

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

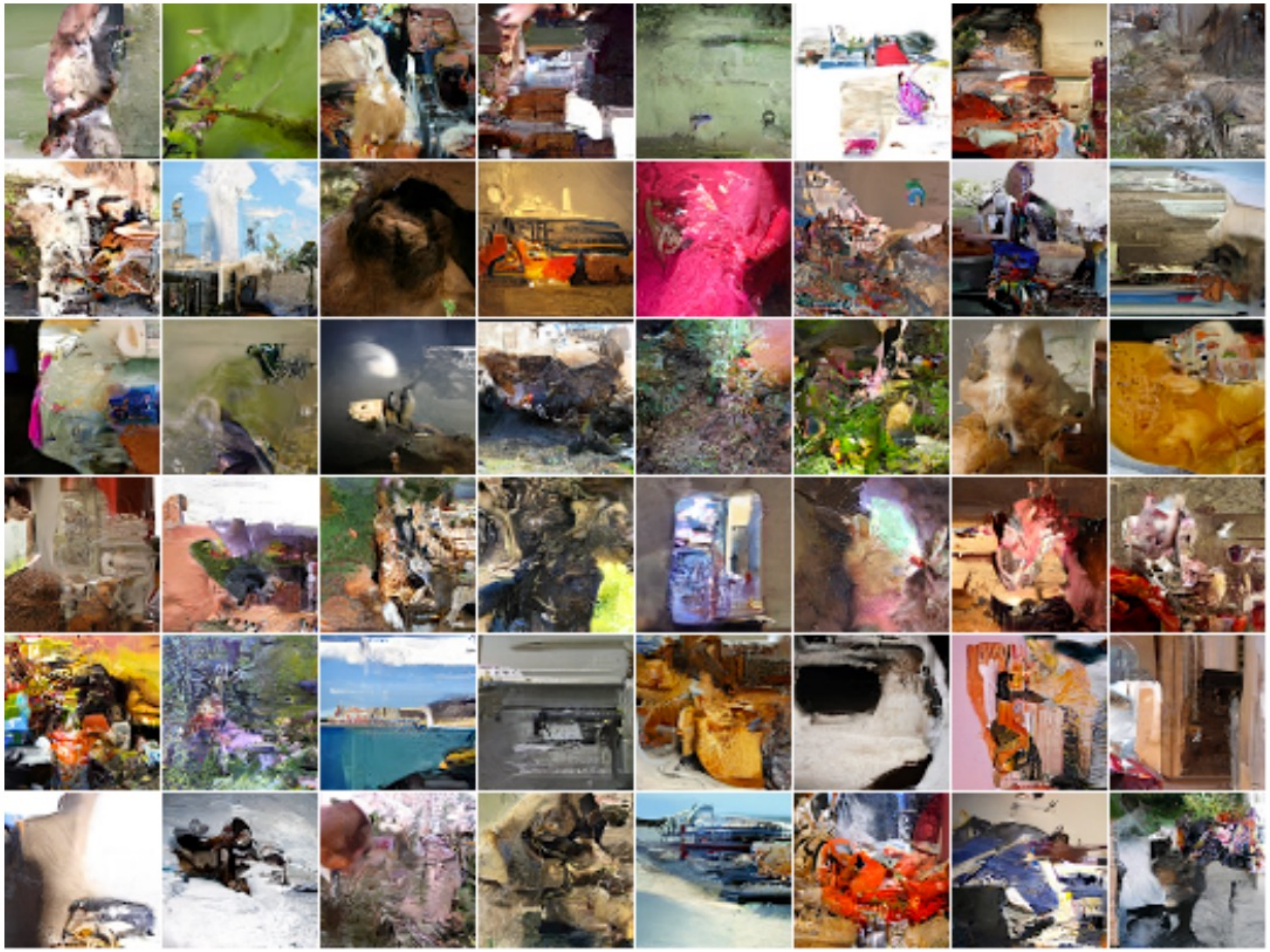
$$p_4 \sim P(p_4|p_1, p_2, p_3)$$

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

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

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

Samples from PixelRNN

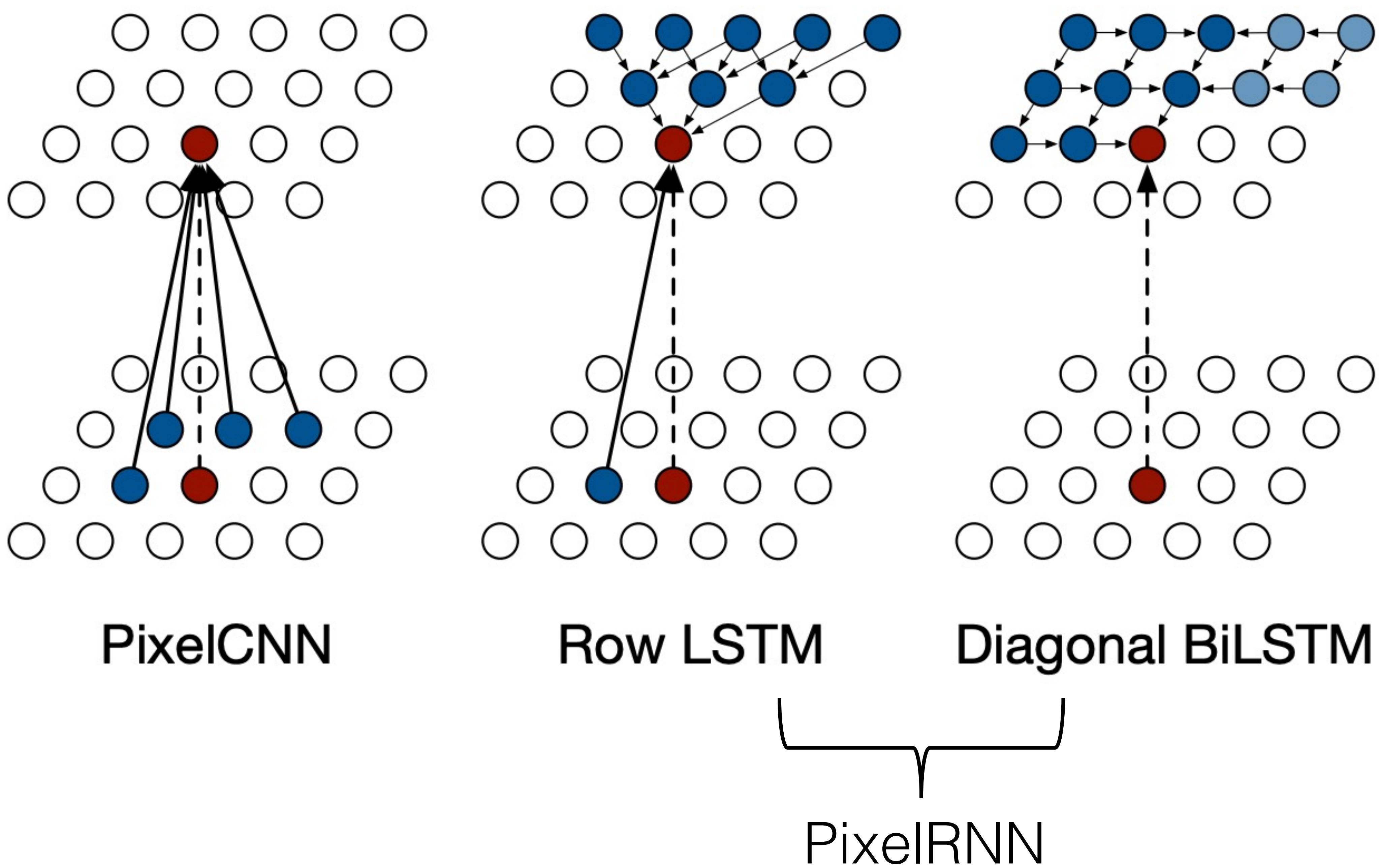


[PixelRNN, van der Oord et al. 2016]

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?

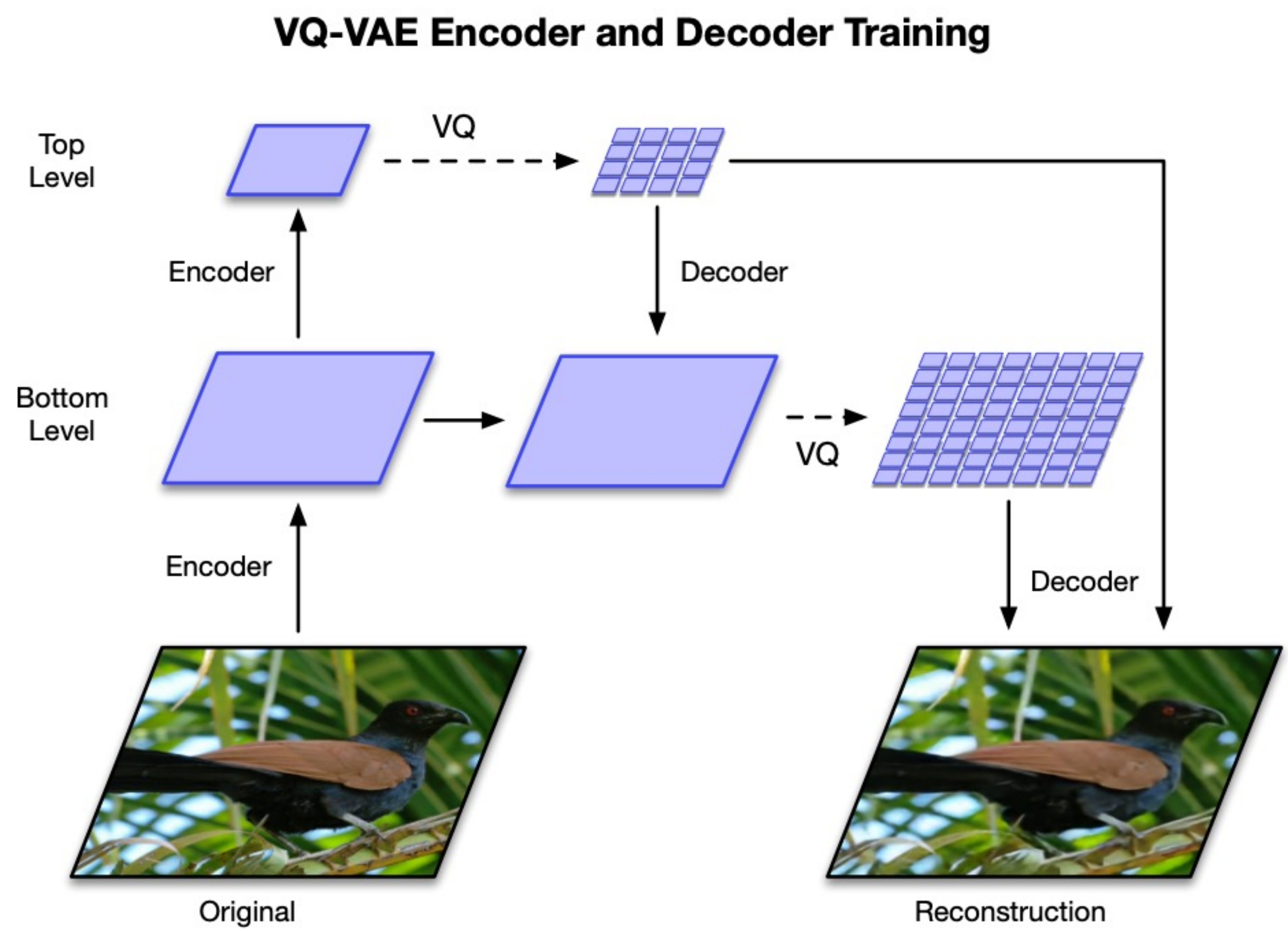
- What are the limitations of PixelCNN/RCN?
  - Slow sampling time.
  - May accumulate errors over multiple steps. (might not be a big issue for image completion)
- How can we further improve results?

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]

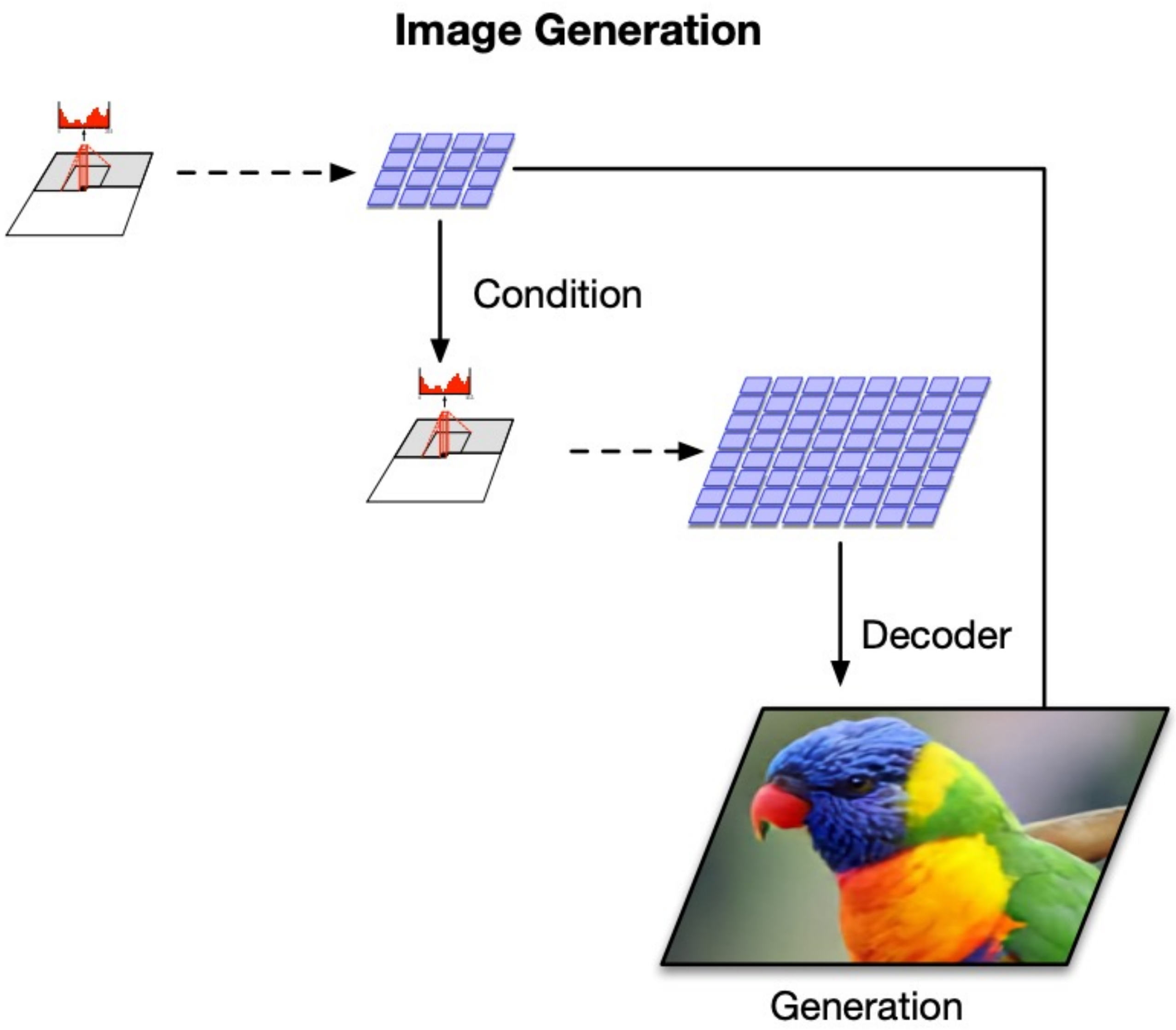
VQ-VAE-2: VAE+PixelCNN



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]

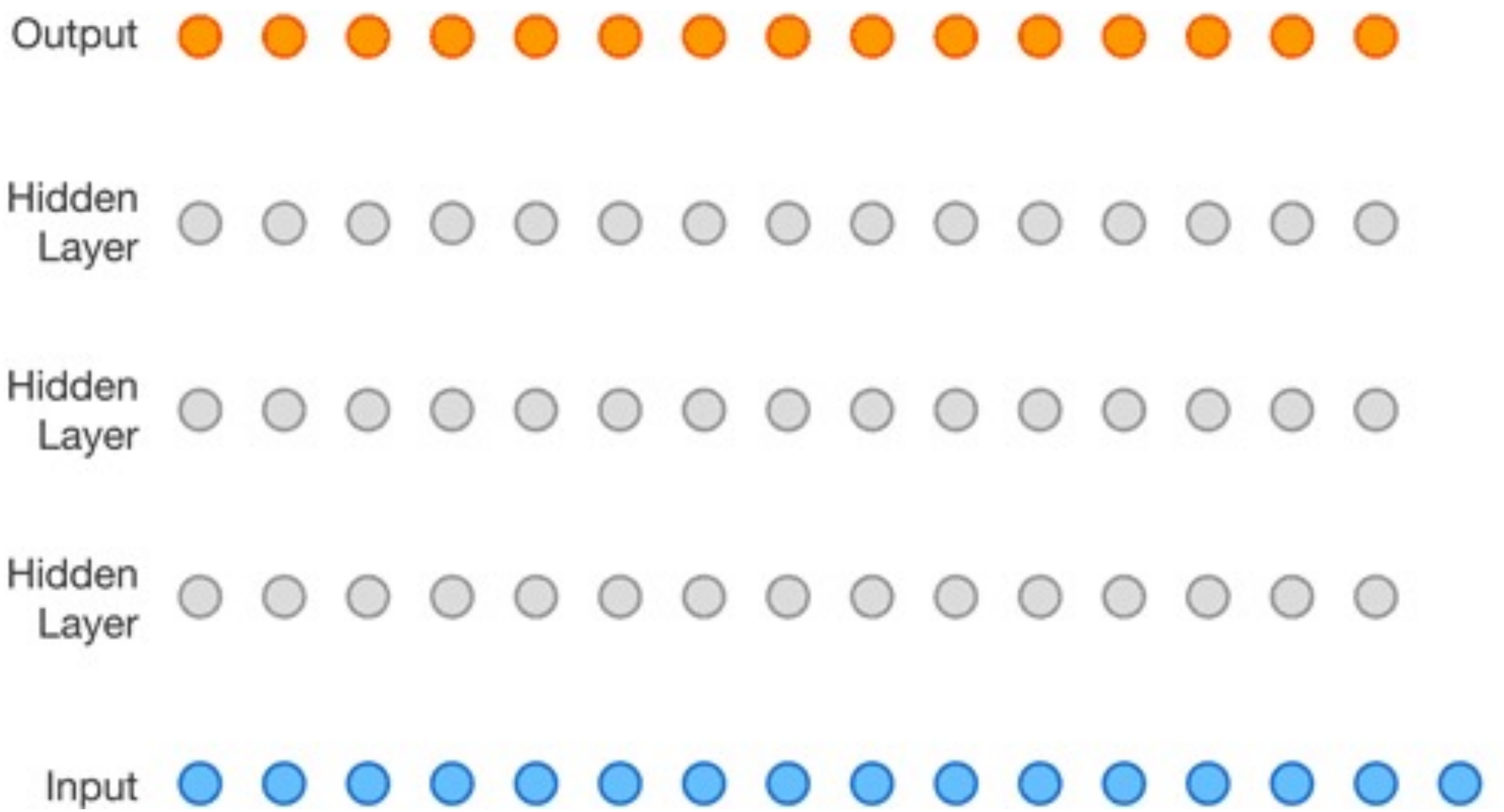
VQ-VAE-2: VAE+PixelCNN



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]

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

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

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

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

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.

Flow-based models

Data $\rightarrow$

Learner

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

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

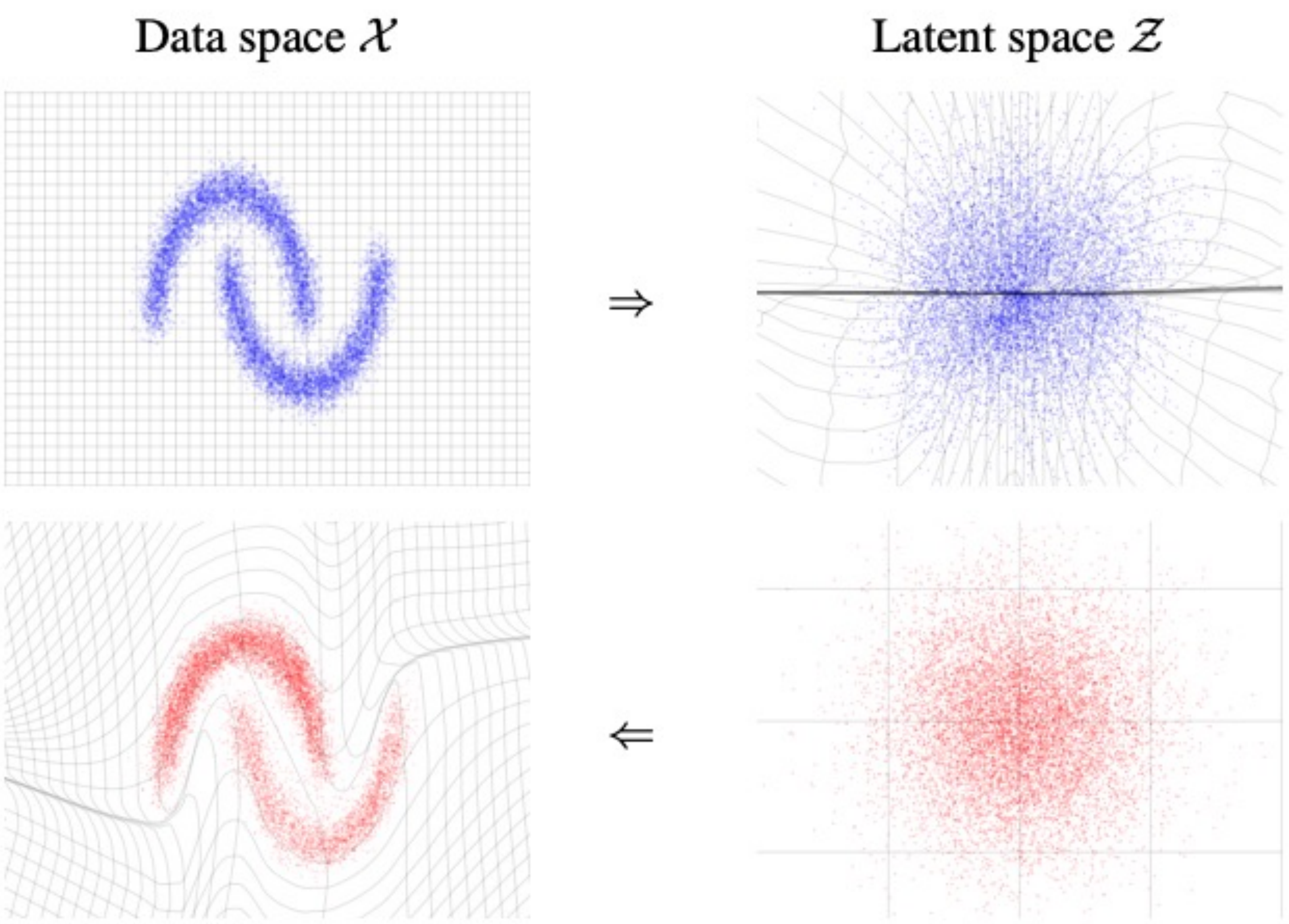
$$z = f(x)$$

- 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

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

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

Easy to compute
as z follows Gaussian distribution

hard to compute
Jacobian determinant
for most layers

design layers whose Jacobian determinant
is a triangular matrix

[Dinh et al., 2016]

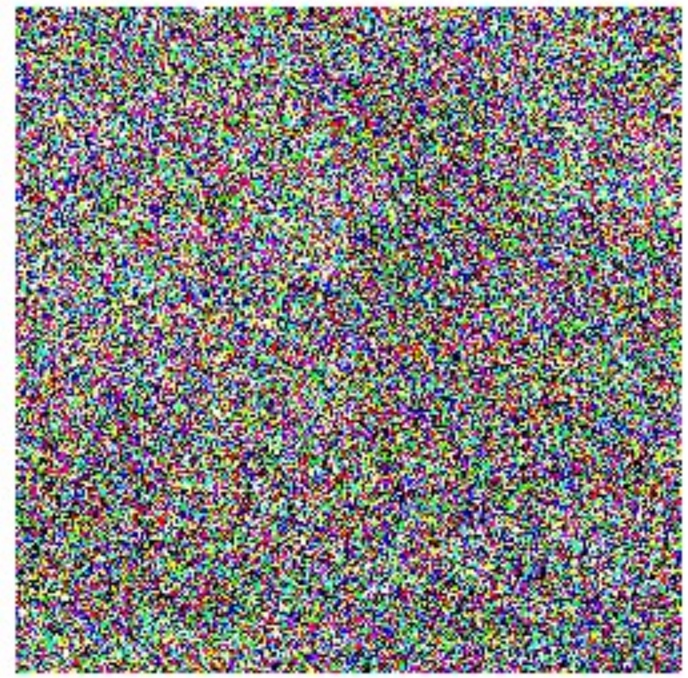
Flow-based models

Reading list

Real NVP [Dinh et al., ICLR 2017]

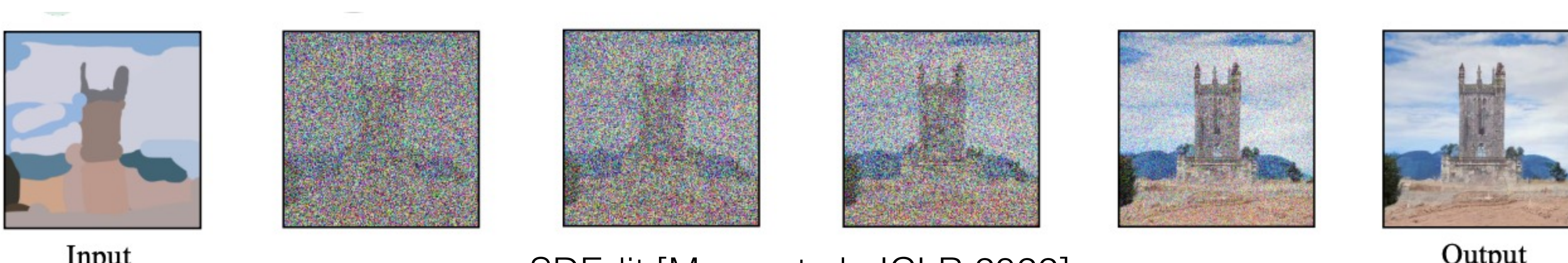
Glow [Kingma and Dhariwal, NeurIPS 2018]

Diffusion Model



Add Gaussian noise Learn to denoise

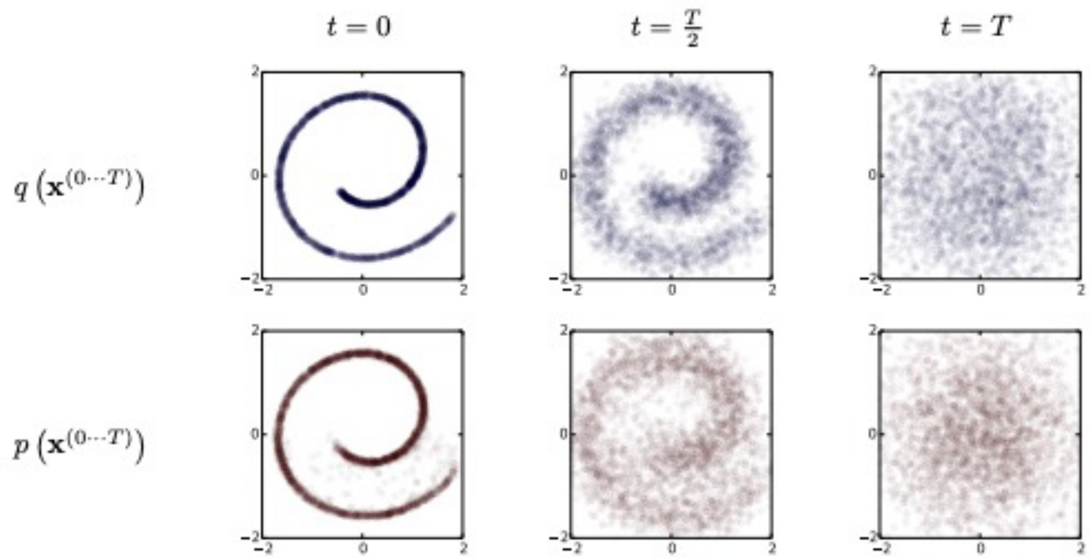
From the blog: https://yang-song.github.io/blog/2021/score/



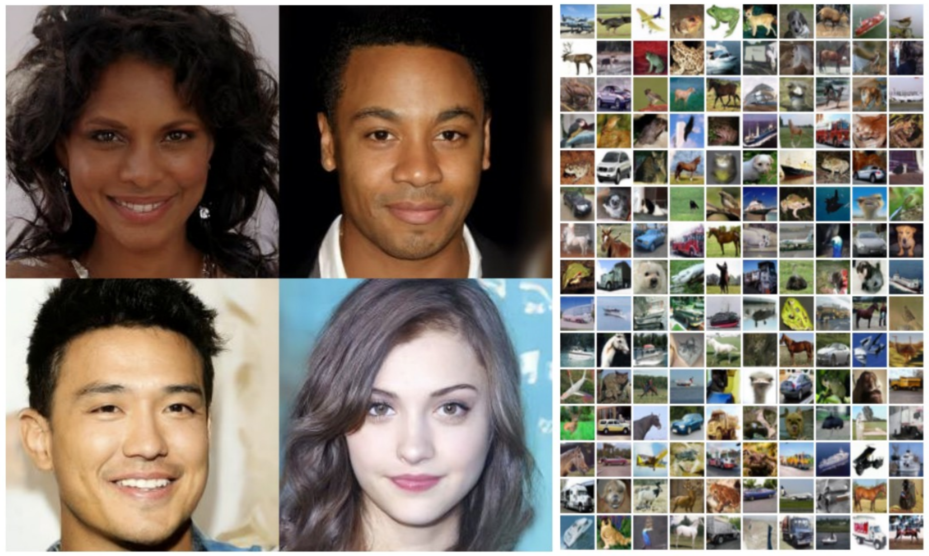
SDEdit [Meng et al., ICLR 2022]

Diffusion Model

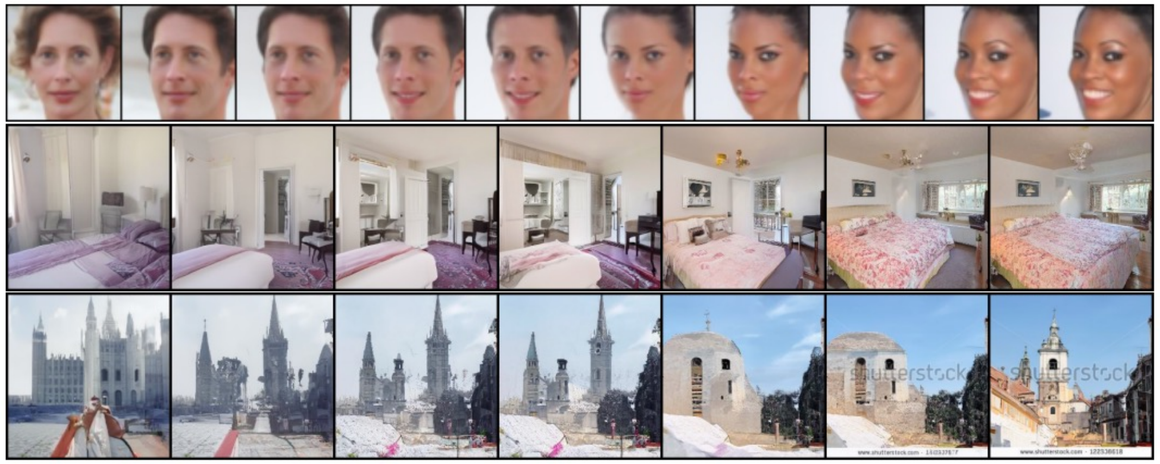
Reading list



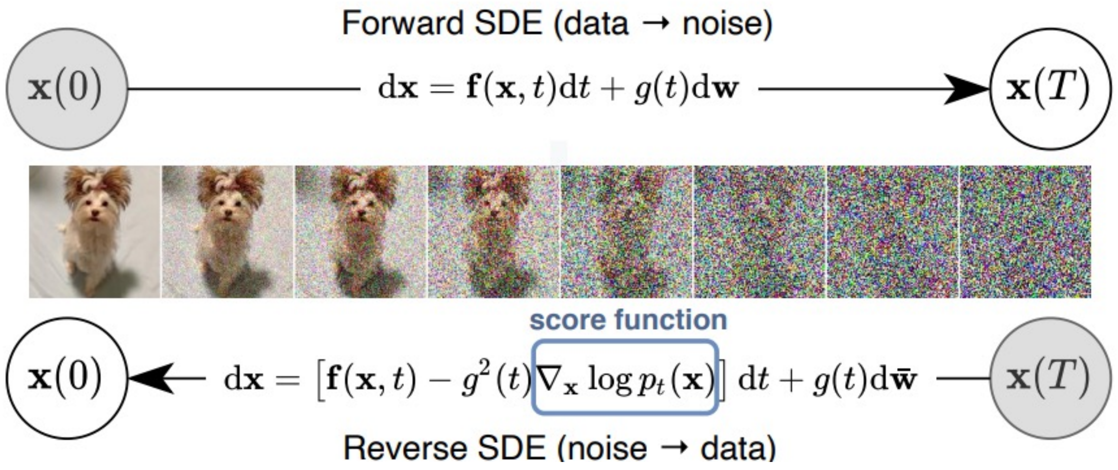
Diffusion model [Sohl-Dickstein et al., ICML 2015]



DDPM [Ho, Jain, Abbeel, NeurIPS 2020]



DDIM [Song, Meng, Ermon, ICLR 2021]



Score-based Model [Song et al., ICLR 2021]

Ideal models (Dream)

Pros: good sample, fast sample, Exact/fast likelihoods
good coverage, easy to training, learn low-dimensional latent representation.

Autoregressive models

Pros: Exact likelihoods, good coverage
Cons: Slow to evaluate or sample

VAEs

Pros: Cheap to sample, good coverage

Cons: Blurry samples (in practice)

GANs

Pros: Cheap to sample, fast to train, good samples

Cons: No likelihoods (density), bad coverage (mode collapse)

Flow-based models

Pros: Cheap to sample, exact likelihoods

Cons: memory-intensive; slow training; limited choices for generators,
high-dimensional codes

Diffusion models

Pros: good samples, good coverage

Cons: slow training, slow sampling

[adapted from Phillip Isola and David Duvenaud]

Which model is better?

- It depends on your applications
  - Synthesis
  - Classification
  - Density estimation
- Which model is easier to train?
- Which model is faster (training & inference)?

Thank You!

16-726, Spring 2022

https://learning-image-synthesis.github.io/sp22/

© ATOM