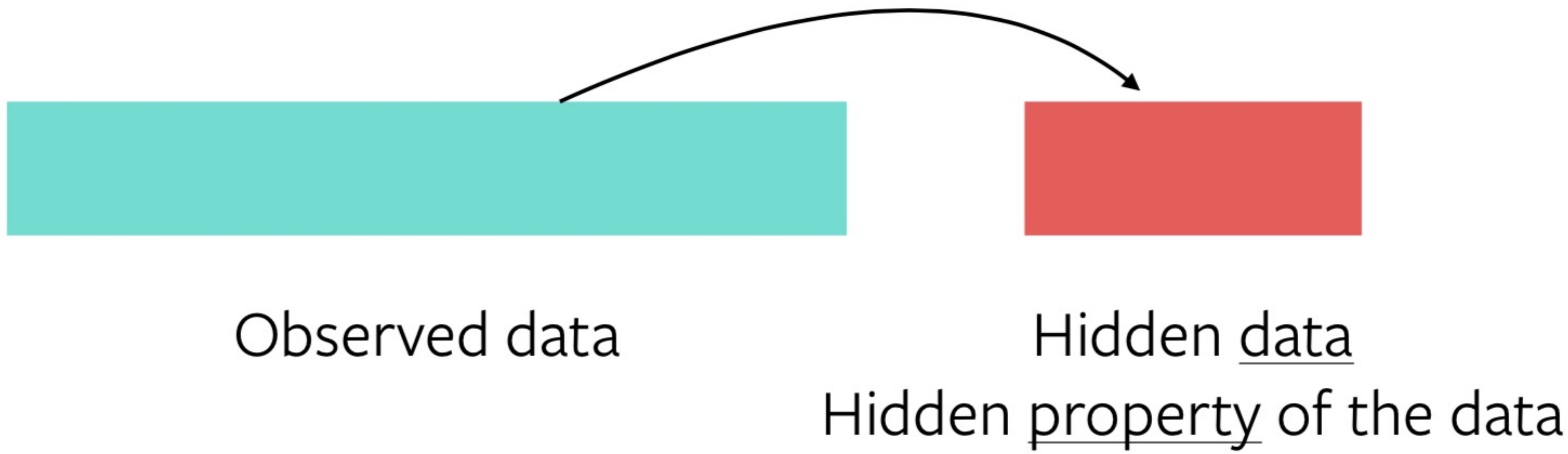
Overview of Unsupervised Learning
&
Generative Adversarial Networks

Lecture 8

Slides from: Emily Denton, Ian Goodfellow, Soumith Chintala
Karsten Kreis Ruiqi Gao Arash Vahdat

Modeling Data

- Recall “self-supervised” learning
- Predict some part of the data from another part

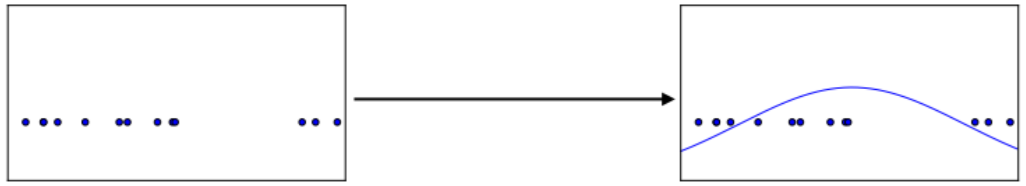


- Modeling p(x_hidden | x_observed)
- Use learned representations for down-steam task (e.g. classification)
- Focus today on **generating novel examples** from underlying distribution p(x)

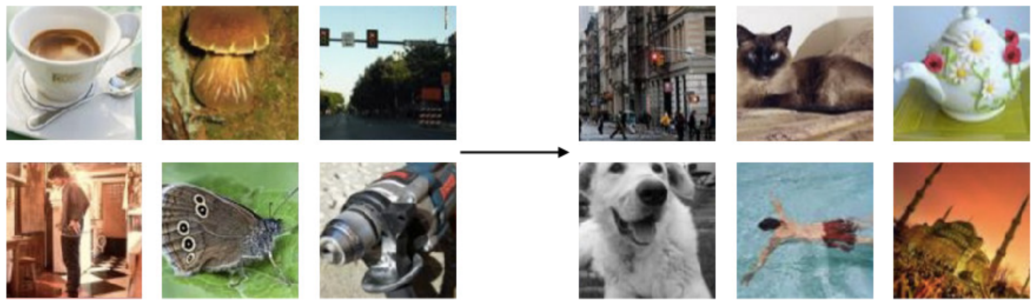
Density Modeling / Building Generative Model

- Have access to $x \sim p_{data}(x)$ through training set
- Want to learn a model $x \sim p_{model}(x)$
- Want p_{model} to be similar to p_{data}:

Samples from true data distribution have high likelihood under p_{model}



Samples drawn from p_{model} reflect structure of p_{data}



Training examples Model samples

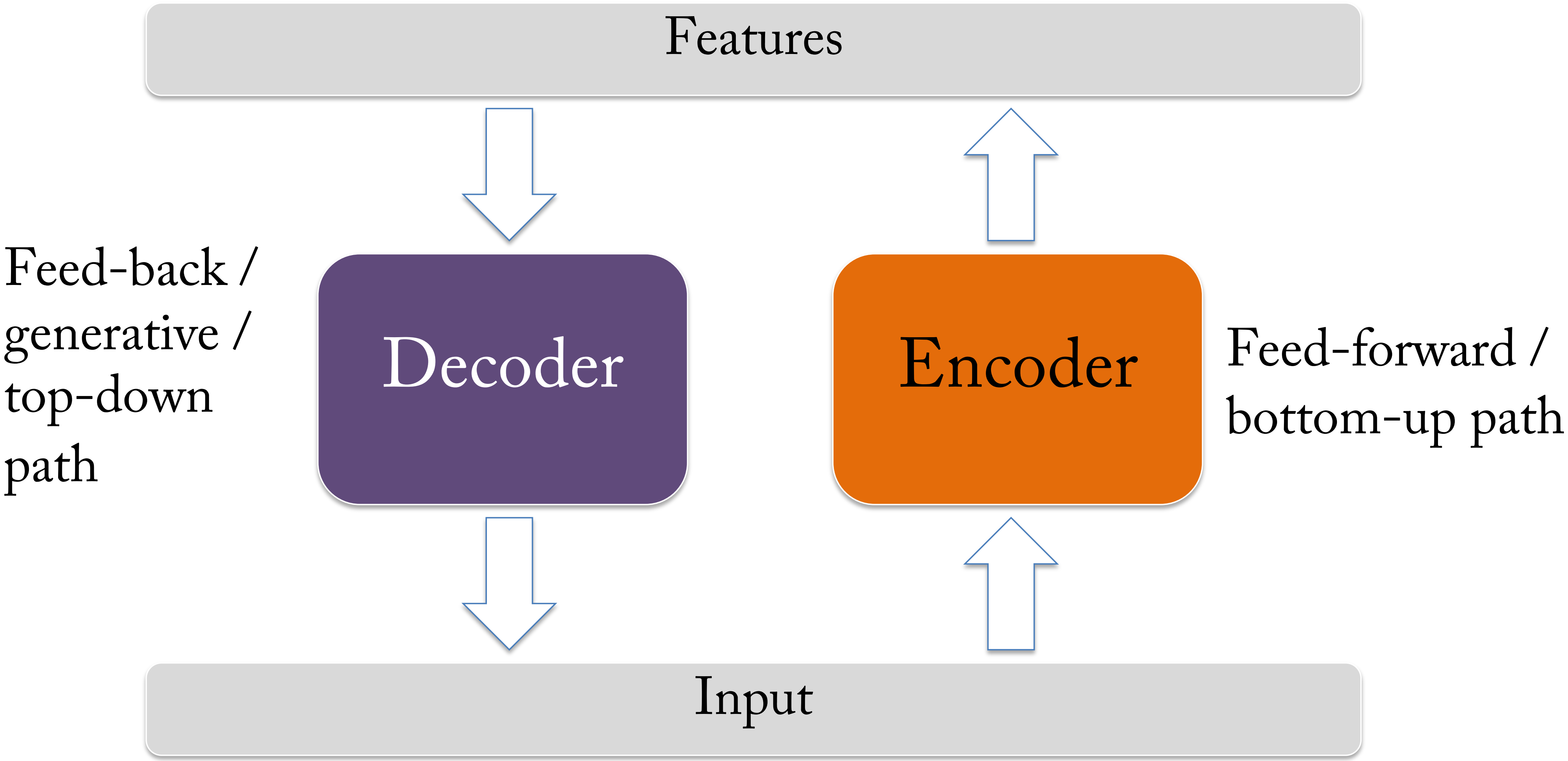
Generative Modeling approaches

- Autoencoders
- Auto-regressive models
- GANs
- Diffusion model

Generative Modeling approaches

- Autoencoders
- Auto-regressive models
- GANs
- Diffusion model

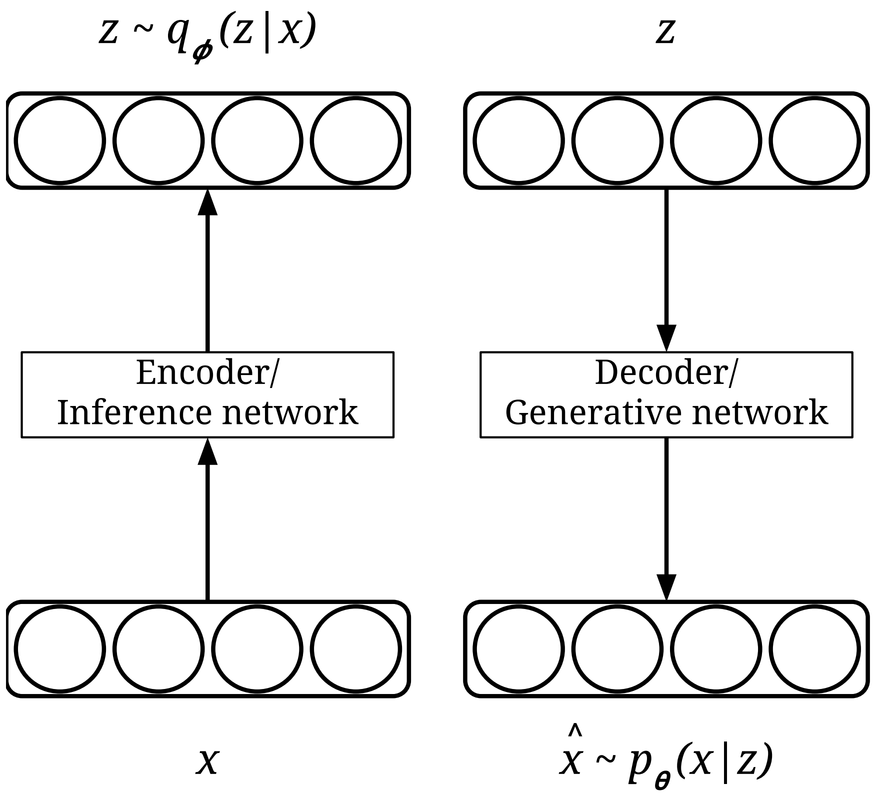
Auto-Encoder



- Encoder/Decoder will be deep network
- Slightly different architectures for decoder (needs to output image)
- Architecture depends on application

Variational autoencoder

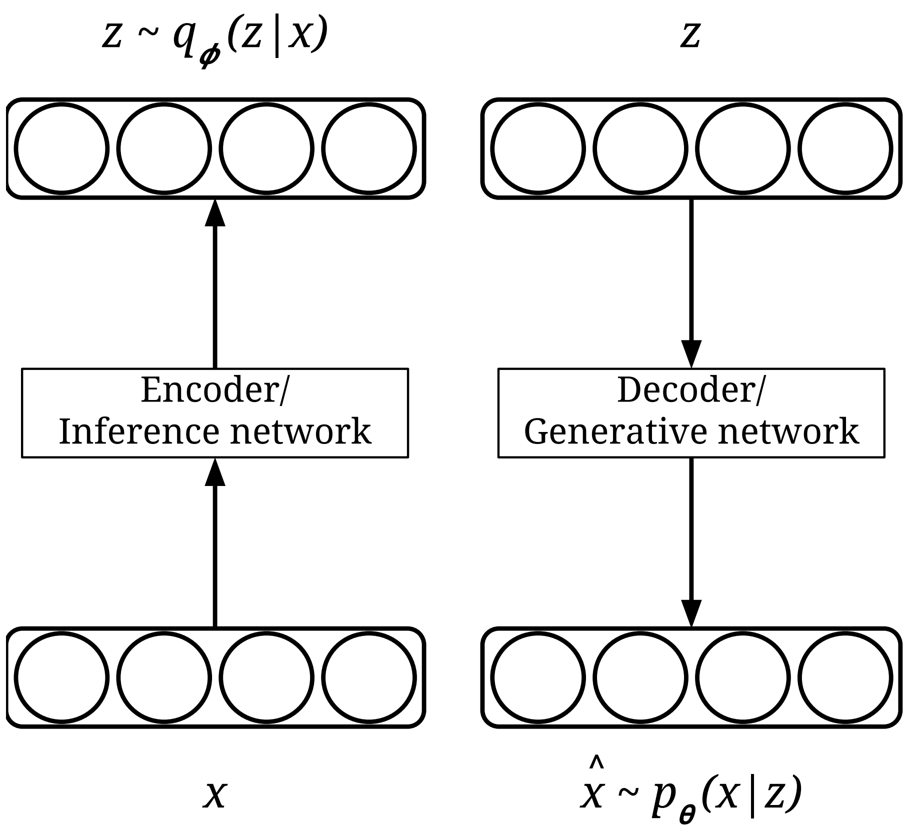
- *Encoder* network maps from image space to latent space
  - Outputs parameters of $q_\phi(z|x)$
- *Decoder* maps from latent space back into image space
  - Outputs parameters of $p_\theta(x|z)$



[Kingma & Welling (2013)]

Example

- *Encoder* network outputs mean and variance of Normal distribution
  - $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$
- *Decoder* network outputs mean (and optionally variance) of Normal distribution
  - $p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \mathbf{I})$



[Kingma & Welling (2013)]

Bounding the marginal likelihood

Recall Jenson's inequality: When f is concave, $f(\mathbb{E}[x]) \geq \mathbb{E}[f(x)]$

$$\begin{aligned}
\log p(x) &= \log \int_z p(x,z) \\
&= \log \int_z q(z) \frac{p(x,z)}{q(z)} \\
&\geq \int_z q(z) \log \frac{p(x,z)}{q(z)} = \underbrace{L(x;\theta,\phi)}_{\text{Evidence Lower BOund (ELBO)}} \qquad \text{(by Jensons inequality)}
\end{aligned}$$

Bound is tight when variational approximation matches true posterior:

$$\log p(x) - \underbrace{L(x;\theta,\phi)}_{\text{Evidence Lower BOund (ELBO)}} = \log p(x) - \int_z q(z) \log \frac{p(x,z)}{q(z)}$$

$$= \int_z q(z) \log p(x) - \int_z q(z) \log \frac{p(x,z)}{q(z)}$$

$$= \int_z q(z) \log \frac{q(z)p(x)}{p(x,z)}$$

$$= \int_z q(z) \log \frac{q(z)}{p(z|x)}$$

$$= D_{KL}(q(z;\phi)||p(z|x))$$

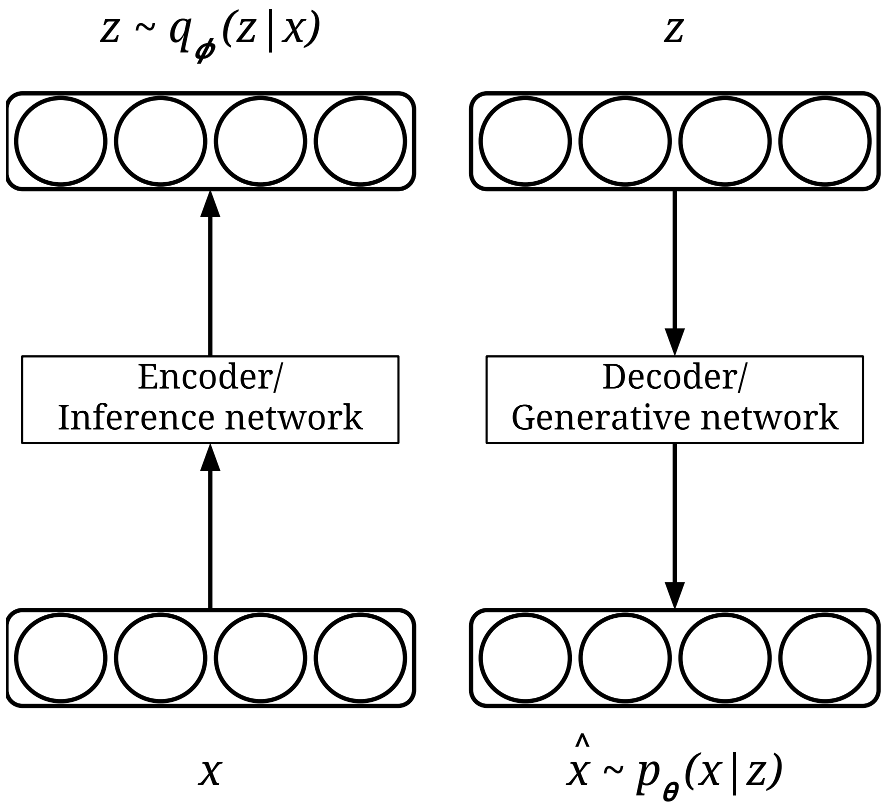
Variational autoencoder

- Rearranging the ELBO:

$$
\begin{aligned}
{\color{blue}L(x;\theta,\phi)} &= \int_z q(z|x)\log\frac{p(x,z)}{q(z|x)} \\
&= \int_z q(z|x)\log\frac{p(x|z)p(z)}{q(z|x)} \\
&= \int_z q(z|x)\log p(x|z) + \int_z q(z|x)\log\frac{p(z)}{q(z|x)} \\
&= \mathbb{E}_{q(z|x)}\log p(x|z) - \mathbb{E}_{q(z|x)}\log\frac{q(z|x)}{p(z)} \\
&= \underbrace{\mathbb{E}_{q(z|x)}\log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}
\end{aligned}
$$

Variational autoencoder

- Inference network outputs parameters of $q_\phi(z|x)$
- Generative network outputs parameters of $p_\theta(x|z)$
- Optimize θ and ϕ jointly by maximizing ELBO:



$$L(x;\theta,\phi) = \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}$$

Stochastic gradient variation bayes (SGVB) estimator

- Reparameterization trick : re-parameterize $z \sim q_\phi(z|x)$ as

$$z = g_\phi(x, \epsilon) \text{ with } \epsilon \sim p(\epsilon)$$

- For example, with a Gaussian can write $z \sim \mathcal{N}(\mu, \sigma^2)$ as

$$z = \mu + \epsilon\sigma^2 \text{ with } \epsilon \sim \mathcal{N}(0, 1)$$

[Kingma & Welling (2013); Rezende *et al.* (2014)]

Stochastic gradient variation bayes (SGVB) estimator

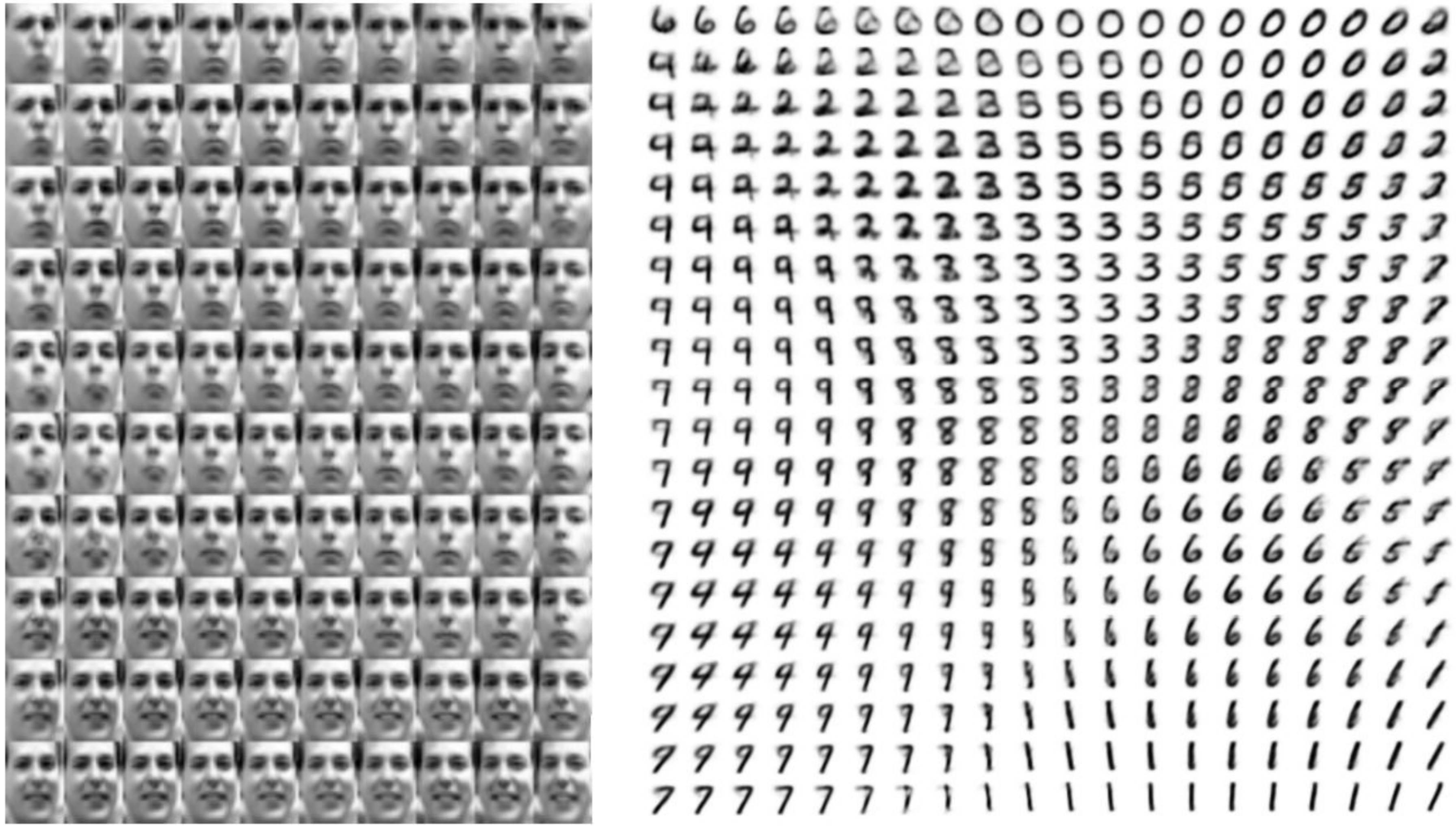
$$\color{blue}{L(x;\theta,\phi)} = \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}$$

- Using reparameterization trick we form Monte Carlo estimate of reconstruction term:

$$\begin{aligned} \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) &= \mathbb{E}_{p(\epsilon)} \log p_\theta(x|g_\phi(x,\epsilon)) \\ &\simeq \frac{1}{L}\sum_{i=1}^{L} \log p_\theta(x|g_\phi(x,\epsilon)) \qquad \text{where } \epsilon \sim p(\epsilon) \end{aligned}$$

- KL divergence term can often be computed analytically (eg. Gaussian)

VAE learned manifold



[Kingma & Welling (2013)]

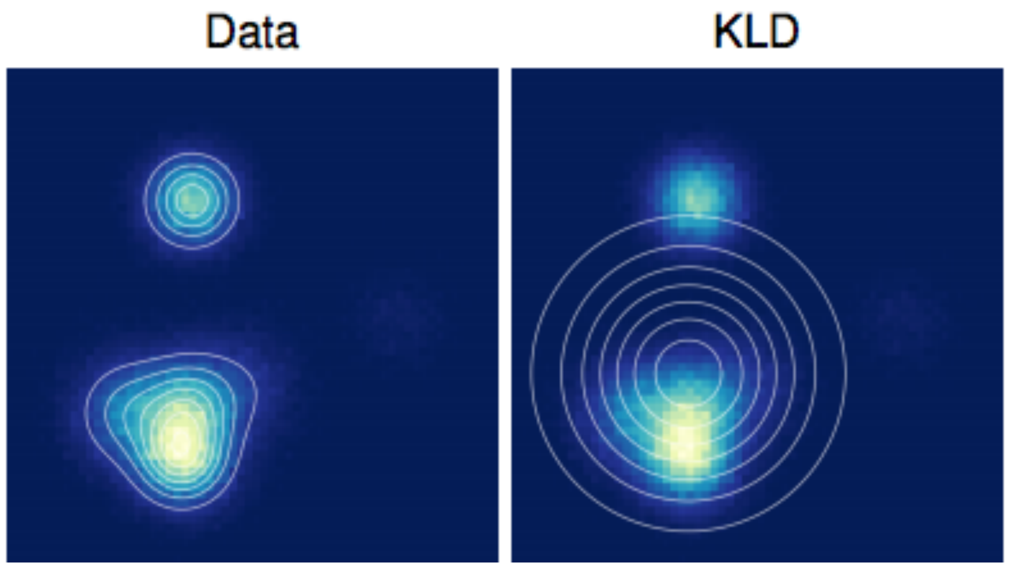
VAE samples

(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

[Kingma & Welling (2013)]

VAE tradeoffs

- Pros:
  - Theoretically pleasing
  - Optimizes bound on likelihood
  - Easy to implement
- Cons:
  - Samples tend to be blurry
    - Maximum likelihood minimizes $D_{KL}(p_{data}||p_{model})$



[Theis *et al.* (2016)]

Many Other Approaches

- VAE Variants
  - VQ-VAE
  - Beta-VAE
  - Etc.
- Other variants of Autoencoder
  - Restricted / Deep Boltzmann Machines
  - Denoising autoencoders
  - Predictive sparse decomposition
- Decoder-only
  - Sparse coding & hierarchical variants

VQ-VAE

VQ = Vector quantized

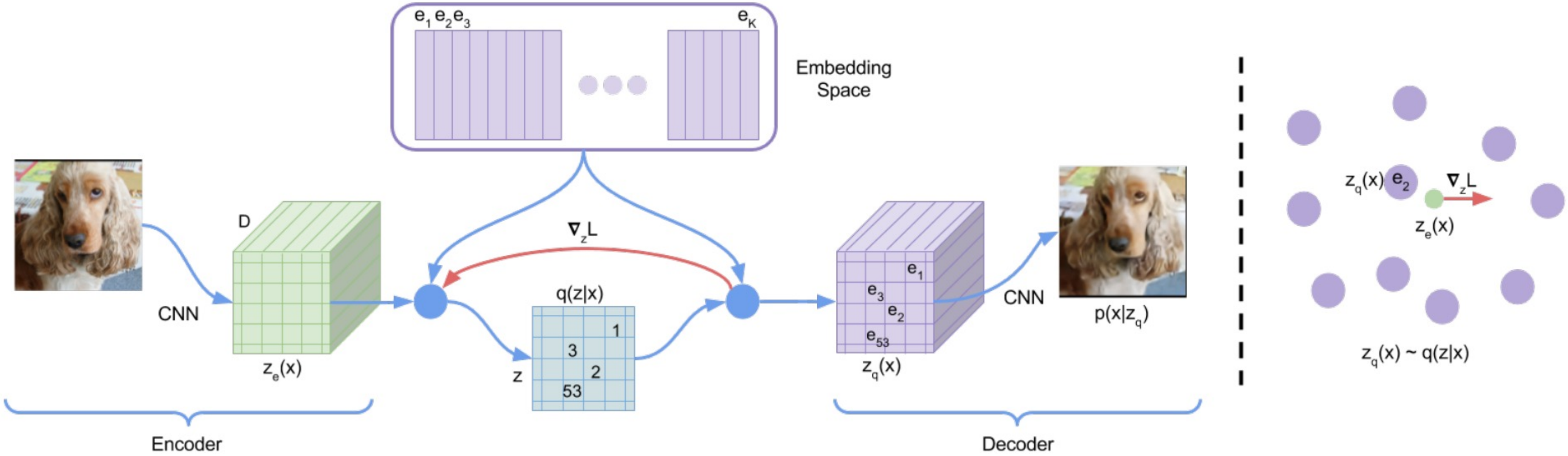
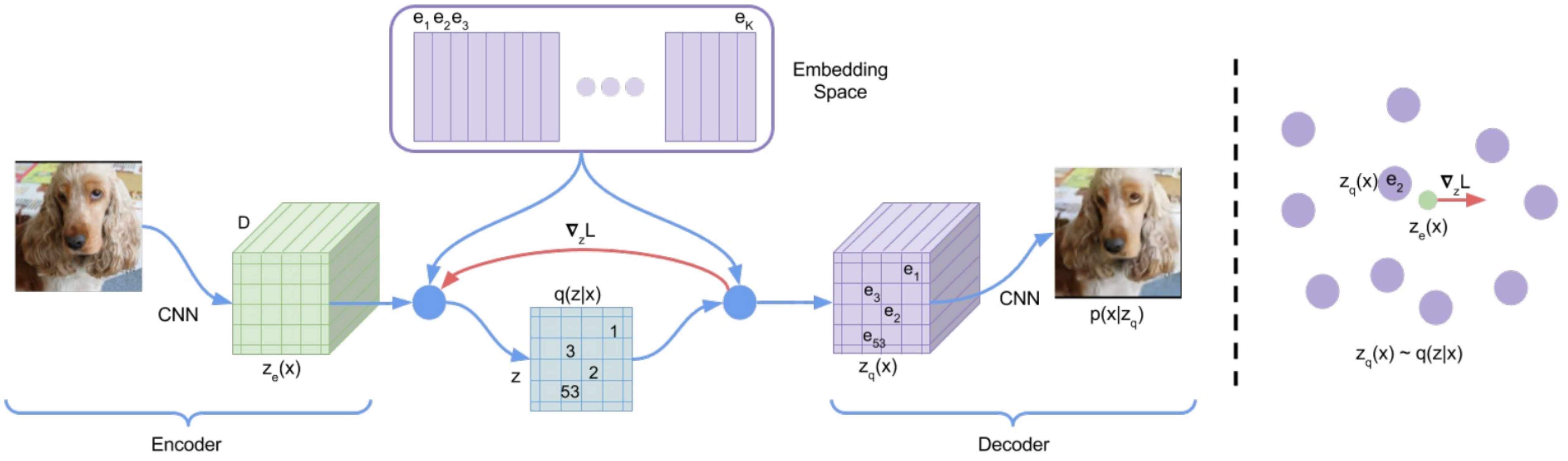


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

VQ-VAE: Neural discrete representation learning. Van Den Oord, Aaron. Vinyals, Oriol. Kavukcuoglu, Koray. NeurIPS 2017

VQ-VAE



$$L = \underbrace{\log p(x|z_q(x))}_{\text{Image reconstruction loss}} + \underbrace{\|\mathrm{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \mathrm{sg}[e]\|_2^2}_{\text{Make the quantized vector as close as the original vector}},$$

sg : the stop gradient operator that

e : quantized vector

$z_e(x)$: vector from the encoder output

VQ-VAE: Neural discrete representation learning. Van Den Oord, Aaron. Vinyals, Oriol. Kavukcuoglu, Koray. NeurIPS 2017

VQ-VAE Reconstructions

Figure 2: Left: ImageNet 128x128x3 images, right: reconstructions from a VQ-VAE with a 32x32x1 latent space, with K=512.

VQ-VAE: Neural discrete representation learning. Van Den Oord, Aaron. Vinyals, Oriol. Kavukcuoglu, Koray. NeurIPS 2017

VQ-VAE Samples

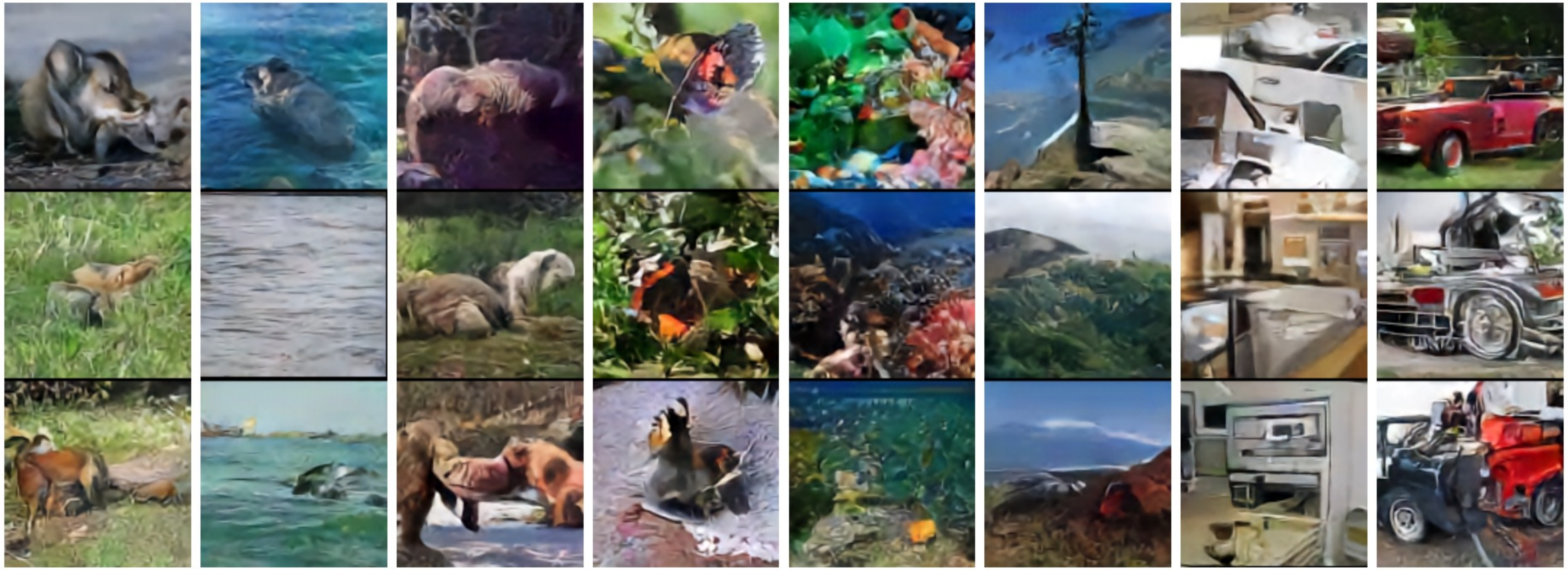


Figure 3: Samples (128x128) from a VQ-VAE with a PixelCNN prior trained on ImageNet images. From left to right: kit fox, gray whale, brown bear, admiral (butterfly), coral reef, alp, microwave, pickup.

VQ-VAE: Neural discrete representation learning. Van Den Oord, Aaron. Vinyals, Oriol. Kavukcuoglu, Koray. NeurIPS 2017

Many Other Approaches

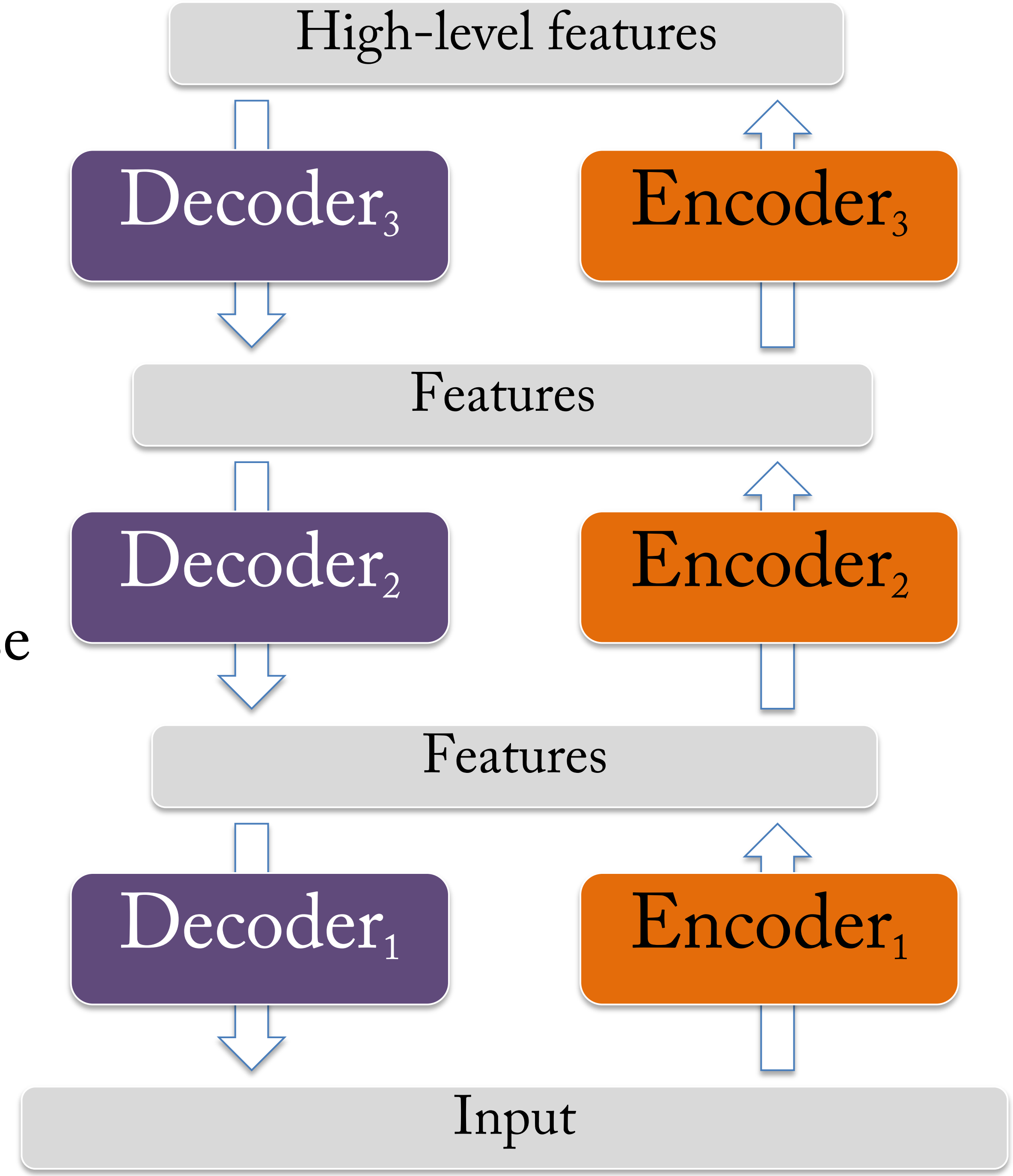
- VAE Variants
  - VQ-VAE
  - Beta-VAE
  - Etc.
- Other variants of Autoencoder
  - Restricted / Deep Boltzmann Machines
  - Denoising autoencoders
  - Predictive sparse decomposition
- Decoder-only
  - Sparse coding & hierarchical variants

Stacked Auto-Encoders

- Ladder Networks
  [Rasmus et al. 2015]
  - Reconstruction constraint at each layer
  - Trained end-to-end

- Can be trained layer-wise
  - Stacked RBMs

  [Hinton & Salakhutdinov 2006]



Generative Modeling approaches

- Autoencoders
- Auto-regressive models
- GANs
- Diffusion model

Auto-regressive language models

- Sequence of words: $w_1, \ldots, w_m$

- n-gram models $P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=2}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$

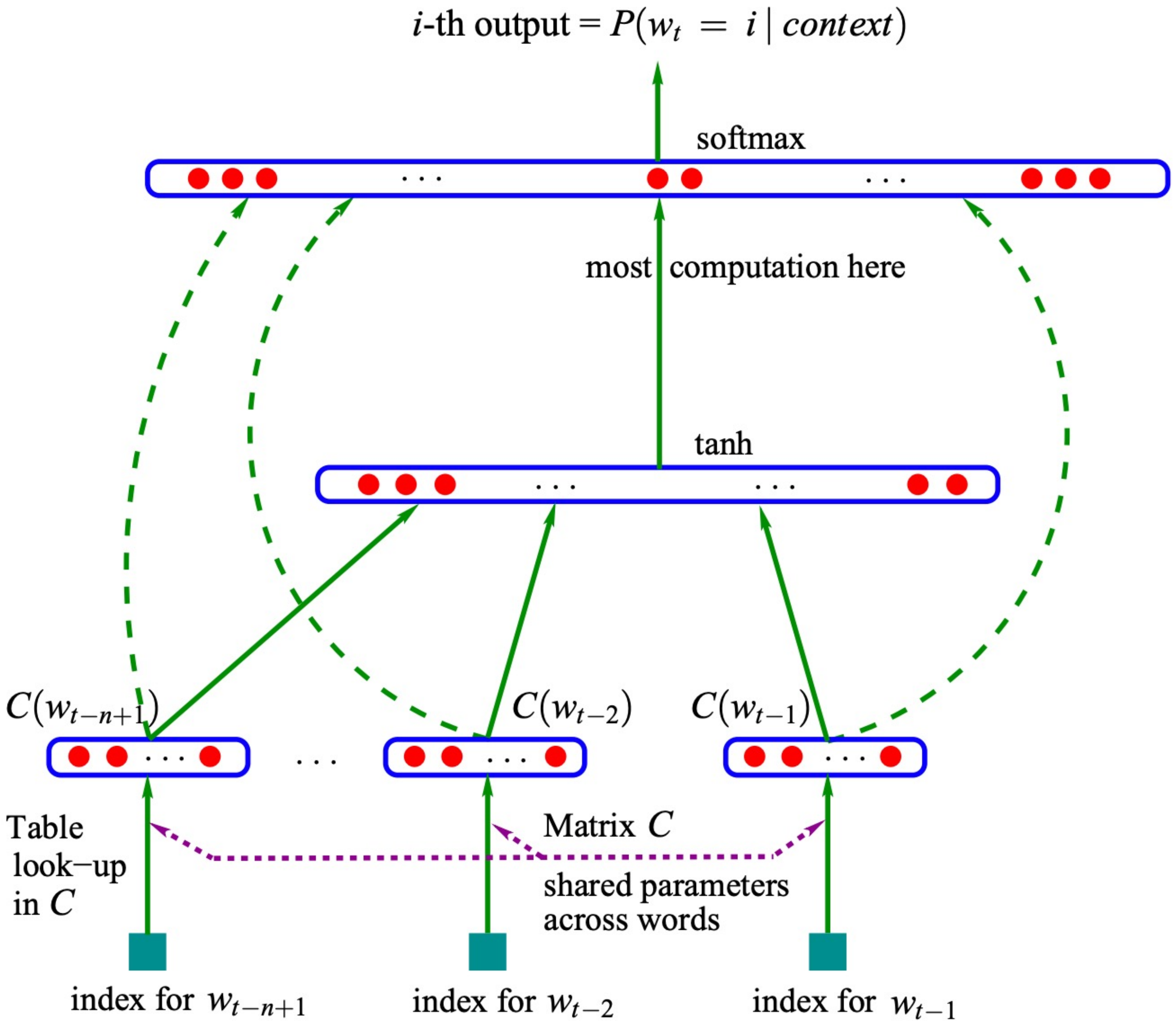
- Bigram (n=2)

$$P(\text{I, saw, the, red, house}) \approx P(\text{I} \mid \langle s \rangle) P(\text{saw} \mid \text{I}) P(\text{the} \mid \text{saw}) P(\text{red} \mid \text{the}) P(\text{house} \mid \text{red}) P(\langle /s \rangle \mid \text{house})$$

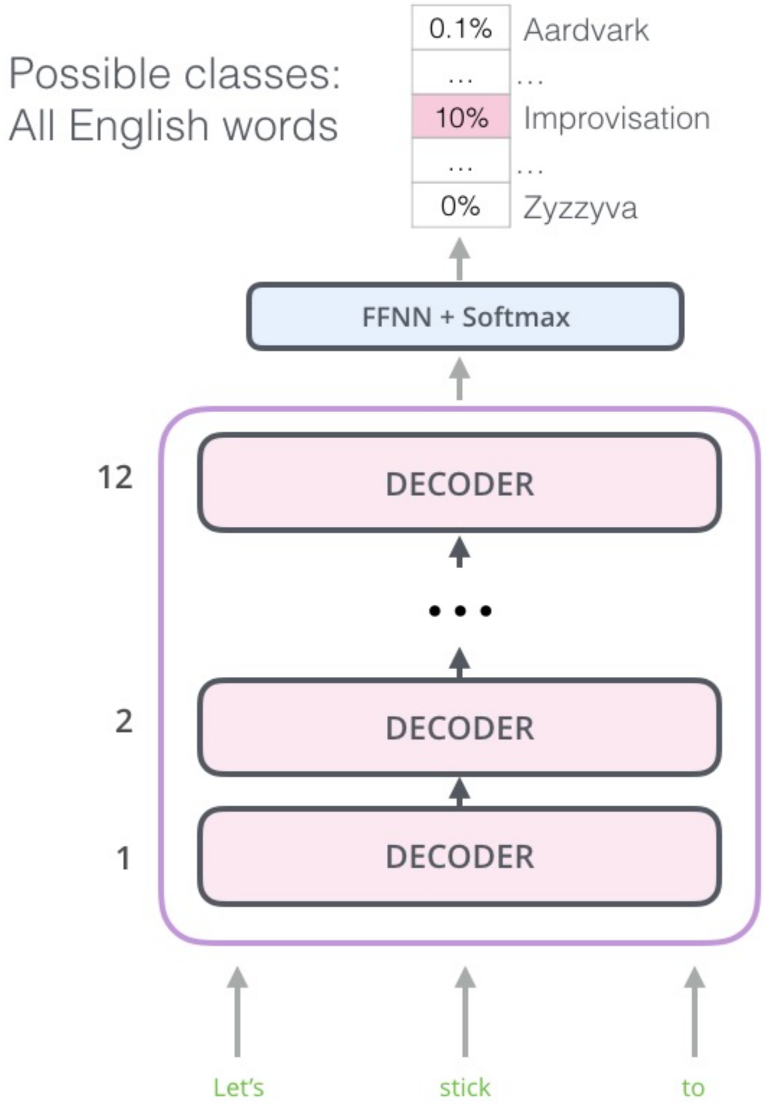
- Trigram (n=3):

$$P(\text{I, saw, the, red, house}) \approx P(\text{I} \mid \langle s \rangle, \langle s \rangle) P(\text{saw} \mid \langle s \rangle, I) P(\text{the} \mid \text{I, saw}) P(\text{red} \mid \text{saw, the}) P(\text{house} \mid \text{the, red}) P(\langle /s \rangle \mid \text{red, house})$$

Auto-regressive Neural language models



[Bengio et al. 2003, JMLR]



GPT-3
= 170B params

[GPT-2, GPT-3, OpenAI]

Autoregressive models

- Tractably model a joint distribution of the pixels in the image
- Learn to predict the next pixel given all the previously generated pixels
- Joint distribution of all pixels just product of conditionals:

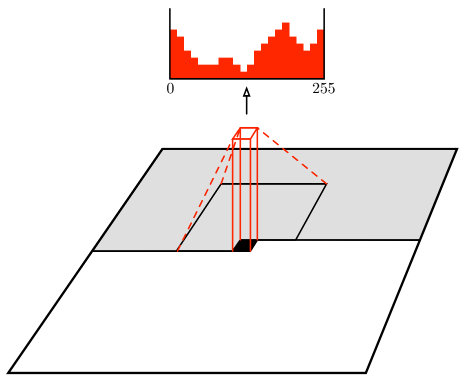
$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

Pixel-CNN

[van den Oord et al., arXiv 1606.05328, 2016]

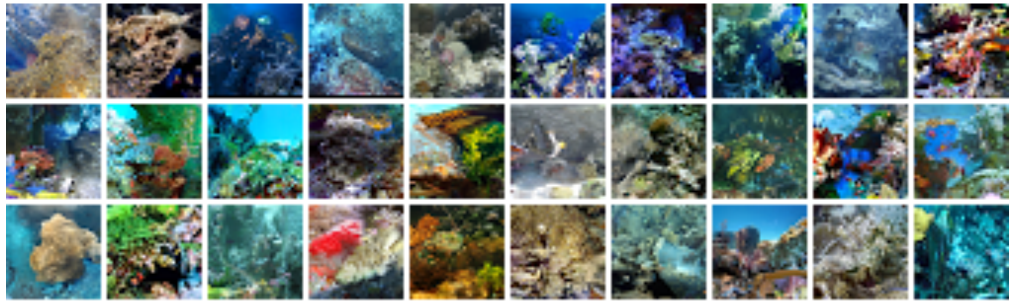
- Conditional generative model of images
- Generate each pixel, in raster-scan order
- Just predict distribution over a single pixel (can be multi-modal)
- See also Video Pixel Networks [Kalchbrenner et al., 2016],
- NADE [Larochelle & Murray 2011] & RIDE [Theis and Bethge, NIPS 2015].

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1}).$$



1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

African elephant



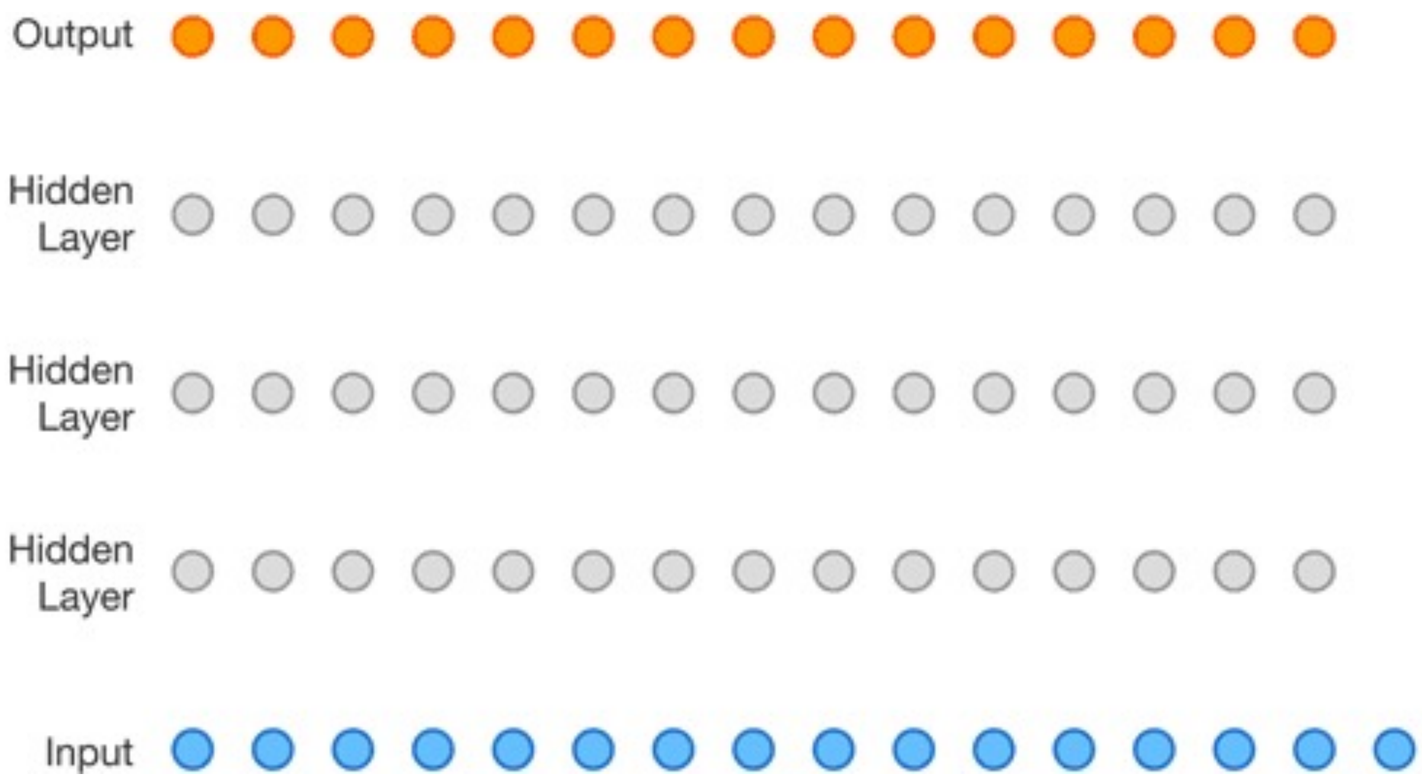
Coral Reef

Wavenet

[van den Oord et al., arXiv 1609.03499, 2016]

- Generative model of raw speech waveform
- Condition on previous parts of waveform
- Dilated causal convolution layers
- Discrete output distribution (use softmax)

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



DALL-E 1 model

https://openai.com/blog/dall-e/

- 12B parameter Transformer - version of GPT-3 trained to generate images from text descriptions.
- 1. Pre-train VAE model to represent 8x8 natural image patches as discrete code (K=8192).
  - So 256x256 image → 32x32 grid of discrete codes.
- 2. Concatenate 1024 image codes with text embedding of description. Pass to auto-regressive Transformer. Model joint distribution of text, images.
- Train on Conceptual Captions (3.3M text/image pairs) + JFT-300M.
- Sample by conditioning on user-specified caption.
- Reranking procedure using pre-trained model (CLIP).

Ramesh, Pavlov, Goh, Gray, Voss, Radford, Chen, Sutskever Zero-Shot Text-to-Image Generation, arXiv 2102.12092v2, 2021.

DALL-E 1 model

https://openai.com/blog/dall-e/

TEXT PROMPT an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES

Edit prompt or view more images↓

TEXT PROMPT an armchair in the shape of an avocado. . . .

AI-GENERATED IMAGES

Edit prompt or view more images↓

DALL-E 1 model [OpenAI]

https://openai.com/blog/dall-e/

TEXT PROMPT

a store front that has the word ‘openai’ written on it. . . .

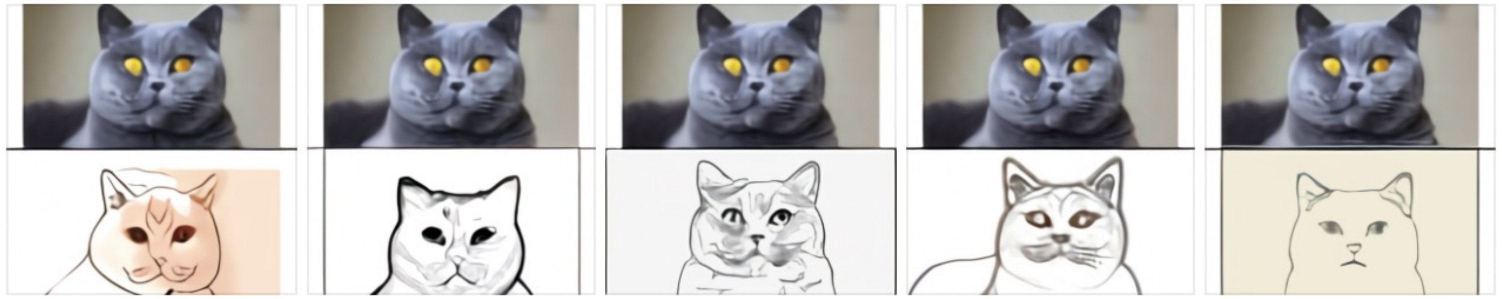
AI-GENERATED IMAGES

Edit prompt or view more images↓

TEXT & IMAGE PROMPT

the exact same cat on the top as a sketch on the bottom

AI-GENERATED IMAGES



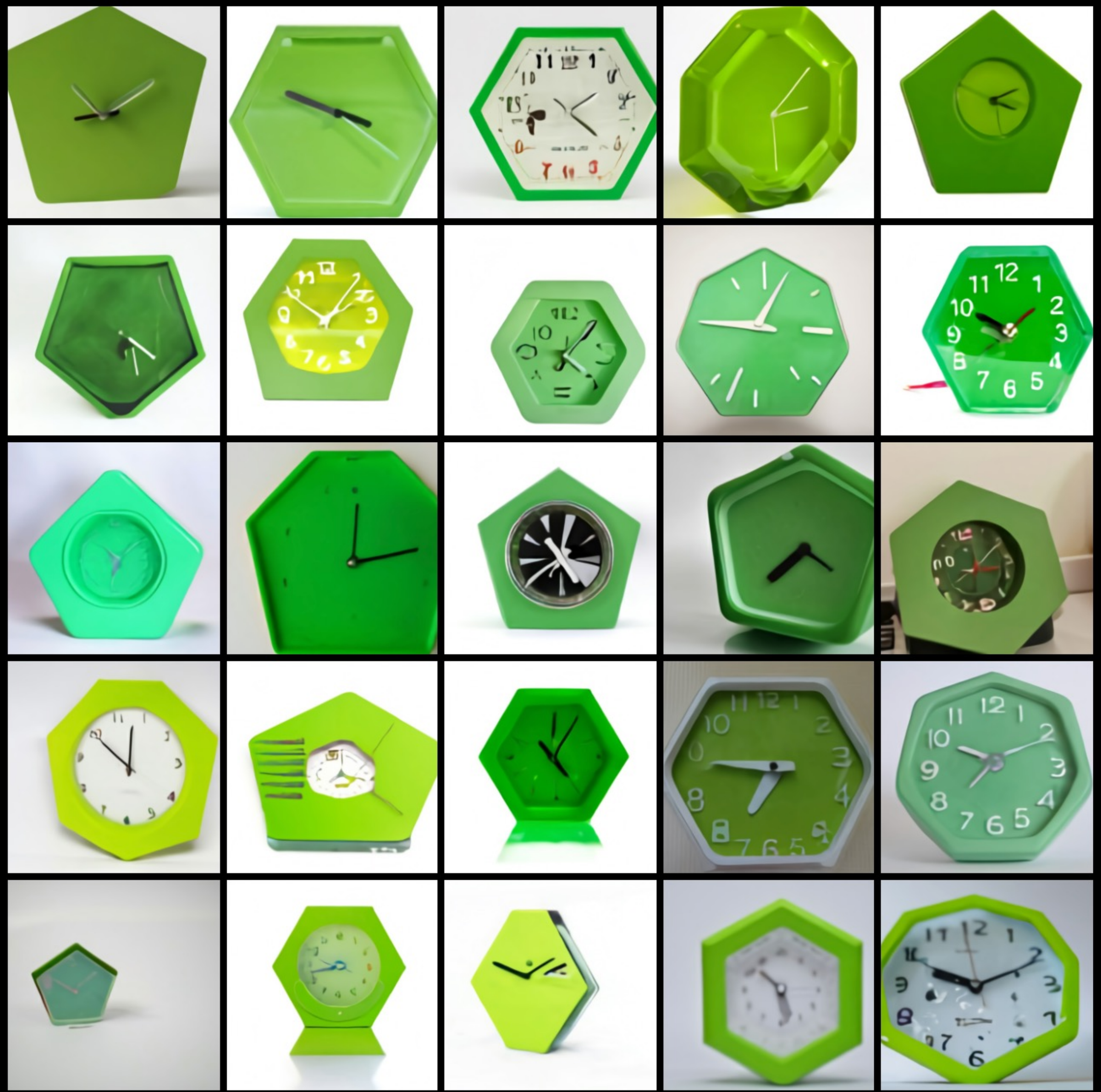
Edit prompt or view more images↓

https://openai.com/blog/dall-e/

DALL-E 1 model

TEXT PROMPT a pentagonal green clock. a green clock in the shape of a pentagon.

AI-GENERATED IMAGES



We find that DALL·E can render familiar objects in polygonal shapes that are sometimes unlikely to occur in the real world. For some objects, such as "picture frame" and "plate," DALL·E can reliably draw the object in any of the polygonal shapes except heptagon. For other objects, such as "manhole cover" and "stop sign," DALL·E's success rate for more unusual shapes, such as "pentagon," is considerably lower.

For several of the visuals in this post, we find that repeating the caption, sometimes with alternative phrasings, improves the consistency of the results.

Generative Modeling approaches

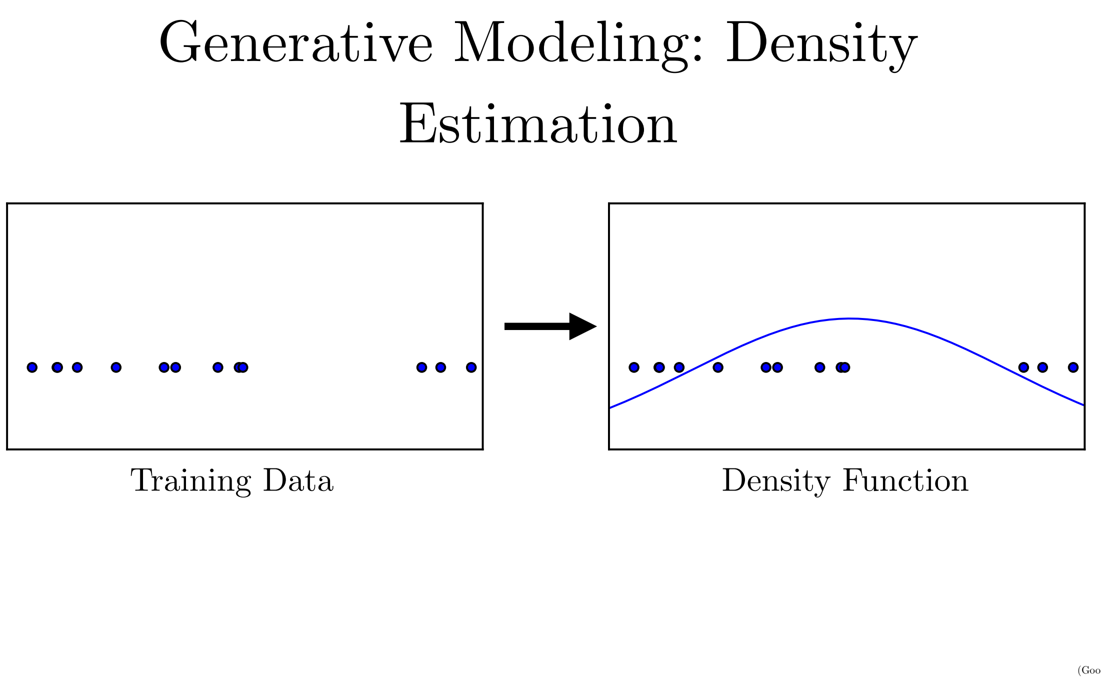
- Autoencoders
- Auto-regressive models
- GANs
- Diffusion model

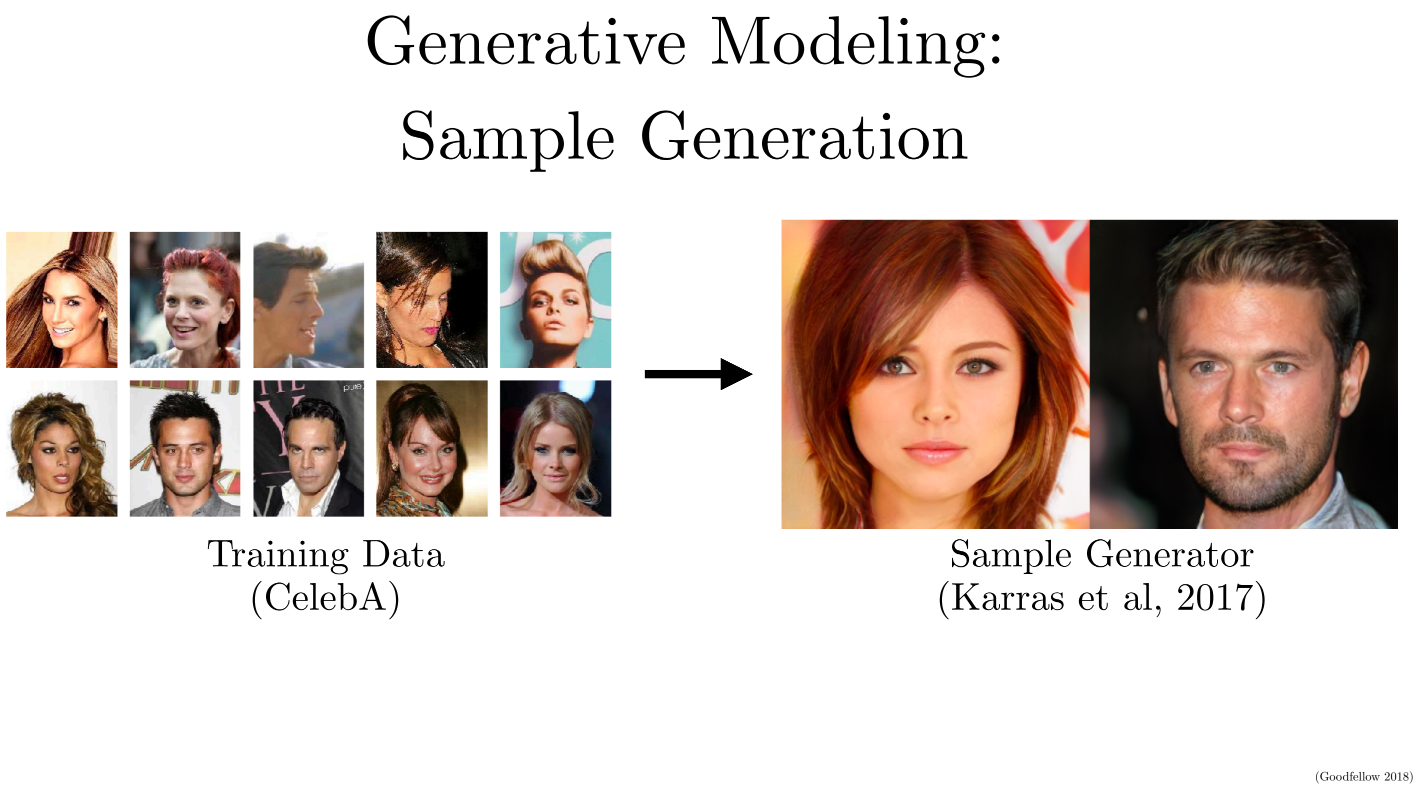
Generative Adversarial Networks

Slides from: Emily Denton, Ian Goodfellow, Soumith Chintala

Generative Adversarial Networks

- [Generative Adversarial Nets, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, NIPS 2014]
- Focus on sample generation

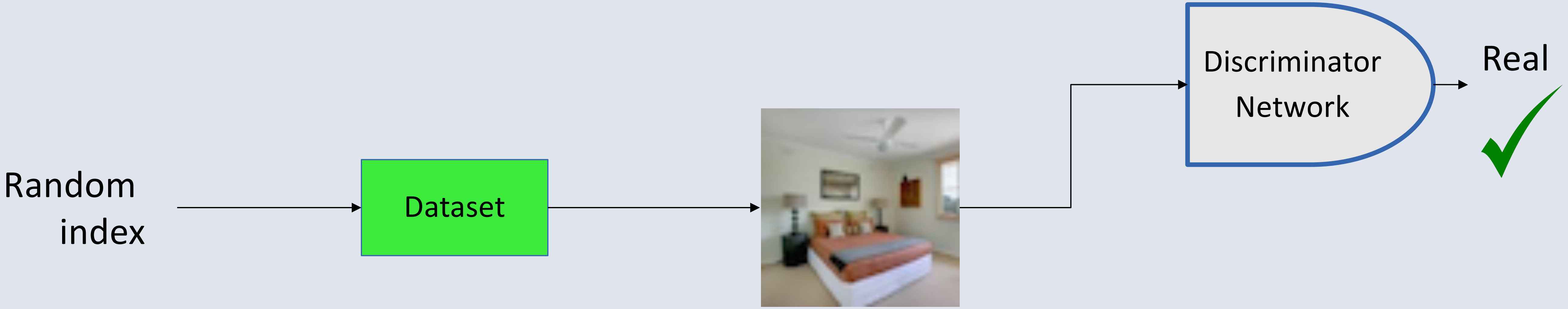




Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

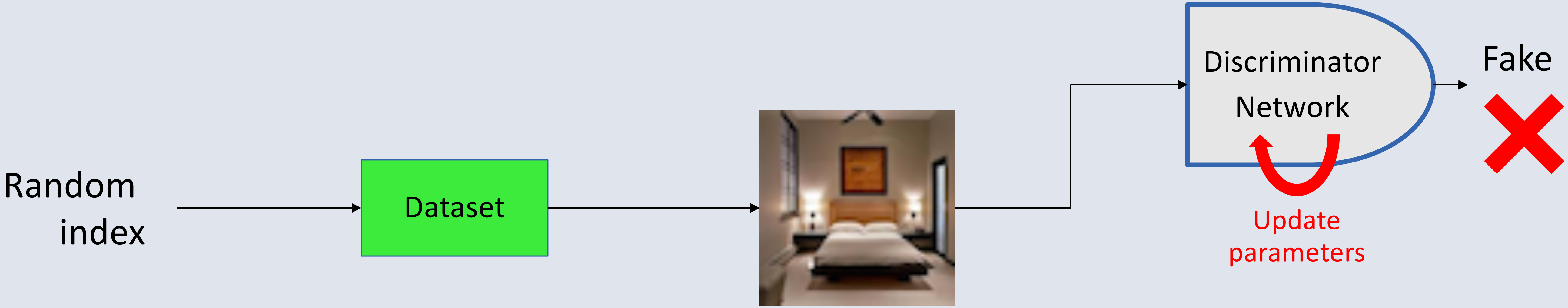
- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)



Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)



Generative Adversarial Network

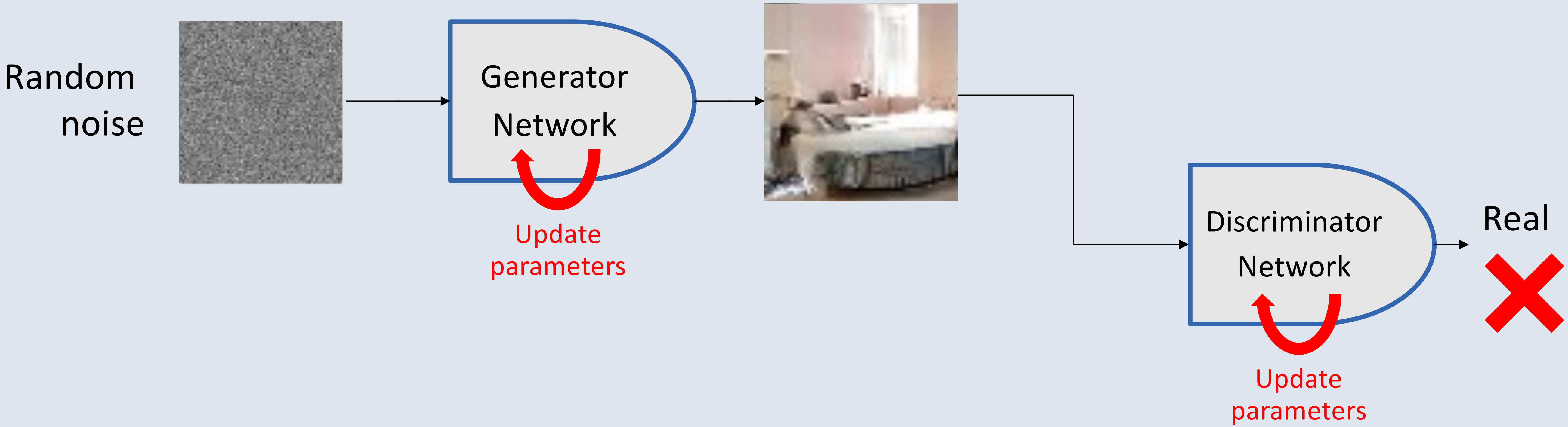
[Goodfellow et al. NIPS 2014]

- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)
- Generator network tries to confuse Discriminator

Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

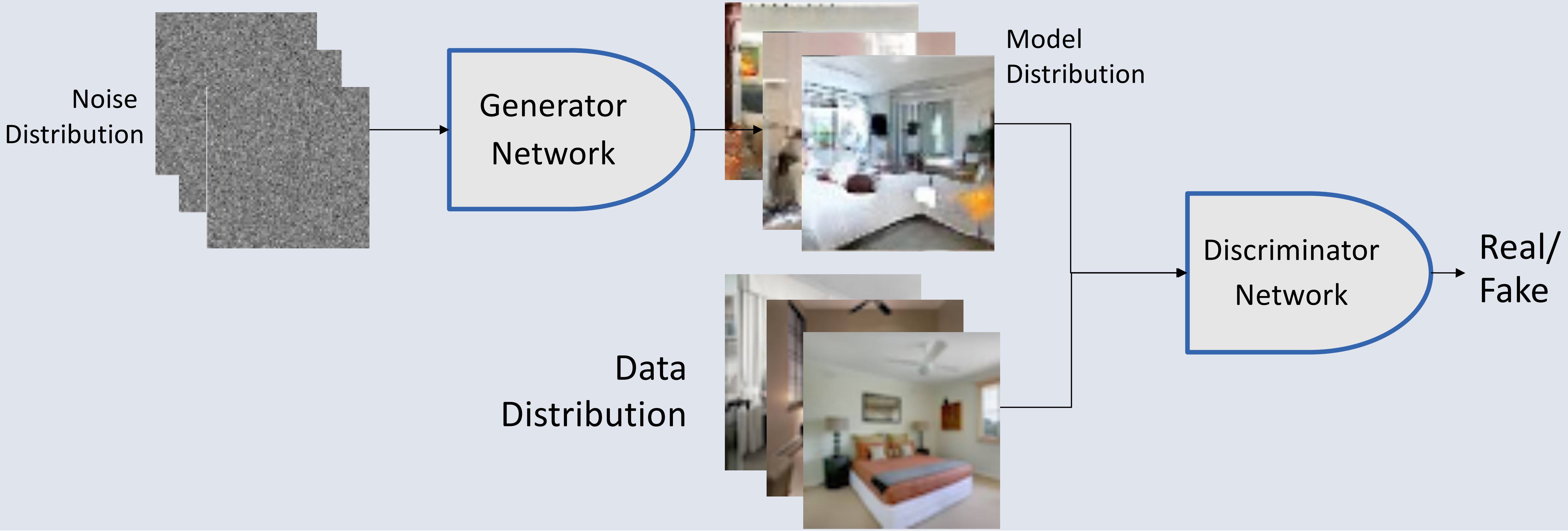
- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)
- Generator network tries to confuse discriminator



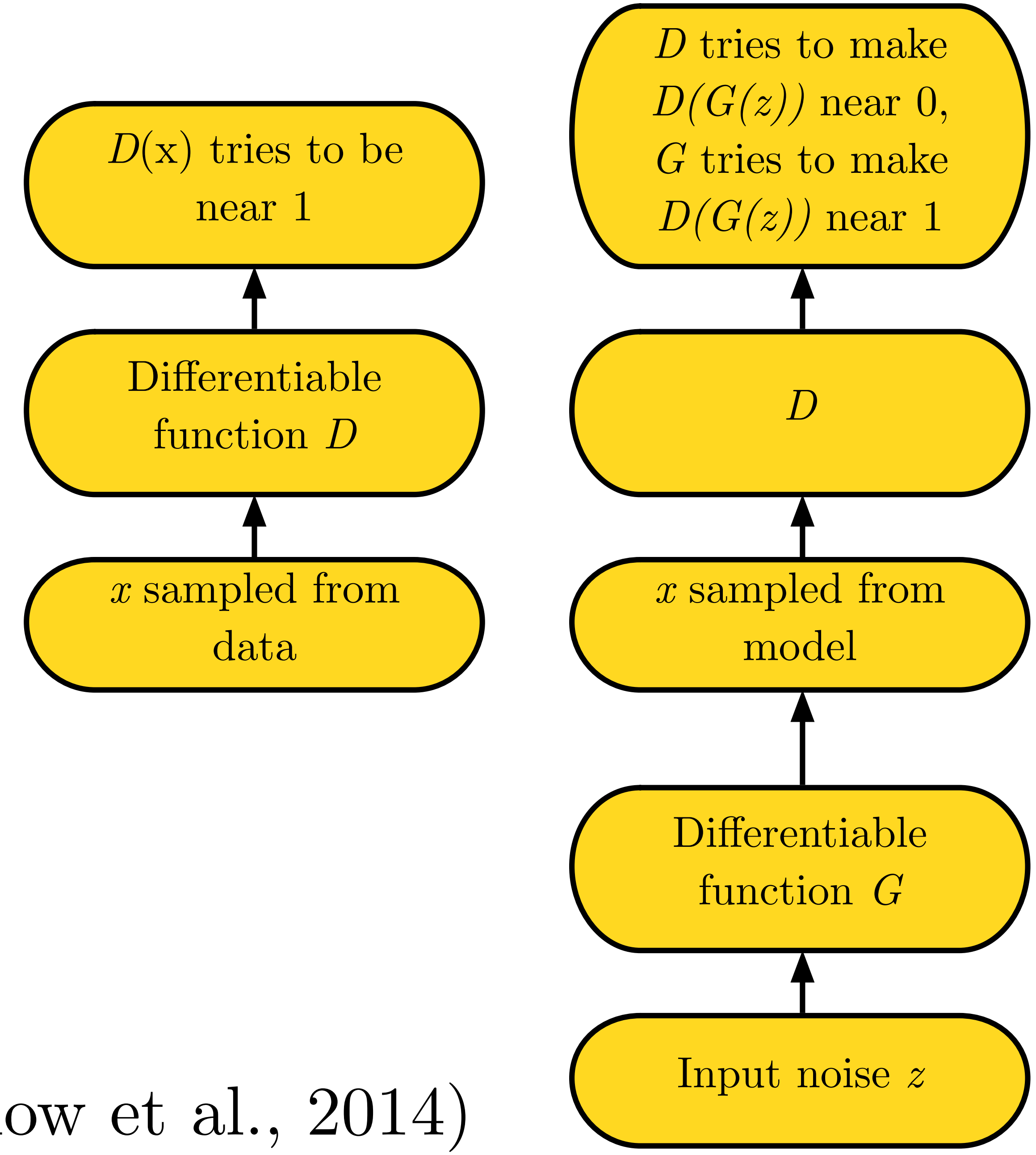
Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)
- Generator network tries to confuse Discriminator



Generative Adversarial Networks



(Goodfellow et al., 2014)

Generative Adversarial Networks

[Generative Adversarial Nets, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, NIPS 2014]

- Minimax value function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Generator pushes down

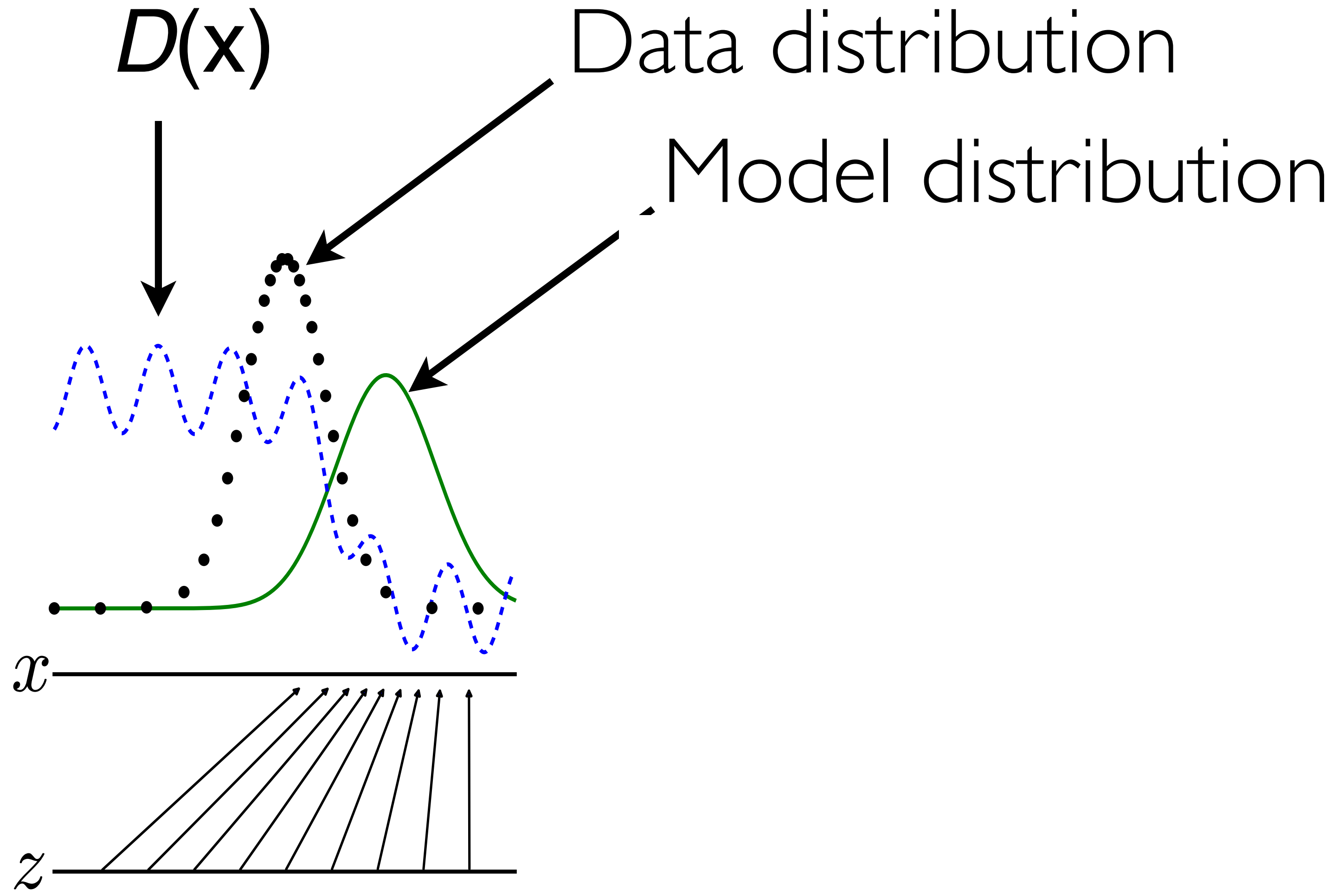
Discriminator pushes up

Discriminator's ability to recognize data as being real

Discriminator's ability to recognize generator samples as being fake

[Slide: Ian Goodfellow, Deep Learning workshop, ICML 2015]

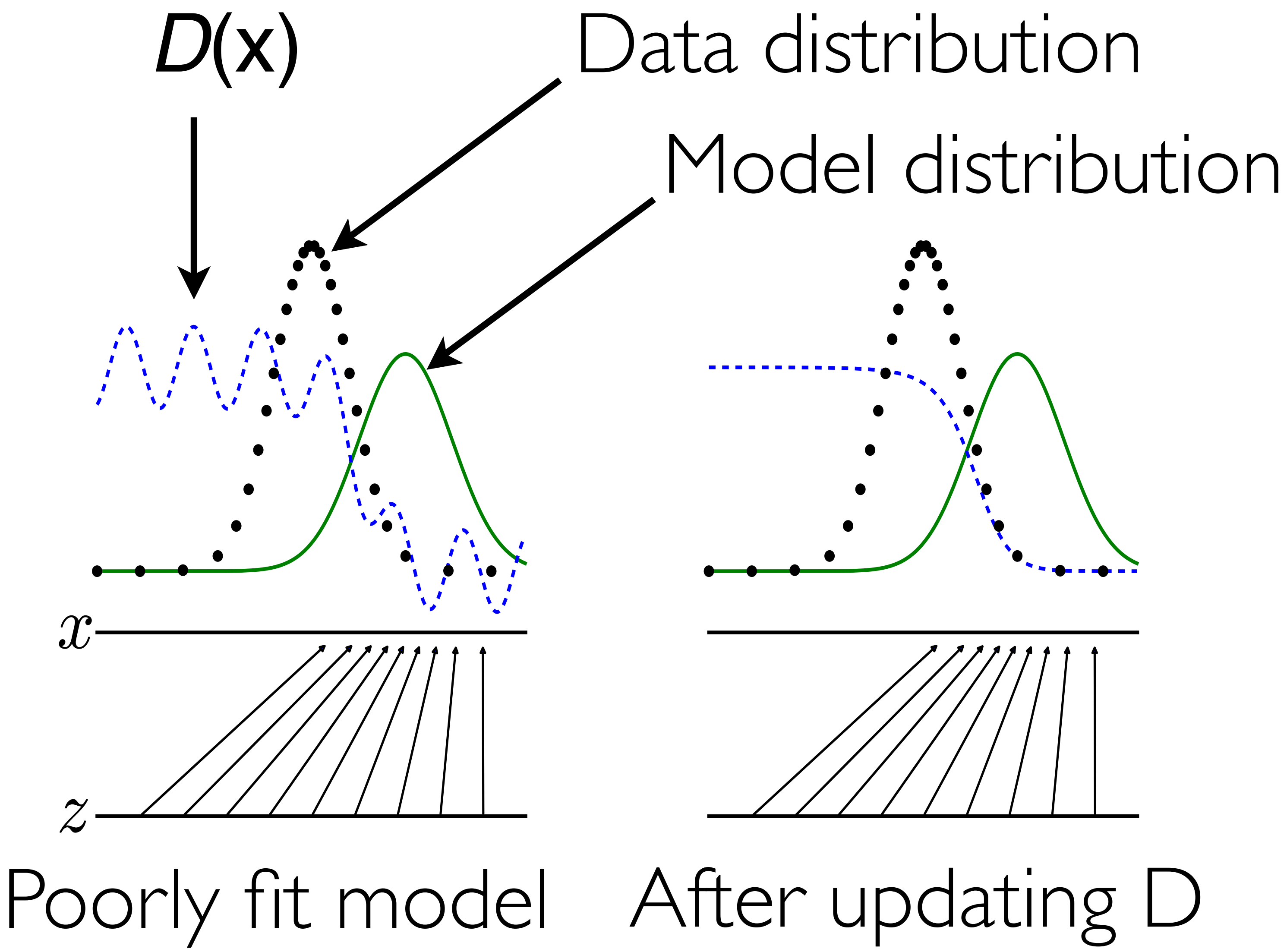
Generative Adversarial Networks



Poorly fit model

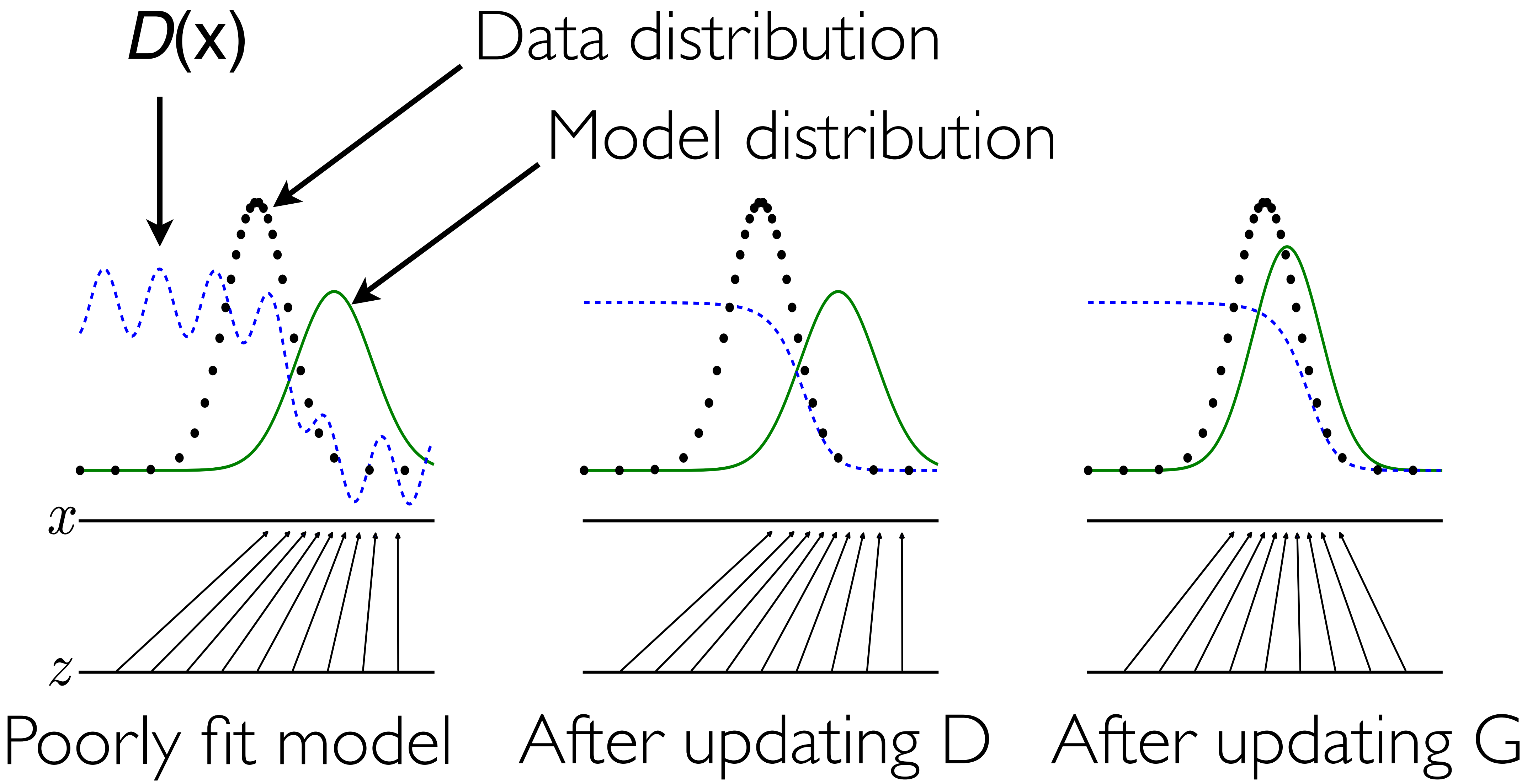
[Slide: Ian Goodfellow, Deep Learning workshop, ICML 2015]

Generative Adversarial Networks



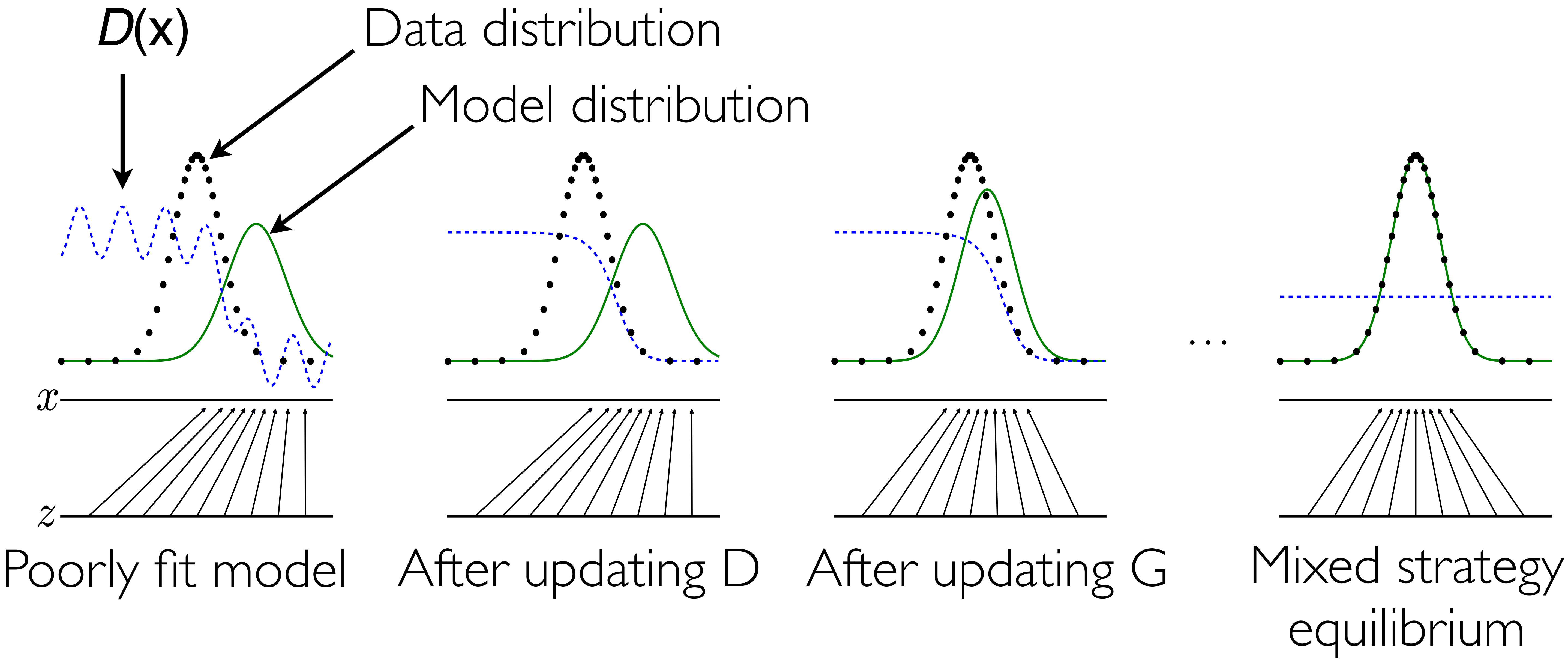
[Slide: Ian Goodfellow, Deep Learning workshop, ICML 2015]

Generative Adversarial Networks



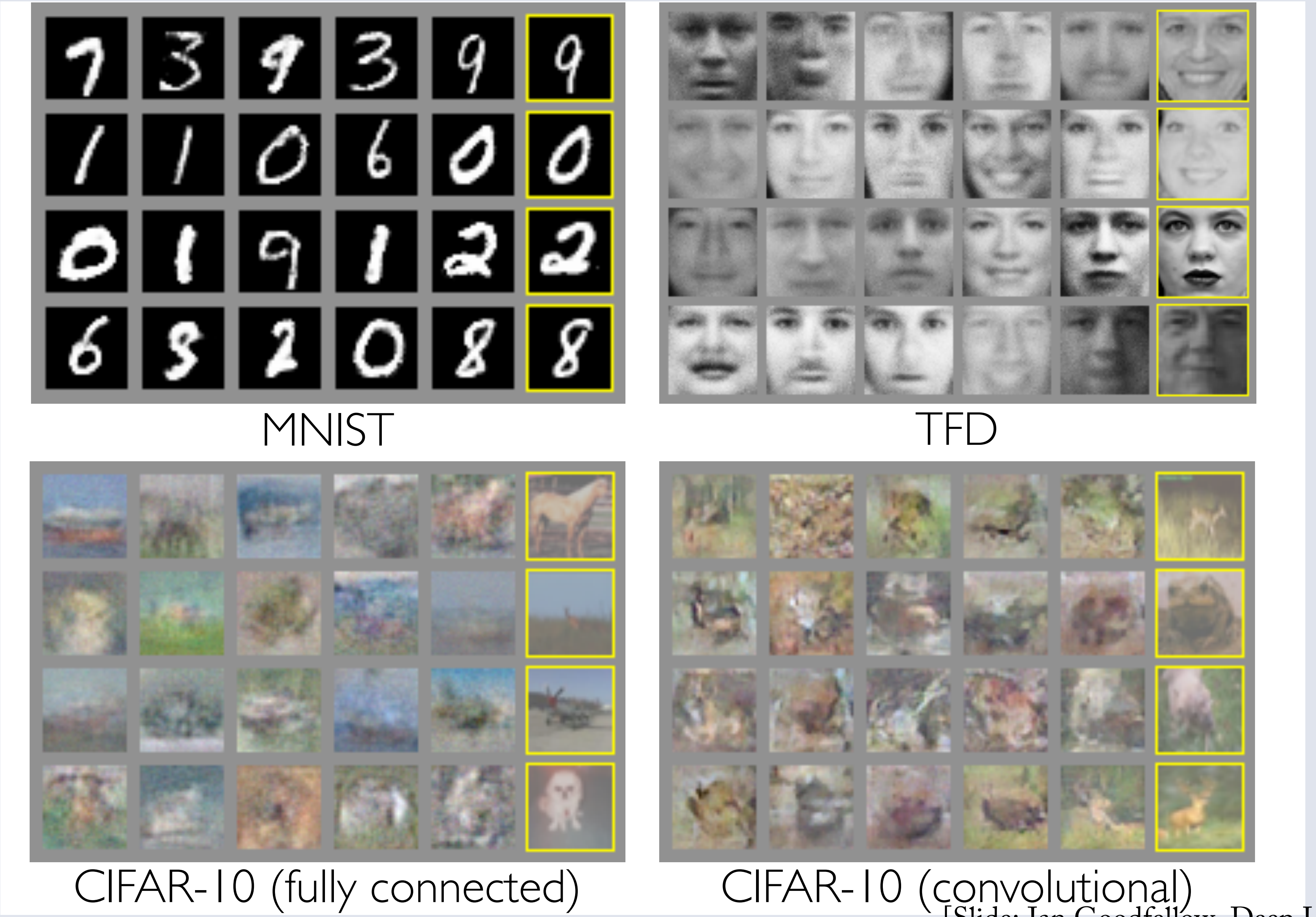
[Slide: Ian Goodfellow, Deep Learning workshop, ICML 2015]

Generative Adversarial Networks



[Slide: Ian Goodfellow, Deep Learning workshop, ICML 2015]

Adversarial Network Samples



[Slide: Ian Goodfellow, Deep Learning workshop, ICML 2015]

DCGAN

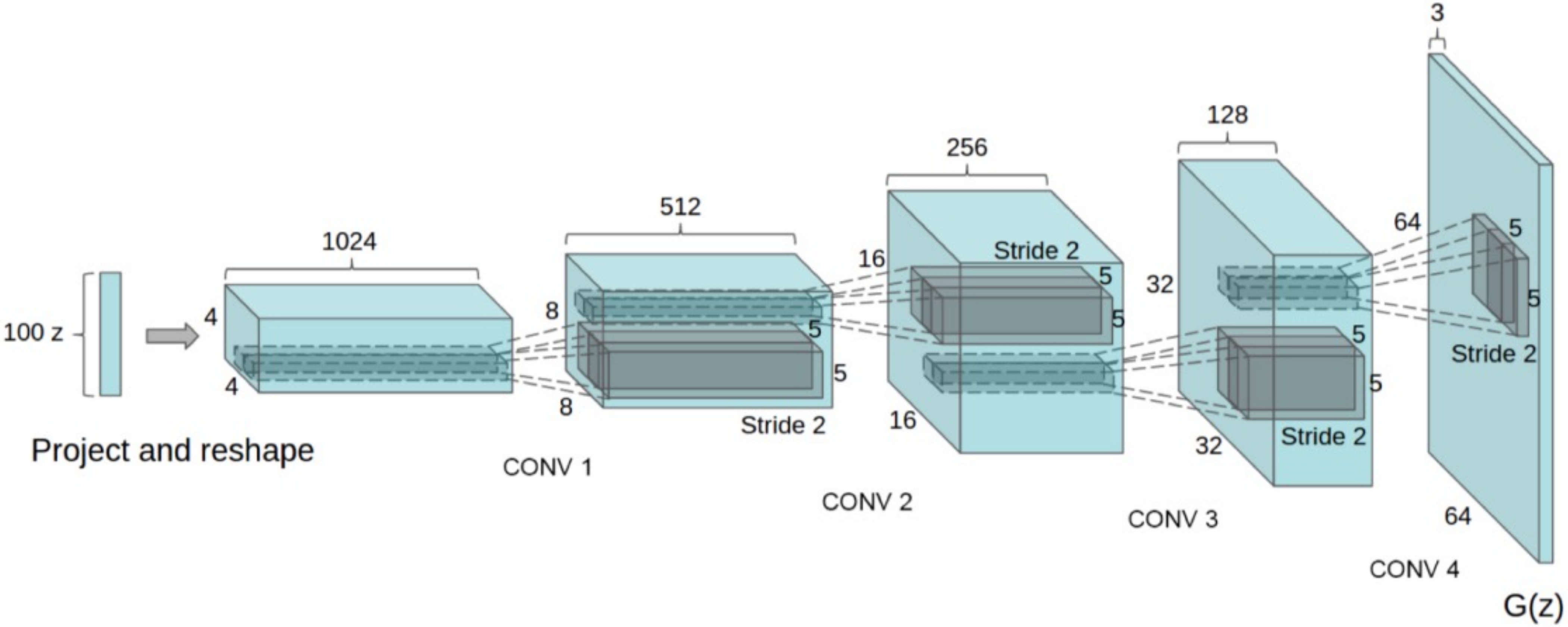
UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ICLR 2016

- **First to generate plausible results at 64x64.**
- **Improved architectures for generator/discriminator**
  - **Use strided convolution – no pooling**
- **Most GAN archi**
  similar



- **Generator architecture**
- **Good default GAN architecture**

DCGAN

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ICLR 2016

- First to generate plausible results at 64x64.
- Improved architectures for generator/discriminator
- Most GAN architectures used now are similar

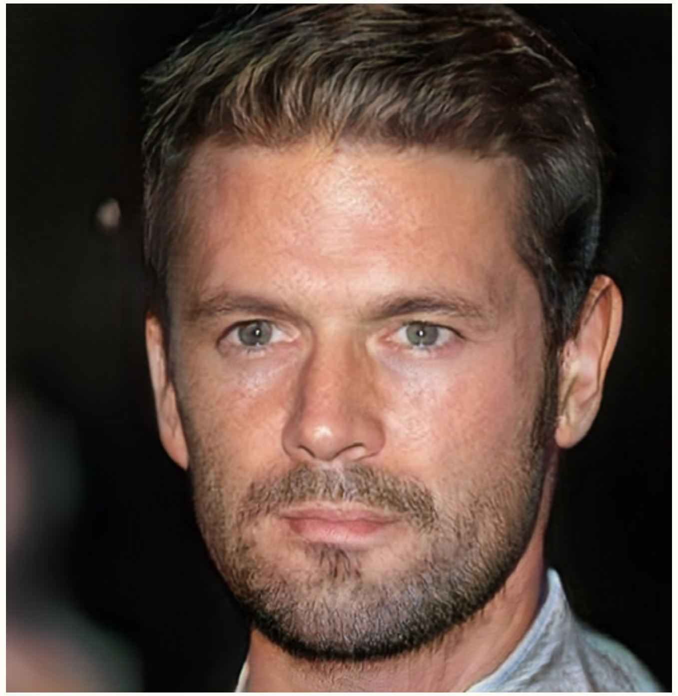
- Lots of tricks to get GANs to train well

3.5 Years of Progress on Faces

2014

2015

2016



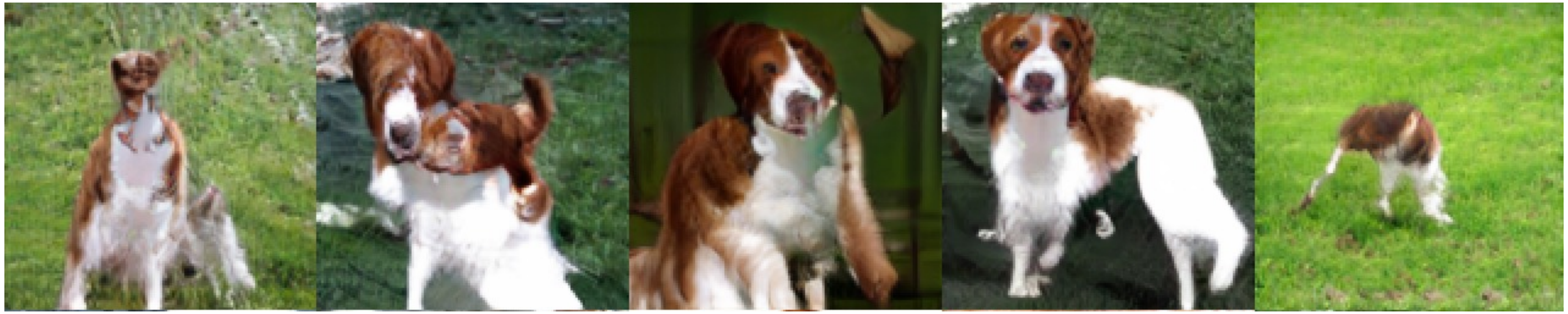
2017

(Brundage et al, 2018)

<2 Years of Progress on ImageNet

Odena et al
2016

Miyato et al
2017



Zhang et al
2018

(Goodfellow 2018)

Evaluation of Generative Models

- Short answer: hard
- Log-likelihood possible for some models
  - Compute log p(x) on validation set
  - Not GANs
  - See [A note on the evaluation of generative models, Lucas Theis, Aäron van den Oord, Matthias Bethge, ICLR 2016]
- Inception score (IS)
- Frechet Inception Distance (FID)
- User-study: can humans tell fake from real?

Inception Score

Proposed in 2016

Improved Techniques for Training GANs

Tim Salimans tim@openai.com
Ian Goodfellow ian@openai.com
Wojciech Zaremba woj@openai.com
Vicki Cheung vicki@openai.com
Alec Radford alec.radford@gmail.com
Xi Chen peter@openai.com

Abstract

Inception Score

- Send generated image through Inception model (trained on Imagenet)

generated image to get the conditional label distribution $p(y|\boldsymbol{x})$. Images that contain meaningful objects should have a conditional label distribution $p(y|\boldsymbol{x})$ with low entropy. Moreover, we expect the model to generate varied images, so the marginal $\int p(y|\boldsymbol{x} = G(z))dz$ should have high entropy. Combining these two requirements, the metric that we propose is: $\exp(\mathbb{E}_{\boldsymbol{x}}\mathrm{KL}(p(y|\boldsymbol{x})||p(y)))$, where

Inception Score

- Send generated image through Inception model (trained on Imagenet)

generated image to get the conditional label distribution $p(y|\boldsymbol{x})$. Images that contain meaningful objects should have a conditional label distribution $p(y|\boldsymbol{x})$ with low entropy. Moreover, we expect the model to generate varied images, so the marginal $\int p(y|\boldsymbol{x} = G(z))dz$ should have high entropy. Combining these two requirements, the metric that we propose is: $\exp(\mathbb{E}_{\boldsymbol{x}} \mathrm{KL}(p(y|\boldsymbol{x}) || p(y)))$, where

Inception Score

- Send generated image through Inception model (trained on Imagenet)

generated image to get the conditional label distribution $p(y|\boldsymbol{x})$. Images that contain meaningful objects should have a conditional label distribution $p(y|\boldsymbol{x})$ with low entropy. Moreover, we expect the model to generate varied images, so the marginal $\int p(y|\boldsymbol{x} = G(z))dz$ should have high entropy. Combining these two requirements, the metric that we propose is: $\exp(\mathbb{E}_{\boldsymbol{x}}\mathrm{KL}(p(y|\boldsymbol{x})||p(y)))$, where

Frechet Inception Distance

- Two image sets: S1={real_images}, S2={samples_from_model}
- Take features f1,f2 from top of pre-trained convnet for recognition, e.g. Inception-NetV3 → 2048D
- Fit multi-dimensional Gaussians to features:
  - N(mu_1,C_1) & N(mu_2,C_2)
  - mu is mean across images in set
  - Sigma is covariance matrix across images in set
- FID^2 = ||mu_1 – mu_2||^2 + Tr(C_1 + C_2 – 2*sqrt(C_1*C_2))
- Measure between distributions in feature space

How to Train a GAN

Emily Denton, Martin Arjovsky, Michael Mathieu

New York University

Ian Goodfellow

Google

Soumith Chintala

Facebook AI Research

The stability of GANs

Timeline - the stability of GANs

Goodfellow et. al. “Generative Adversarial Networks”

2014

Timeline - the stability of GANs

model architecture generator

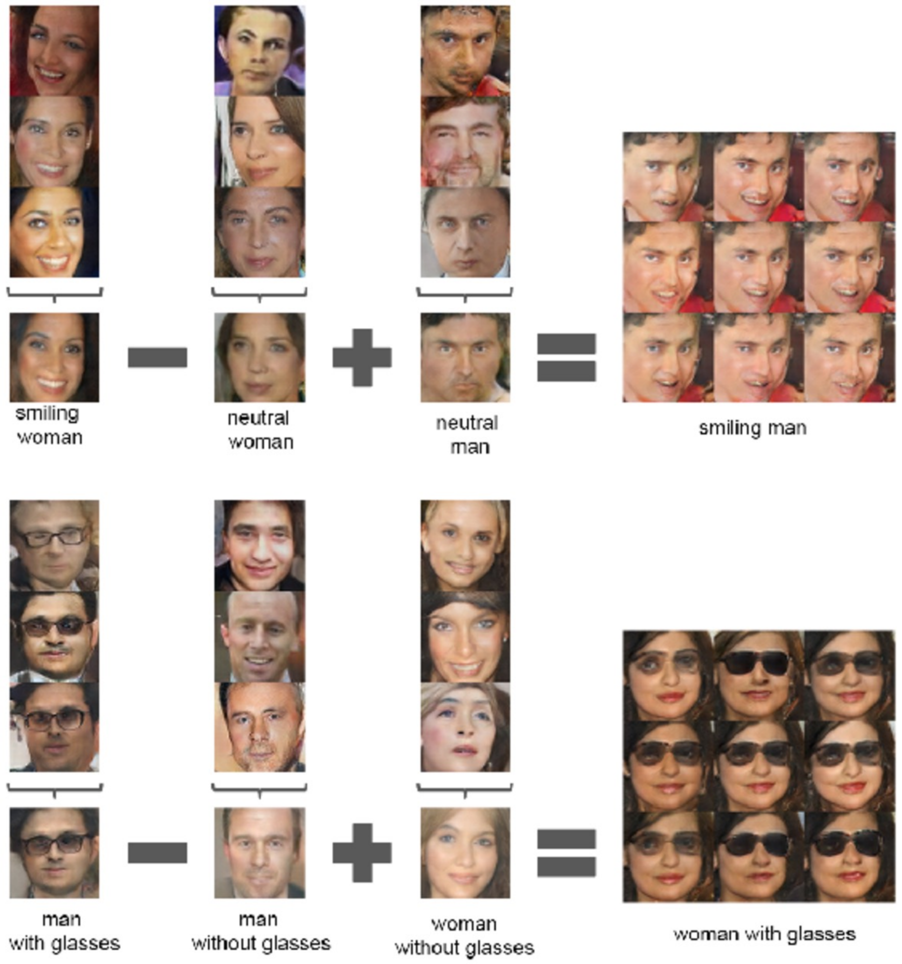
visual inspection

countless failed stability hacks

Denton et. al. “Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks”

2015

Timeline - the stability of GANs



countless hours finding stable models

stable upto 64x64

mode dropping

underfitting

Radford et. al. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"

2015

Timeline - the stability of GANs

more heuristics

more stability

Salimans et. al. "Improved Techniques for Training GANs"

2015

Timeline - the stability of GANs

gradient norm regularization

Least-Squares
Boundary Equilibrium

Gulrajani et. al. "Improved Training of Wasserstein GANs"

Xudong et al. "Least squares generative adversarial networks."

Berthelot et. al. "Began: Boundary equilibrium generative adversarial networks."

2016-2017

Timeline - the stability of GANs

https://github.com/khanrc/tf.gans-comparison
by Junbum Cha

GANs comparison without cherry-picking

Implementations of some theoretical generative adversarial nets: DCGAN, EBGAN, LSGAN, WGAN, WGAN-GP, BEGAN, DRAGAN and CoulombGAN.

I implemented the structure of model equal to the structure in paper and compared it on the CelebA dataset and LSUN dataset without cherry-picking.

Comparison to Classification ConvNets

- Throw things at the wall and see what sticks
- Intuition is poorer
- Theoretical work is somewhat improving but still far away
- Objective validation metrics are not there yet

#1: Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output
  - or some kind of bounds normalization

#2: Modified loss function (classic GAN)

- In papers people write min (log 1-D), but in practice folks practically use max log D
  - because the first formulation has vanishing gradients early on
  - Goodfellow et. al (2014)
- In practice:
  - Flip labels when training generator: real = fake, fake = real

#2: Modified loss function (classic GAN)

- In papers people write min (log 1-D), but in practice folks practically use max log D
  - because the first formulation has vanishing gradients early on
  - Goodfellow et. al (2014)

LOT OF NEW LOSS FORMULATIONS

- In practice:
  - Flip labels when training generator: real = fake, fake = real

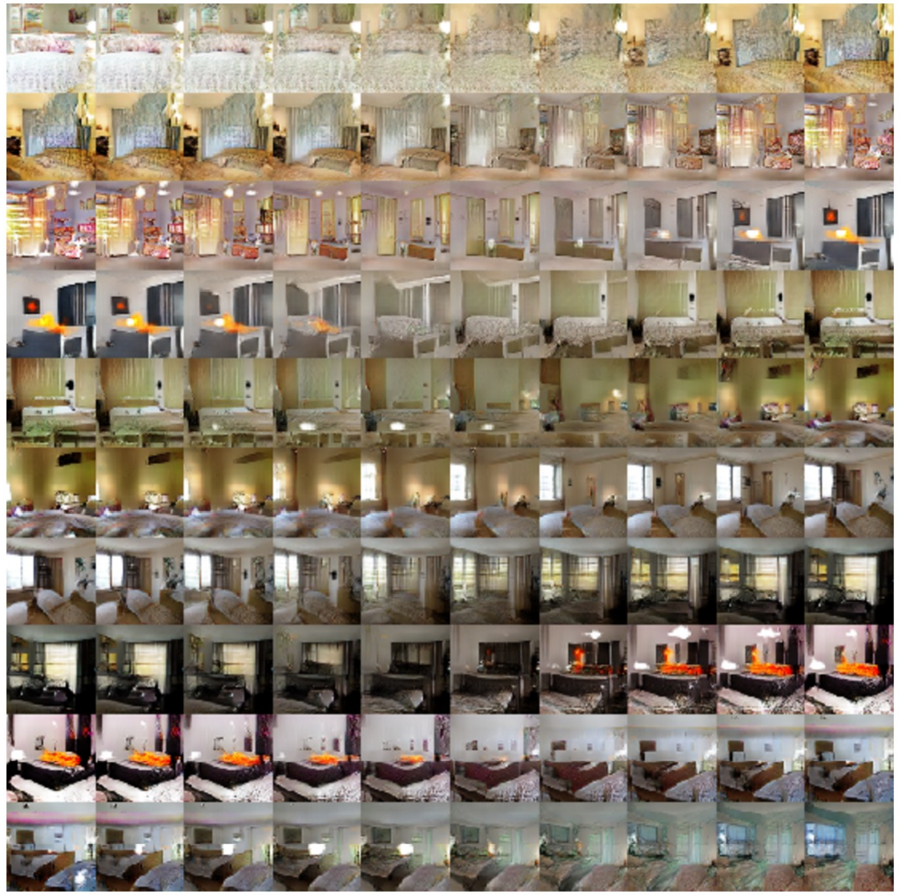
#2: Modified loss function (classic GAN)

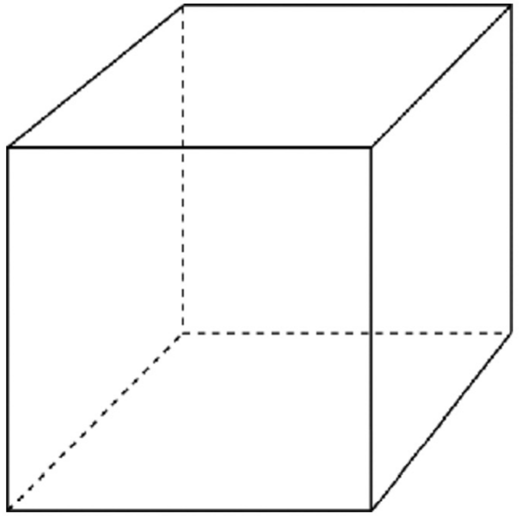
https://github.com/hwalsuklee/tensorflow-generative-model-collections
https://github.com/znxlwm/pytorch-generative-model-collections

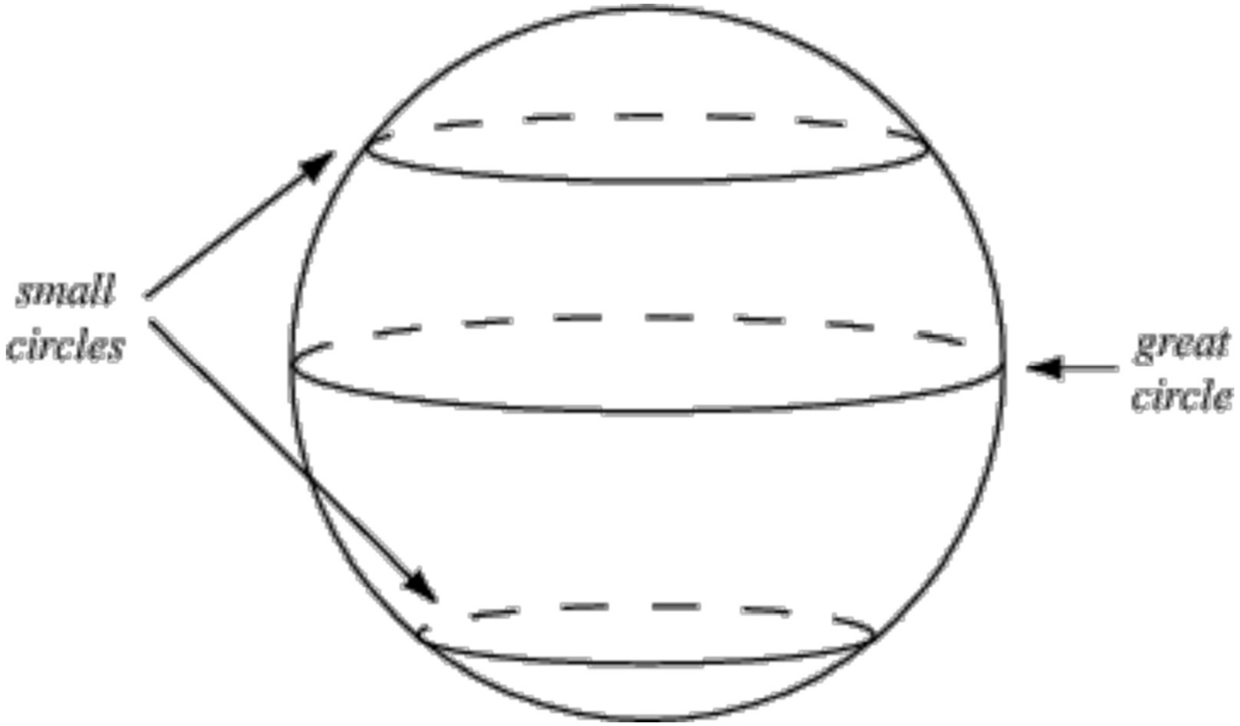
Name	Paper Link	Value Function
GAN	Arxiv	$L_D^{GAN} = E\left[\log\left(D(x)\right)\right] + E\left[\log\left(1 - D(G(z))\right)\right]$ $L_G^{GAN} = E\left[\log\left(D(G(z))\right)\right]$
LSGAN	Arxiv	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	Arxiv	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN-GP	Arxiv	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha G(z)))\rvert - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	Arxiv	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha x_p))\rvert - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	Arxiv	$L_D^{CGAN} = E\left[\log\left(D(x, c)\right)\right] + E\left[\log\left(1 - D(G(z), c)\right)\right]$ $L_G^{CGAN} = E\left[\log\left(D(G(z), c)\right)\right]$
infoGAN	Arxiv	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c, c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c, c')$
ACGAN	Arxiv	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c \mid x)] + E[P(class = c \mid G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c \mid G(z))]$
EBGAN	Arxiv	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	Arxiv	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

#3: Use spherical z

- interpolation via great circle
- Tom White “Sampling Generative Networks”
  - https://arxiv.org/abs/1609.04468

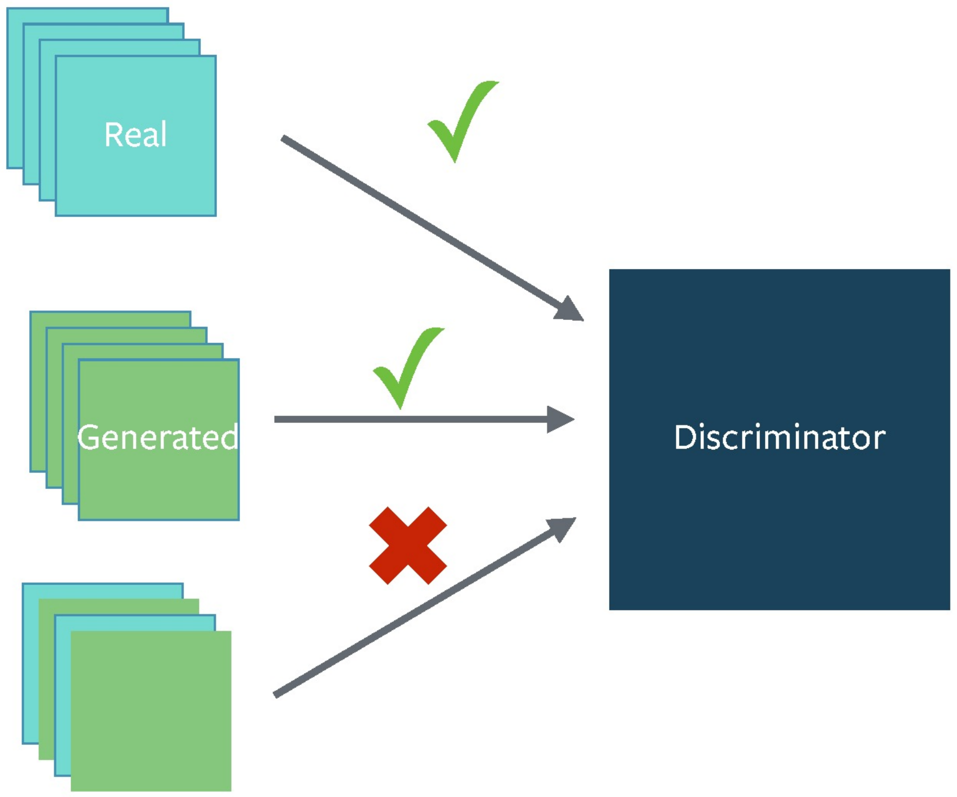






#4: BatchNorm

- different batches for real and fake
- when batchnorm is not an option use instance norm



#5: Avoid Sparse Gradients: ReLU, MaxPool

- the stability of the GAN game suffers
- LeakyReLU (both G and D)
- Downsampling: Average Pooling, Conv2d + stride
- Upsampling: PixelShuffle, ConvTranspose2d + stride
  - PixelShuffle: https://arxiv.org/abs/1609.05158

#6: Soft and Noisy Labels

- Label Smoothing
- making the labels the noisy a bit for the discriminator, sometimes
  - Salimans et. al. 2016

#7: Architectures: DCGANs / Hybrids

- DCGAN when you can

- if you cant use DCGANs and no model is stable,
- use a hybrid model : KL + GAN or VAE + GAN

- ResNets from WGAN-gp also work pretty well (but are very slow)
  - https://github.com/igul222/improved_wgan_training

- Width matters more than Depth

#8: Stability tricks from RL

- Experience replay
- Things that work for deep deterministic policy gradients
- See Pfau & Vinyals (2016)

#9: Optimizer: ADAM

- optim.Adam rules!
  - See Radford et. al. 2015
- [MMathieu] Use SGD for discriminator and ADAM for generator

#10: Use Gradient Penalty

- Regularize the norm of the gradients
  - multiple theories on why this is useful (WGAN-GP, DRAGAN, Stabilizing GANs by Regularization etc.)

#11: Dont balance via loss statistics (classic GAN)

- Dont try to find a (number of G / number of D) schedule to uncollapse training

- while lossD > X:
  - train D
- while lossG > X:
  - train G

#12: If you have labels, use them

- if you have labels available, training the discriminator to also classify the samples: auxillary GANs

#13: Add noise to inputs, decay over time

- Add some artificial noise to inputs to D (Arjovsky et. al., Huszar, 2016)
  - http://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/
  - https://openreview.net/forum?id=Hk4_qw5xe
- adding gaussian noise to every layer of generator (Zhao et. al. EBGAN)
- Improved GANs: OpenAI code also has it (commented out)

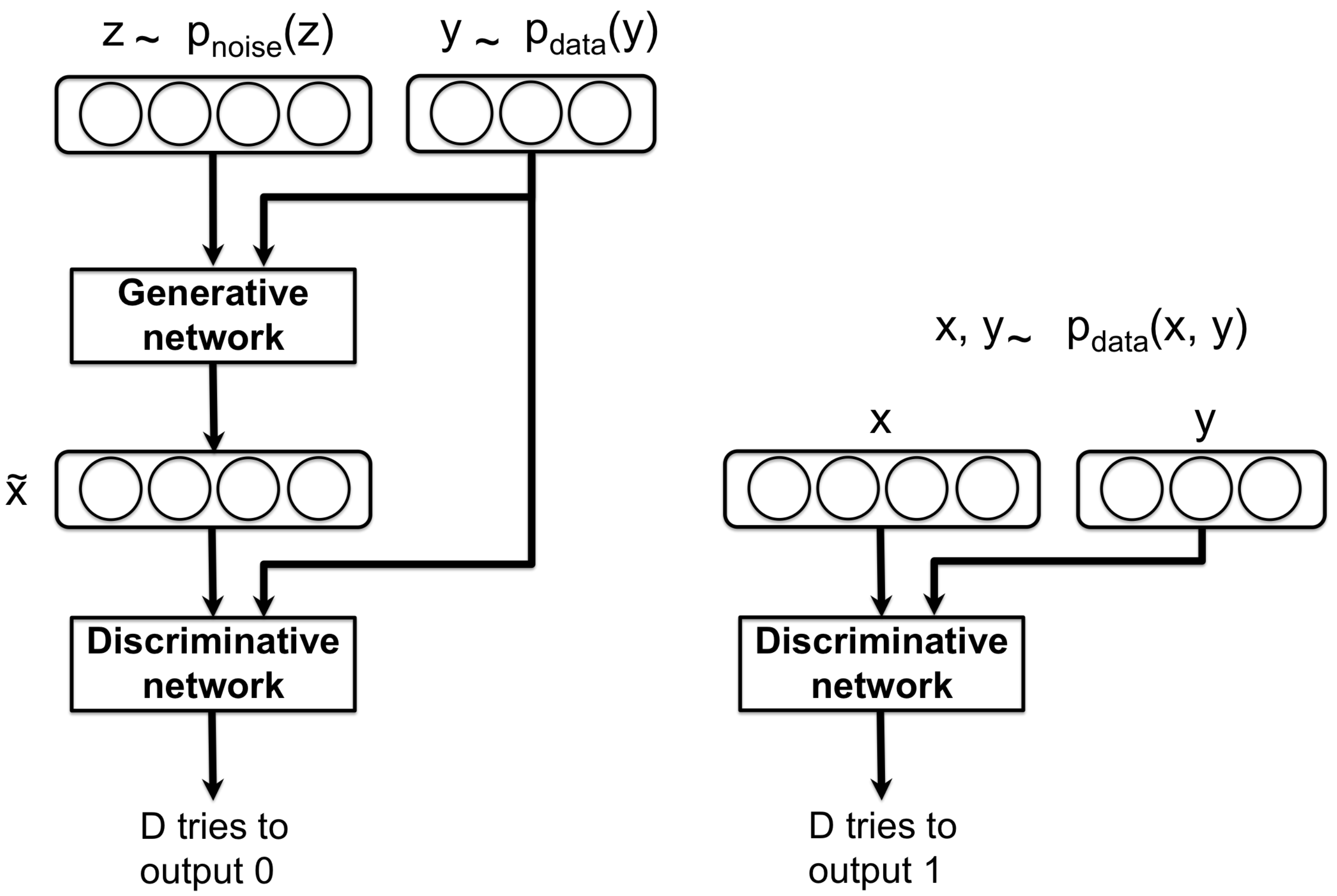
#14: Train discriminator more

- especially when you have noise
- hard to find a schedule of number of D iterations vs G iterations
- WGAN/WGAN-gp papers suggest 5x D iterations per G iteration

Conditional GANs

Conditional generative adversarial networks (CGAN)

- Condition generation on additional info **y** (e.g. class label, another image)
- *D* has to determine if samples are realistic given **y**



[Mirza and Osindero (2014); Gauthier (2014)]

#16: Discrete Variables

- Use an Embedding layer
- Add as additional channels to images
- Keep embedding dimensionality low and upsample to match image channel size

Conclusion

- Model stability is improving
- Theory is improving
- Hacks are a stop-gap

Progressive Growing of GANs for Improved Quality, Stability, and Variation

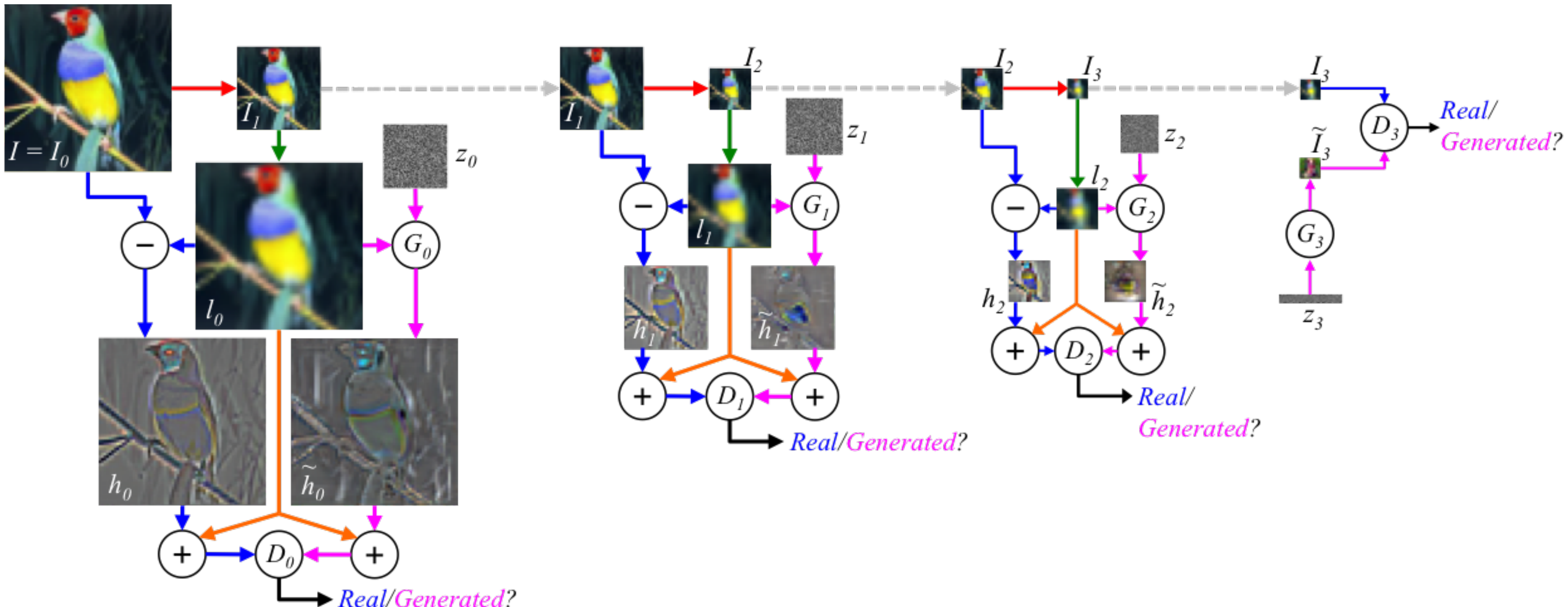
Tero Karras NVIDIA
Timo Aila NVIDIA
Samuli Laine NVIDIA
Jaakko Lehtinen NVIDIA and Aalto University
{tkarras,taila,slaine,jlehtinen}@nvidia.com

ICLR 2018

Prior Work

Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

Emily Denton[1*], Soumith Chintala[2*],
Arthur Szlam[2], Rob Fergus[2]
NeurIPS 2015



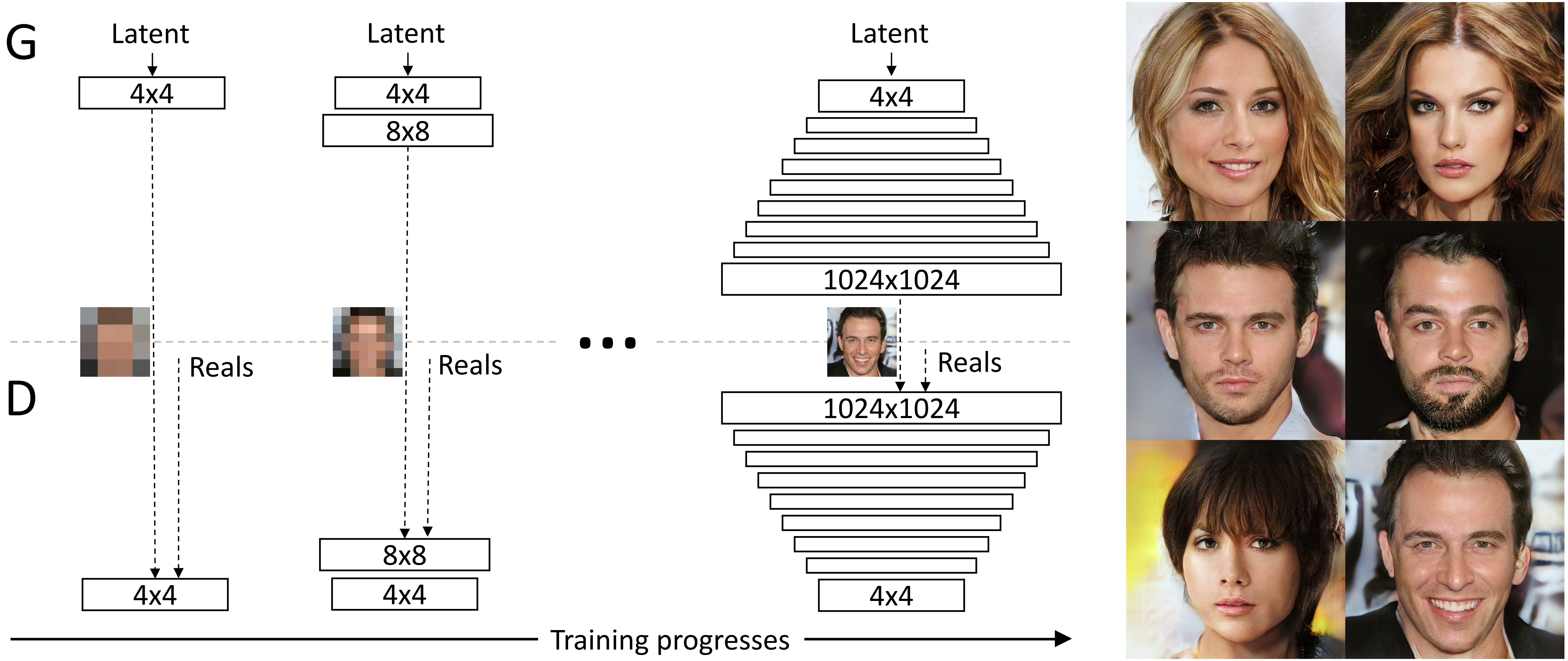
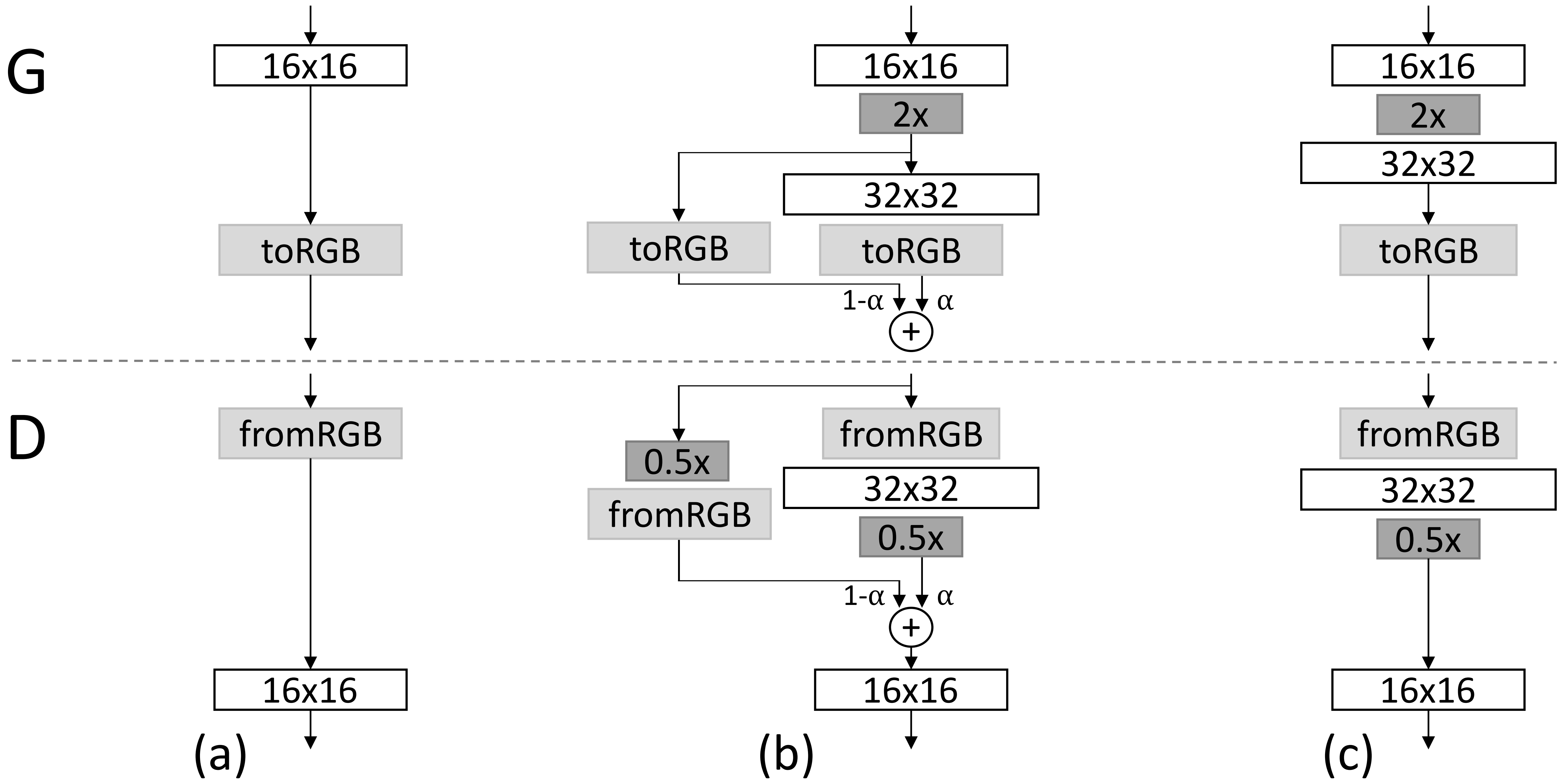


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

Smooth blending with scale increase



Training configuration	CELEBA Sliced Wasserstein distance $\times 10^3$					CELEBA MS-SSIM	LSUN BEDROOM Sliced Wasserstein distance $\times 10^3$					LSUN BEDROOM MS-SSIM
	128	64	32	16	Avg		128	64	32	16	Avg	
(a) Gulrajani et al. (2017)	12.99	7.79	7.62	8.73	9.28	0.2854	11.97	10.51	8.03	14.48	11.25	**0.0587**
(b) + Progressive growing	4.62	**2.64**	3.78	6.06	4.28	**0.2838**	7.09	6.27	7.40	9.64	7.60	0.0615
(c) + Small minibatch	75.42	41.33	41.62	26.57	46.23	0.4065	72.73	40.16	42.75	42.46	49.52	0.1061
(d) + Revised training parameters	9.20	6.53	4.71	11.84	8.07	0.3027	7.39	5.51	3.65	9.63	6.54	0.0662
(e*) + Minibatch discrimination	10.76	6.28	6.04	16.29	9.84	0.3057	10.29	6.22	5.32	11.88	8.43	0.0648
(e) Minibatch stddev	13.94	5.67	2.82	5.71	7.04	0.2950	7.77	5.23	3.27	9.64	6.48	0.0671
(f) + Equalized learning rate	4.42	3.28	2.32	7.52	4.39	0.2902	**3.61**	3.32	**2.71**	6.44	4.02	0.0668
(g) + Pixelwise normalization	**4.06**	3.04	**2.02**	**5.13**	**3.56**	0.2845	3.89	**3.05**	3.24	**5.87**	**4.01**	0.0640
(h) Converged	2.42	2.17	2.24	4.99	2.96	0.2828	3.47	2.60	2.30	4.87	3.31	0.0636

Table 1: Sliced Wasserstein distance (SWD) between the generated and training images (Section 5) and multi-scale structural similarity (MS-SSIM) among the generated images for several training setups at 128×128. For SWD, each column represents one level of the Laplacian pyramid, and the last one gives an average of the four distances.

(a) (b) (c) (d) (e∗) (e) (f) (g) (h) Converged

Figure 3: (a) – (g) CELEBA examples corresponding to rows in Table 1. These are intentionally non-converged. (h) Our converged result. Notice that some images show aliasing and some are not sharp – this is a flaw of the dataset, which the model learns to replicate faithfully.

Figure 5: 1024×1024 images generated using the CELEBA-HQ dataset. See Appendix F for a

Mao et al. (2016b) (128×128) Gulrajani et al. (2017) (128×128) Our (256×256)

POTTEDPLANT HORSE SOFA BUS CHURCHOUTDOOR BICYCLE TVMONITOR

Nearest-Neighbor Sanity Check

A Style-Based Generator Architecture for Generative Adversarial Networks

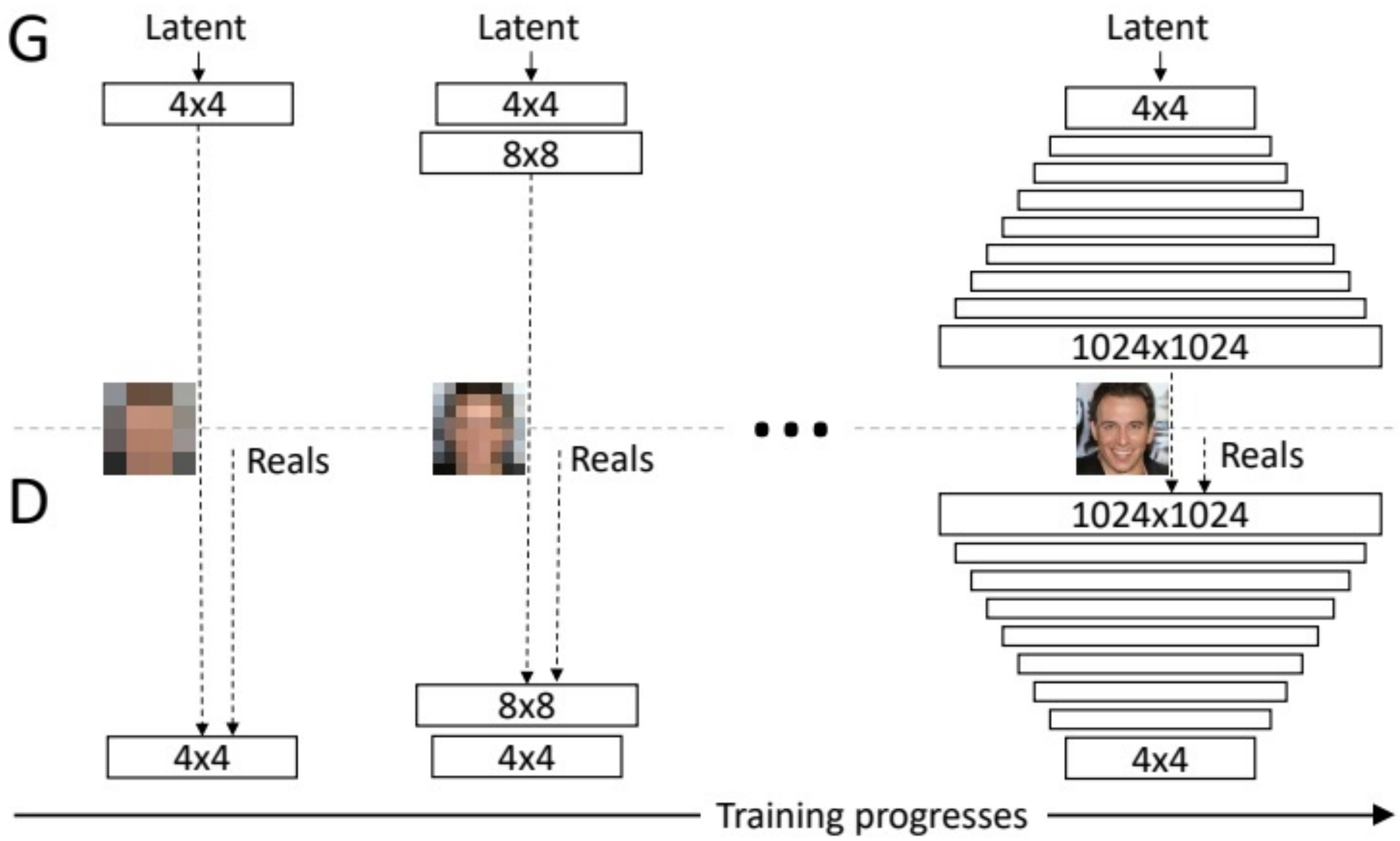
Tero Karras
NVIDIA
tkarras@nvidia.com

Samuli Laine
NVIDIA
slaine@nvidia.com

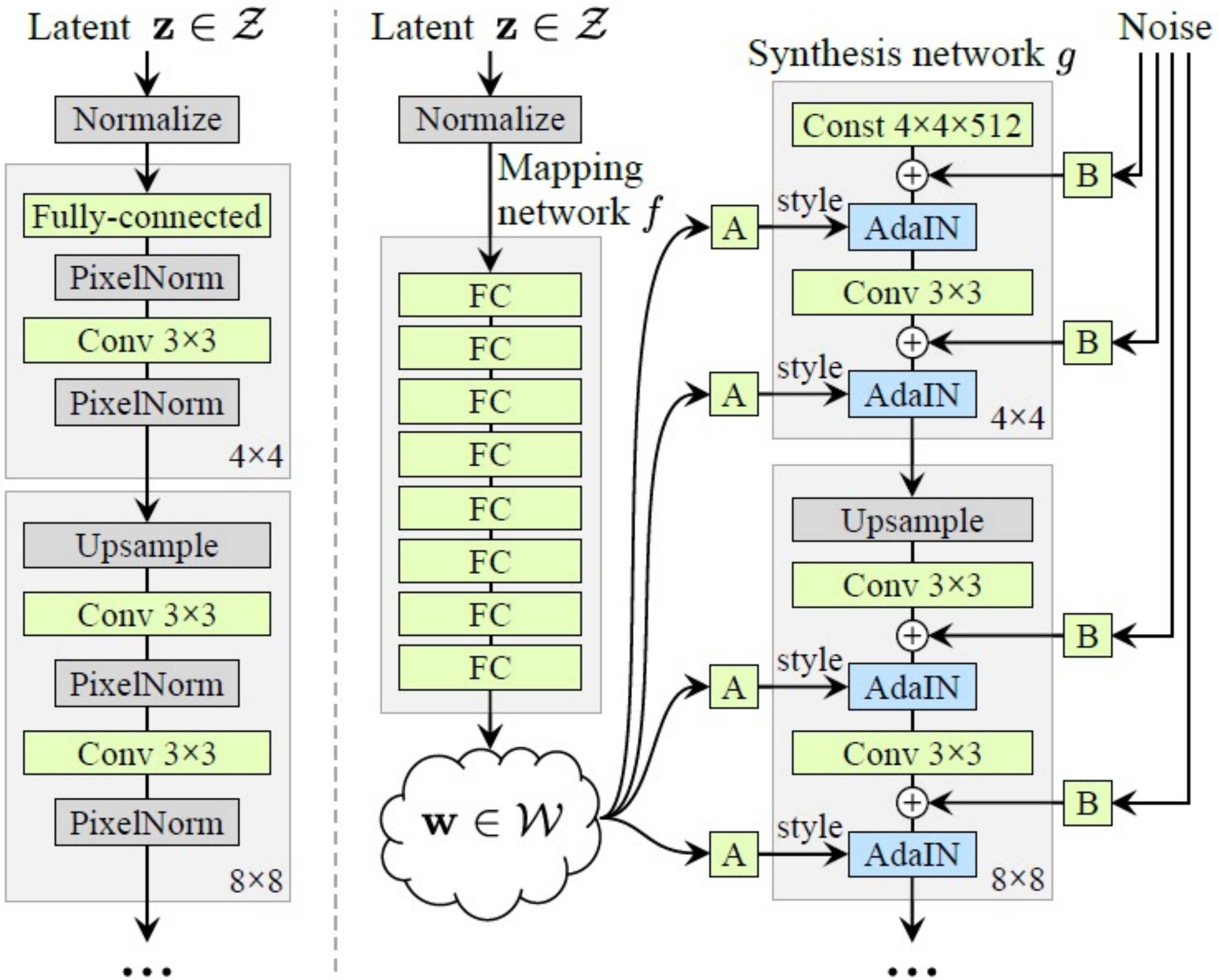
Timo Aila
NVIDIA
taila@nvidia.com

https://arxiv.org/pdf/1812.04948.pdf
CVPR 2019

Baseline Progressive GAN



Style-based generator



(a) Traditional (b) Style-based generator

AdaIN: adaptive instance normalization

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i}\frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

A: a learned affine transform

B: learned per-channel scaling factors

A Style-Based Generator Architecture for Generative Adversarial Networks. Ero Karras, Samuli Laine, Timo Aila. arXiv 2018


Source A
Source B
Coarse styles from source B
Middle styles from source B
Fine from B

Stochastic variation

(a) Generated image (b) Stochastic variation (c) Standard deviation

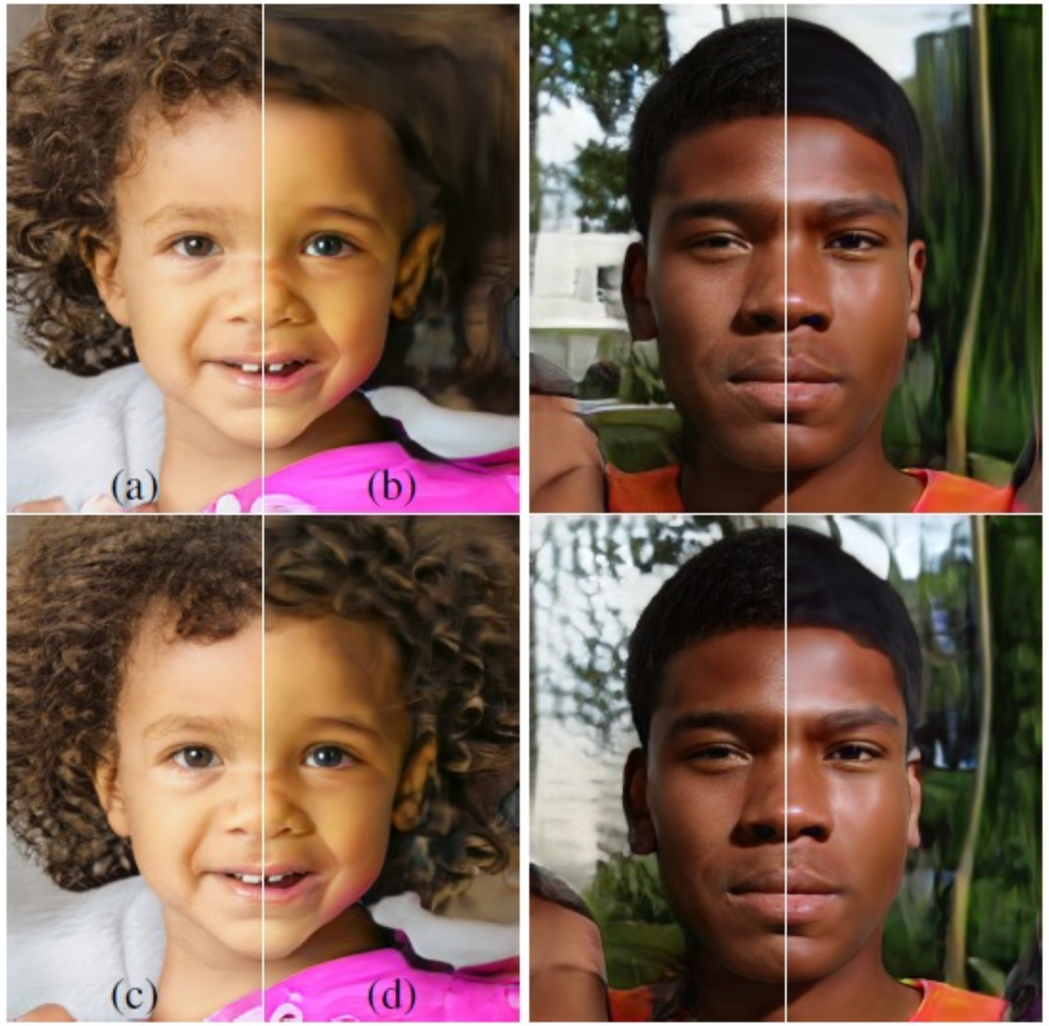
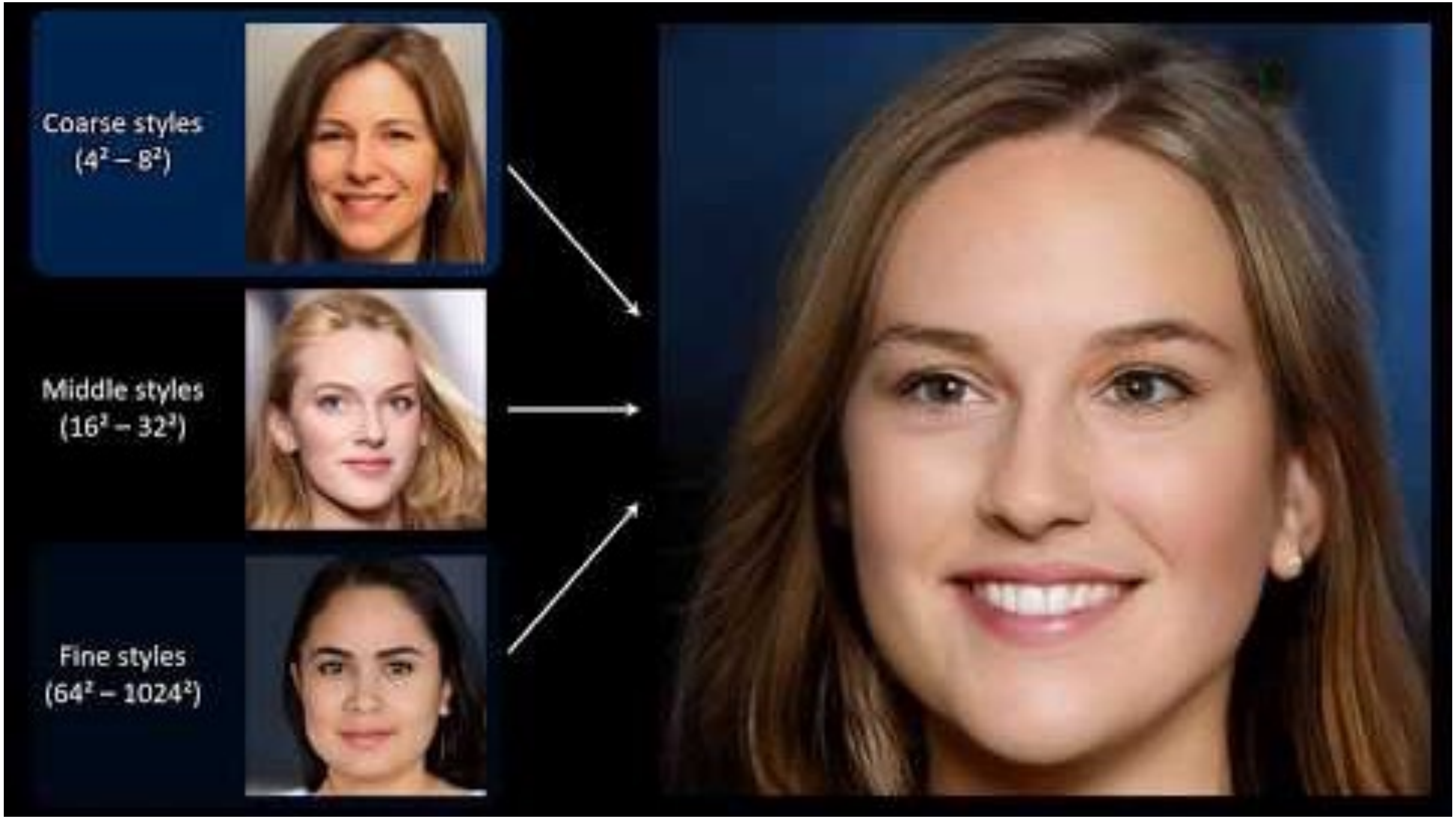


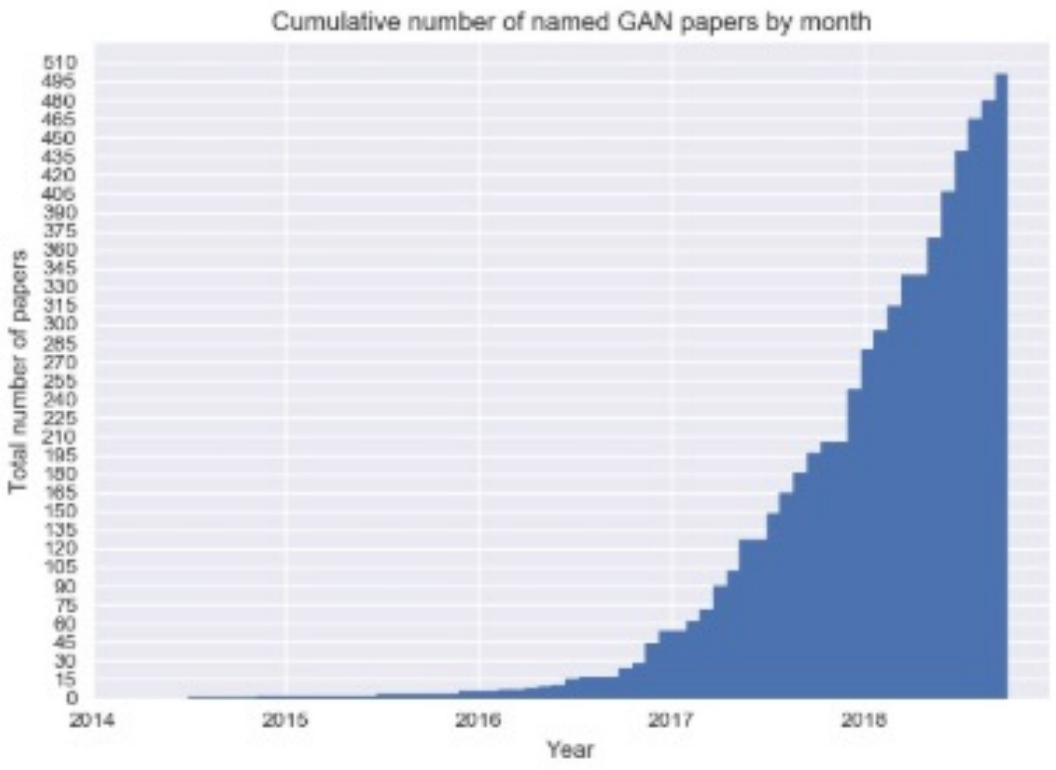
Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only (64^2 – 1024^2). (d) Noise in coarse layers only (4^2 – 32^2). We can see that the artificial omission of noise leads to featureless "painterly" look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

StyleGAN


Coarse styles
($4^2 - 8^2$)
Middle styles
($16^2 - 32^2$)
Fine styles
($64^2 - 1024^2$)

The GAN Zoo

https://github.com/hindupuravinash/the-gan-zoo



- 3D-ED-GAN - Shape Inpainting using 3D Generative Adversarial Network and Recurrent Convolutional Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling (github)
- 3D-IWGAN - Improved Adversarial Systems for 3D Object Generation and Reconstruction (github)
- 3D-PhysNet - 3D-PhysNet: Learning the Intuitive Physics of Non-Rigid Object Deformations
- 3D-RecGAN - 3D Object Reconstruction from a Single Depth View with Adversarial Learning (github)
- ABC-GAN - ABC-GAN: Adaptive Blur and Control for improved training stability of Generative Adversarial Networks (github)
- ABC-GAN - GANs for LIFE: Generative Adversarial Networks for Likelihood Free Inference
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- ACGAN - Coverless Information Hiding Based on Generative adversarial networks
- acGAN - On-line Adaptative Curriculum Learning for GANs
- ACtuAL - ACtuAL: Actor-Critic Under Adversarial Learning
- AdaGAN - AdaGAN: Boosting Generative Models
- Adaptive GAN - Customizing an Adversarial Example Generator with Class-Conditional GANs
- AdvEntuRe - AdvEntuRe: Adversarial Training for Textual Entailment with Knowledge-Guided Examples
- AdvGAN - Generating adversarial examples with adversarial networks
- AE-GAN - AE-GAN: adversarial eliminating with GAN
- AE-OT - Latent Space Optimal Transport for Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AF-DCGAN - AF-DCGAN: Amplitude Feature Deep Convolutional GAN for Fingerprint Construction in Indoor Localization System
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AIM - Generating Informative and Diverse Conversational Responses via Adversarial Information Maximization
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference (github)

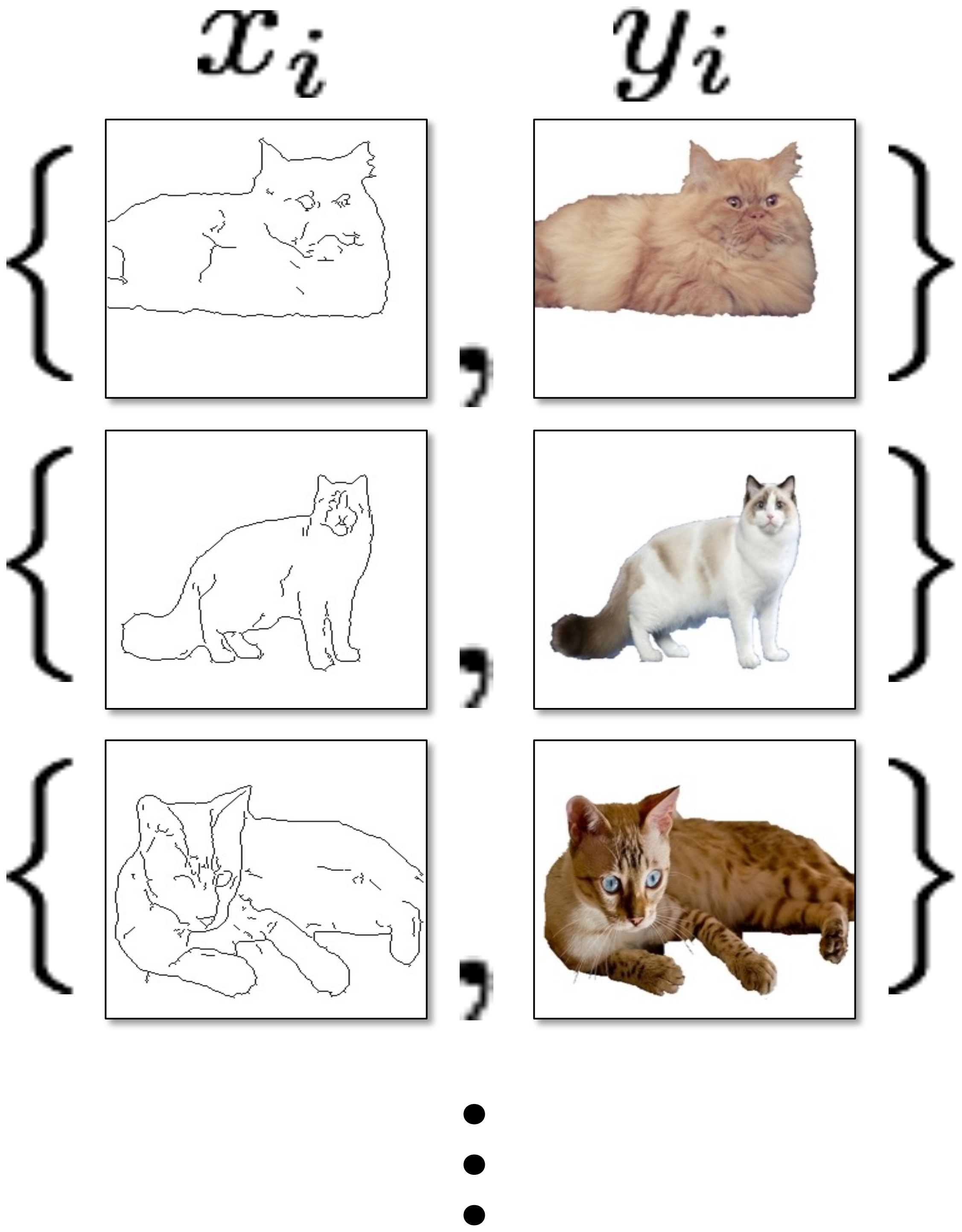
Unpaired Image-to-Image Translation with CycleGAN

Jun-Yan Zhu and Taesung Park
Joint work with Phillip Isola and Alexei A. Efros

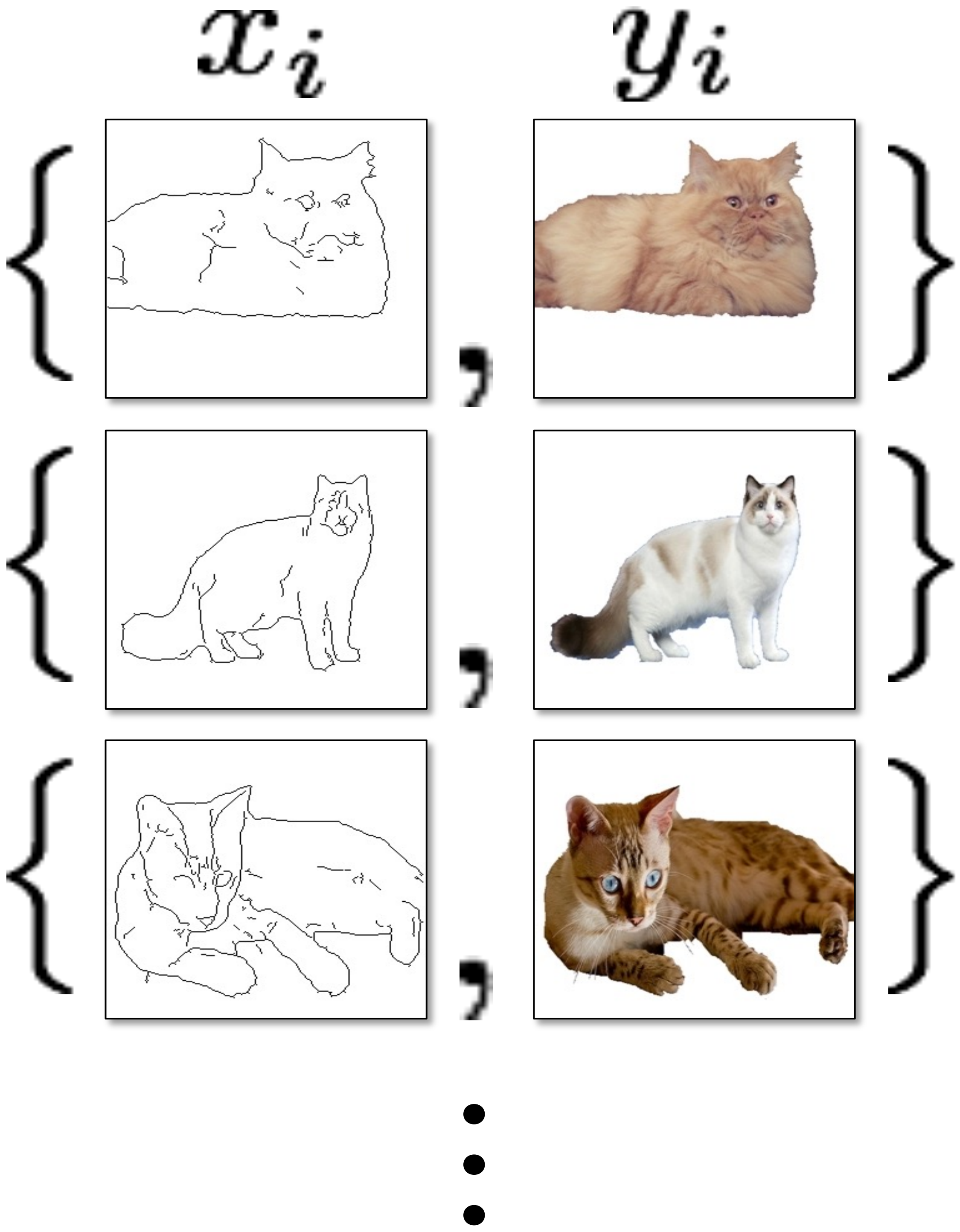
BAIR
BERKELEY ARTIFICIAL INTELLIGENCE RESEARCH

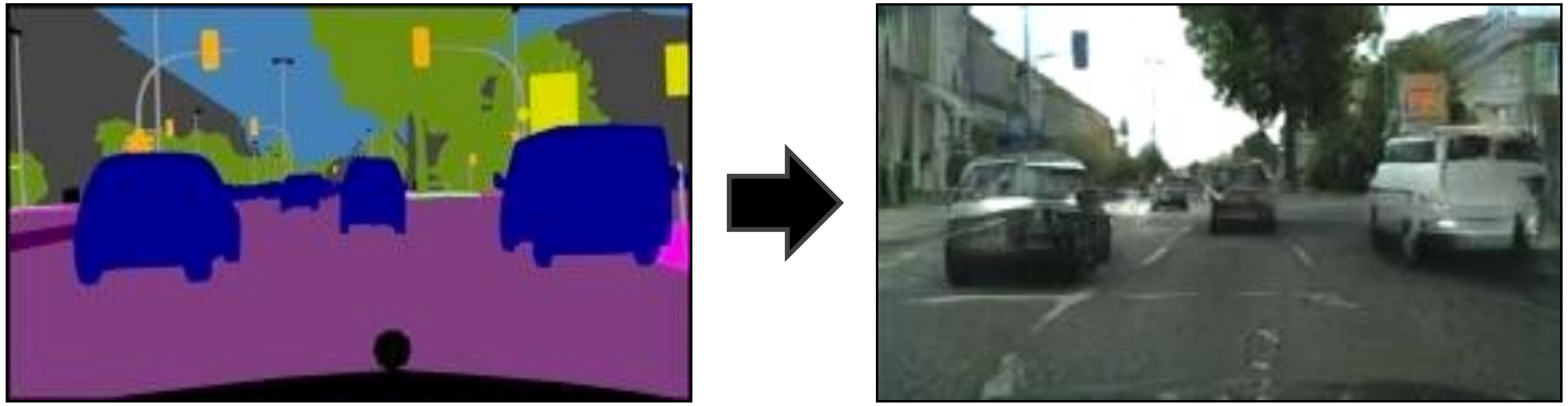
CSAIL

Paired

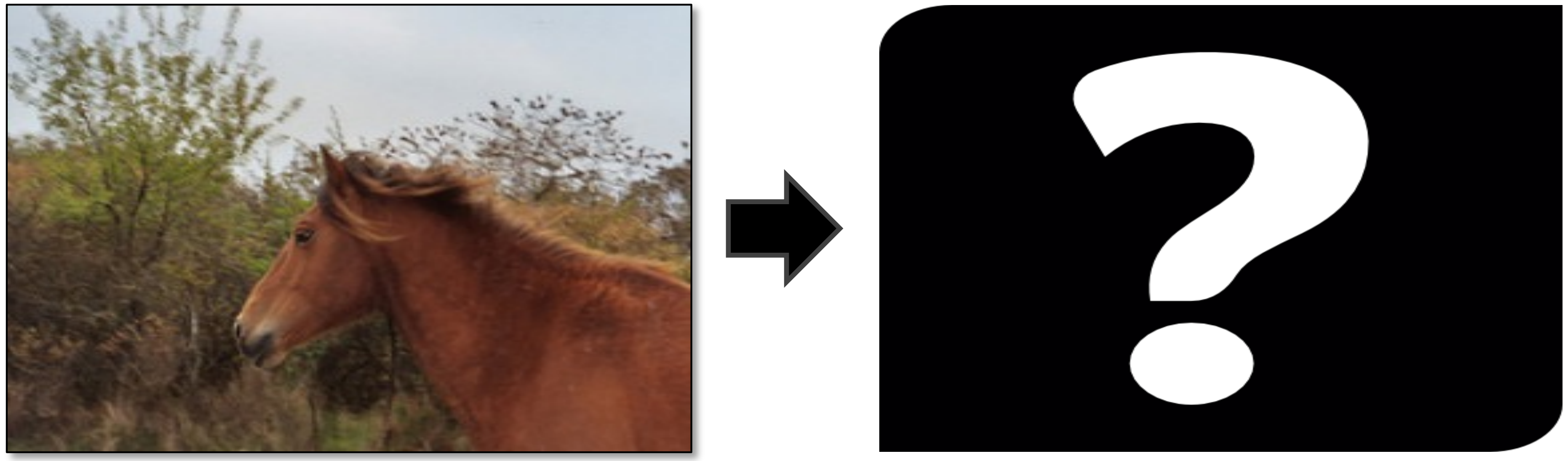


Paired



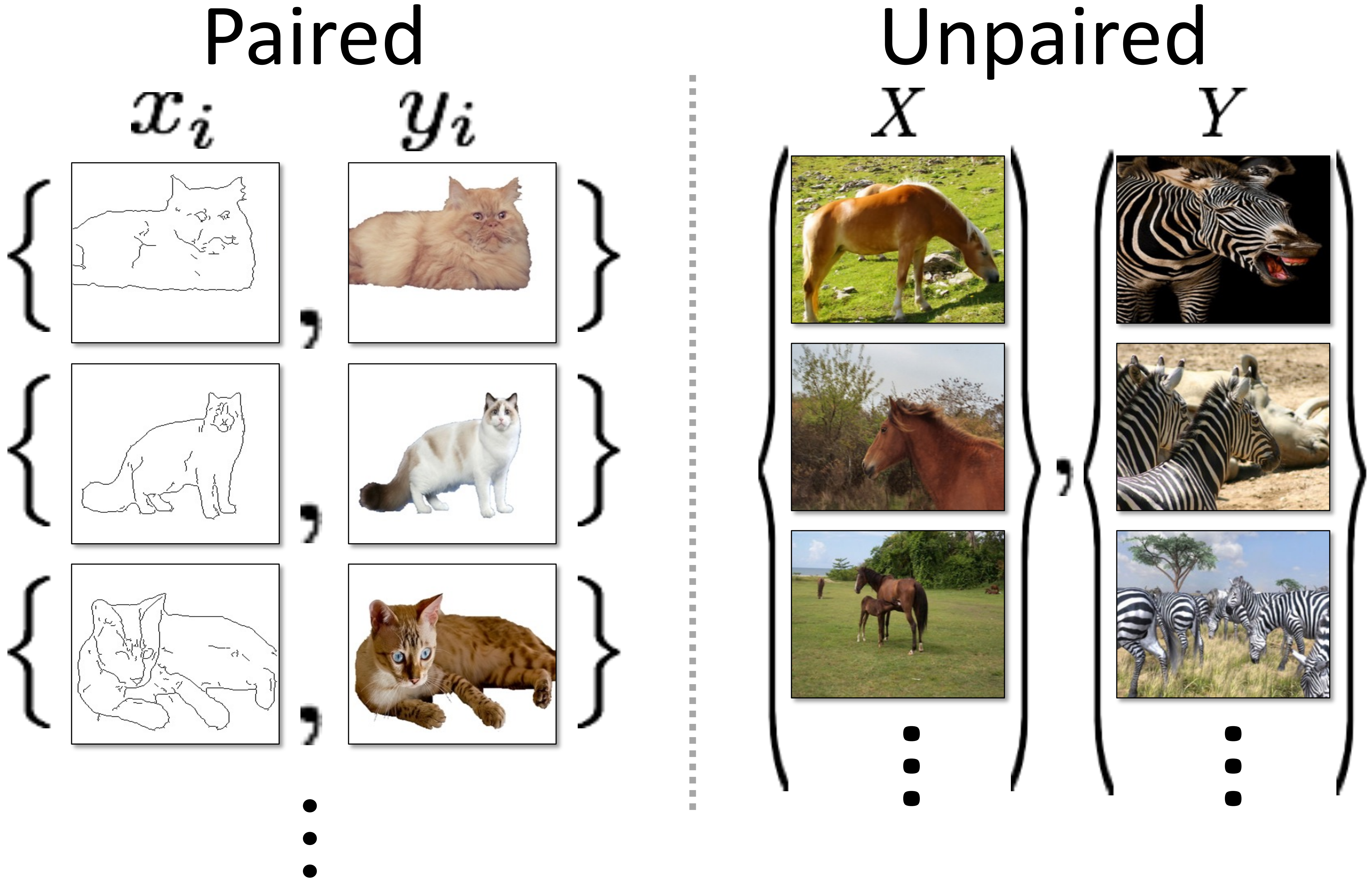


Label ↔ photo: per-pixel labeling

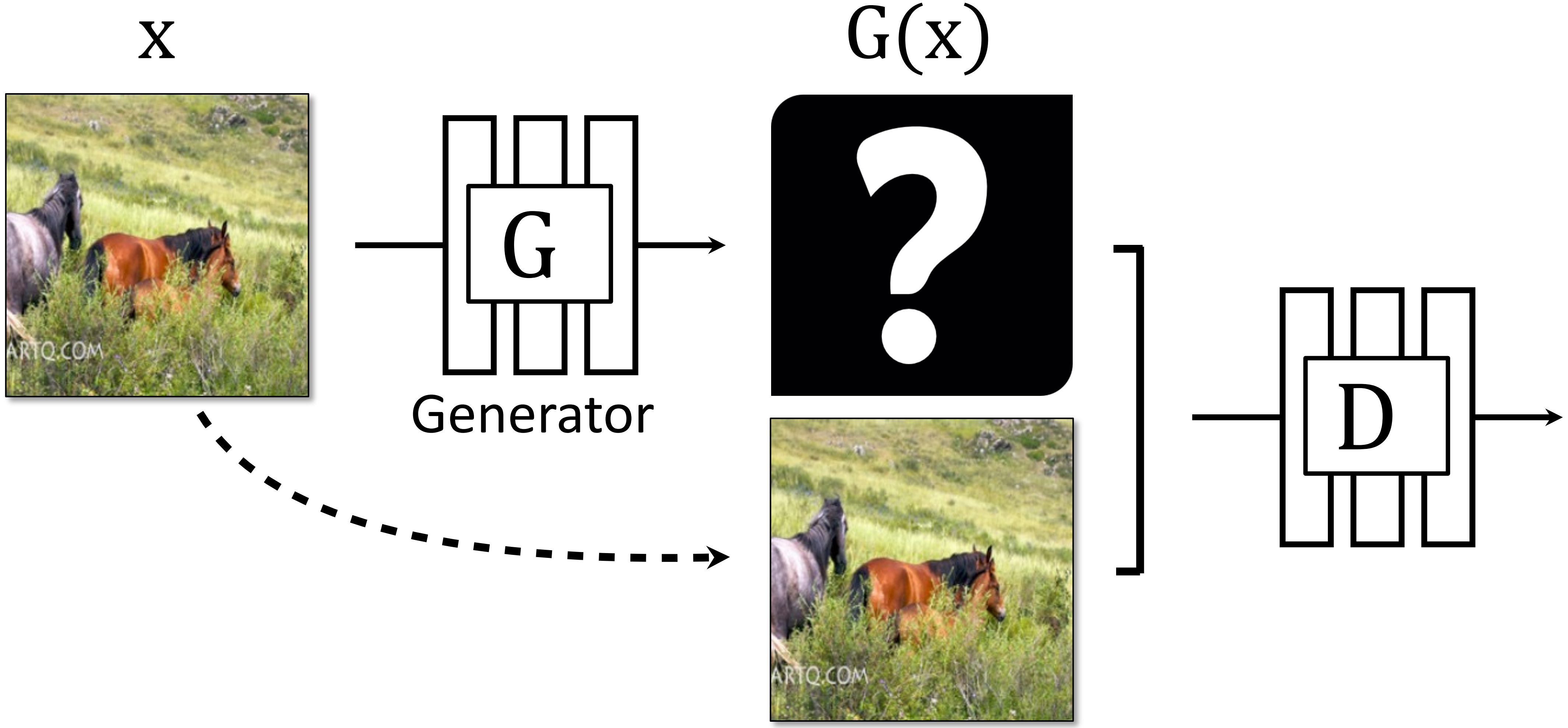


Horse ↔ zebra: how to get zebras?

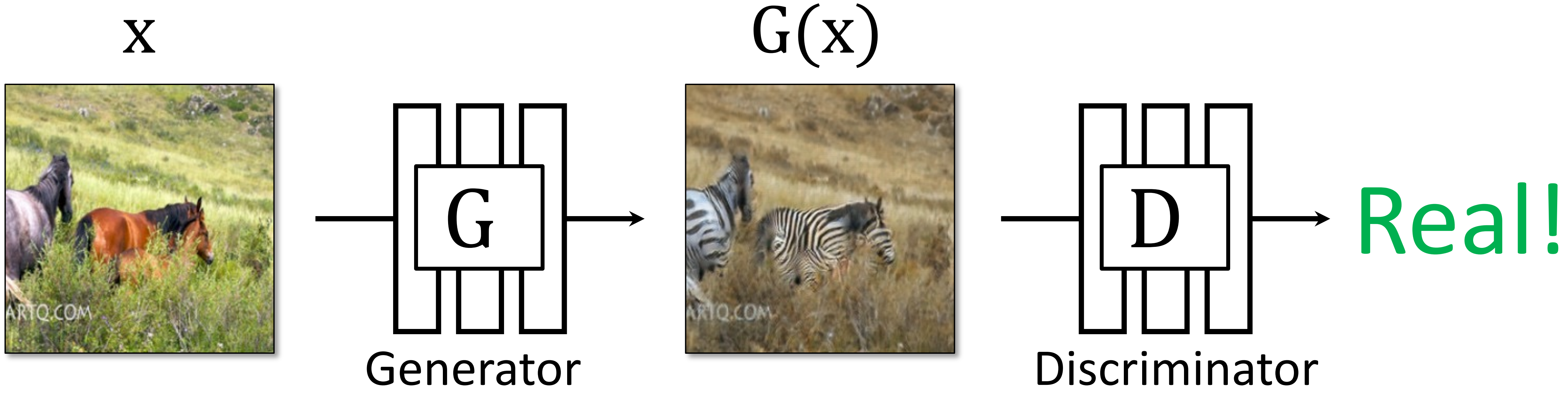
- Expensive to collect pairs.
- Impossible in many scenarios.

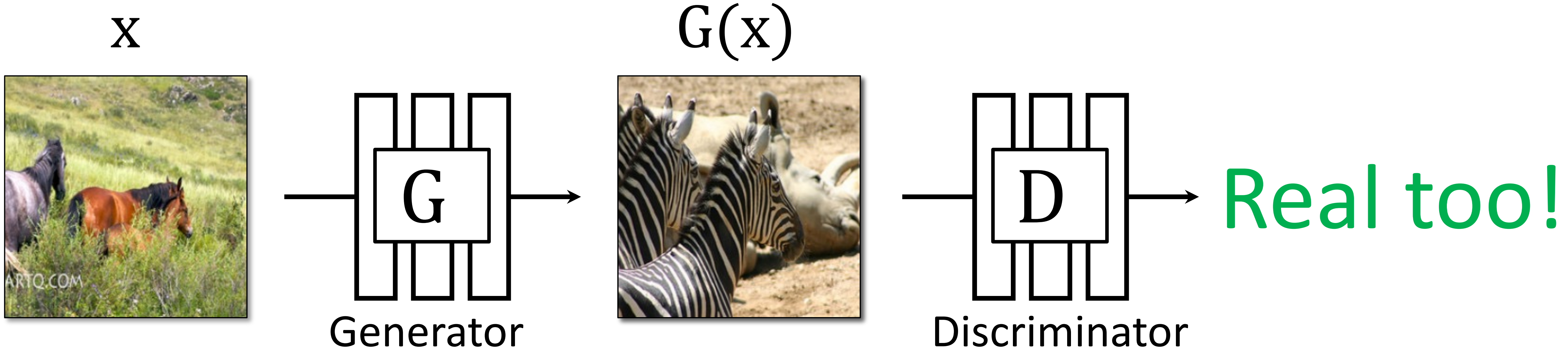


Paired
Unpaired
x_i
y_i
X
Y

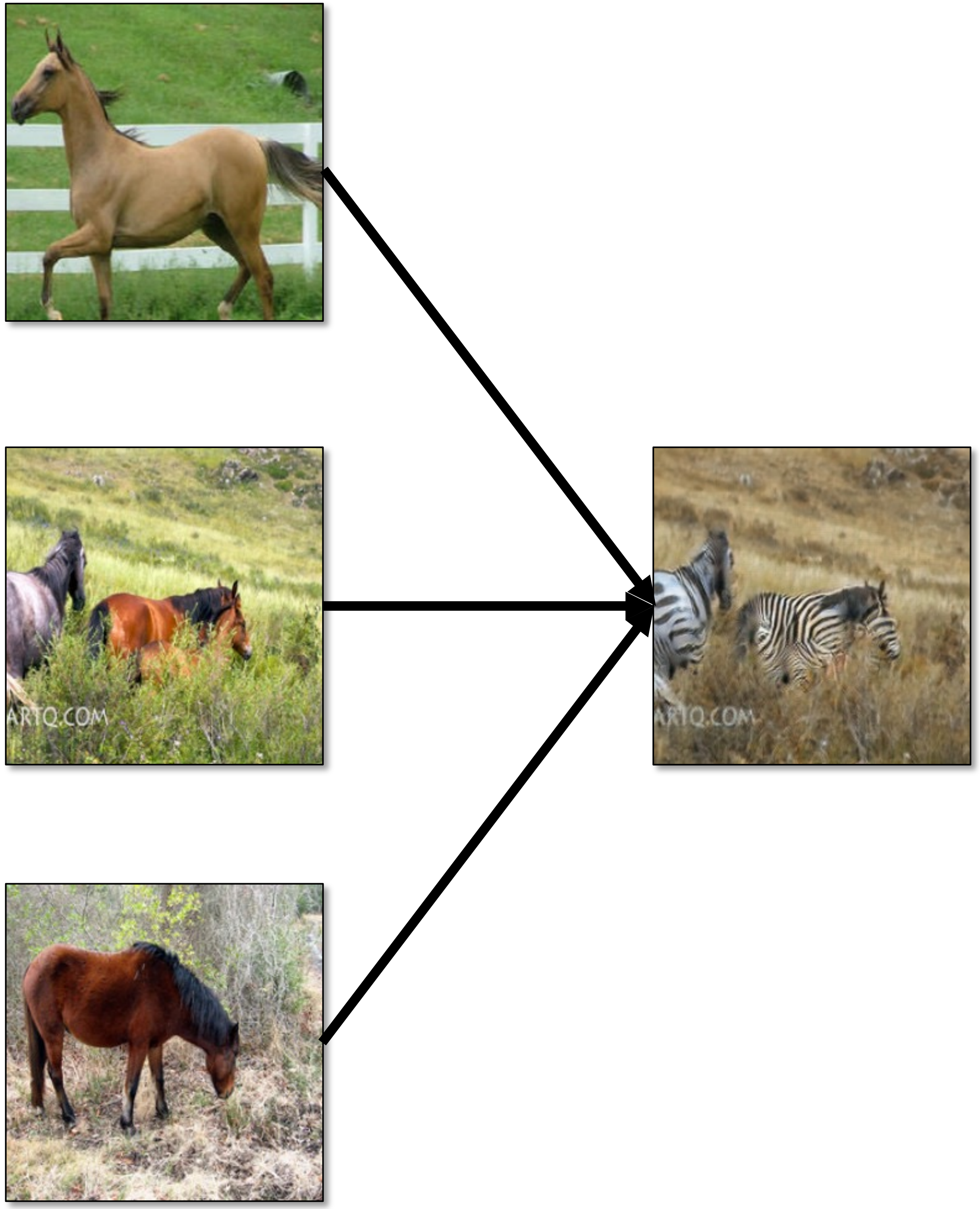


No input-output pairs!


x
G
Generator
G(x)
D
Discriminator
Real!

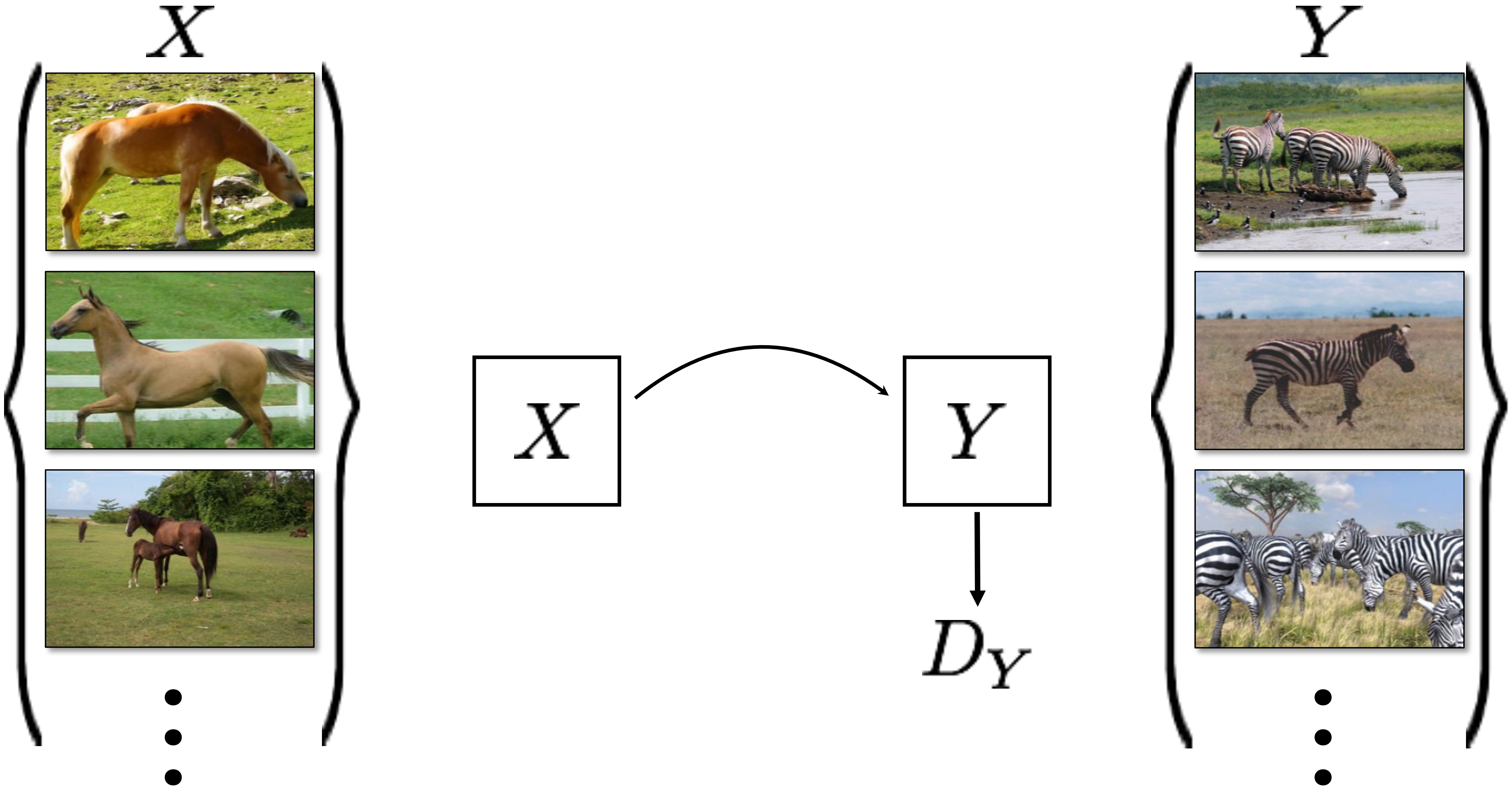


GANs do **not** force output to correspond to input



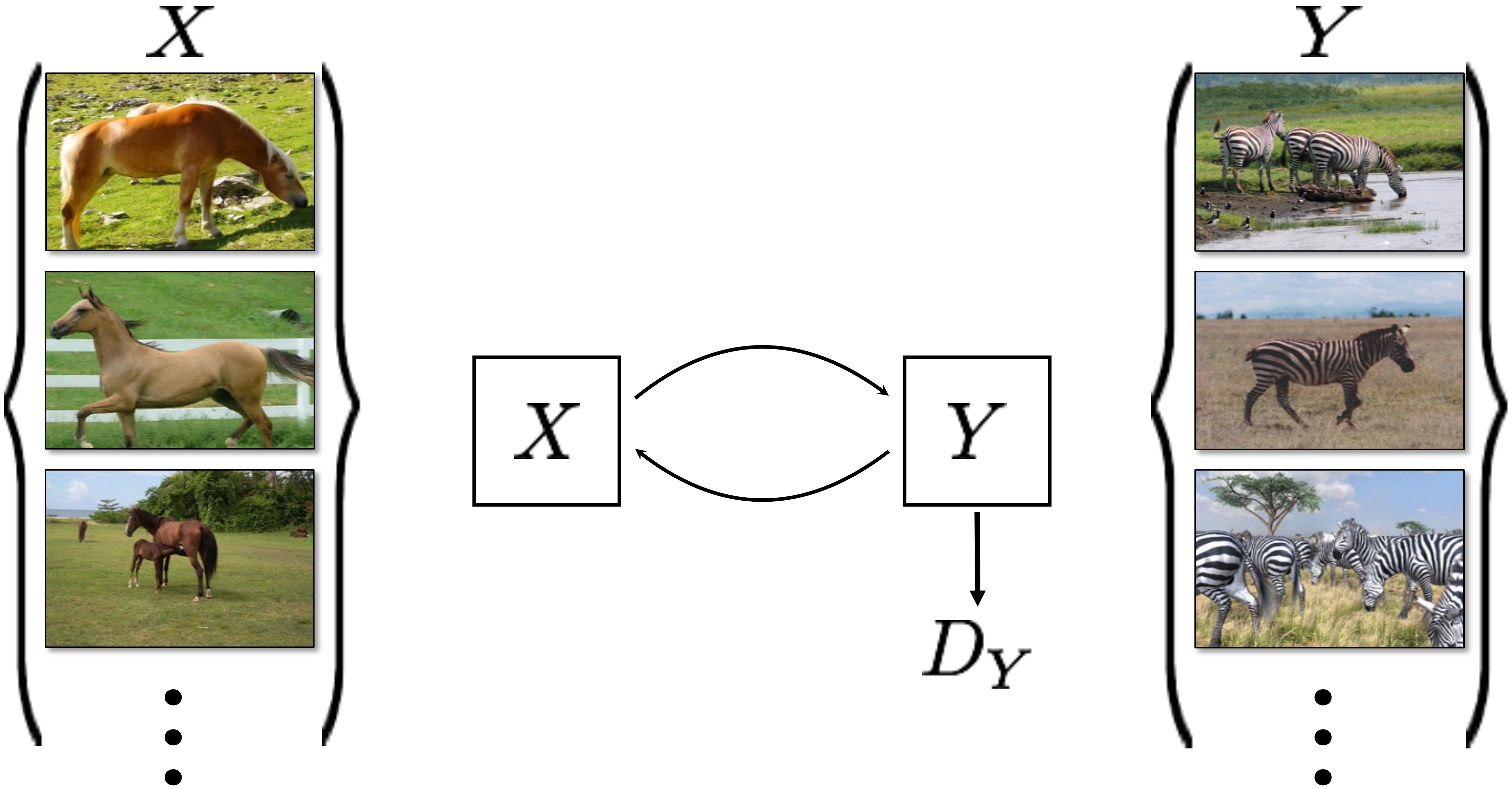
mode collapse!

Cycle-Consistent Adversarial Networks



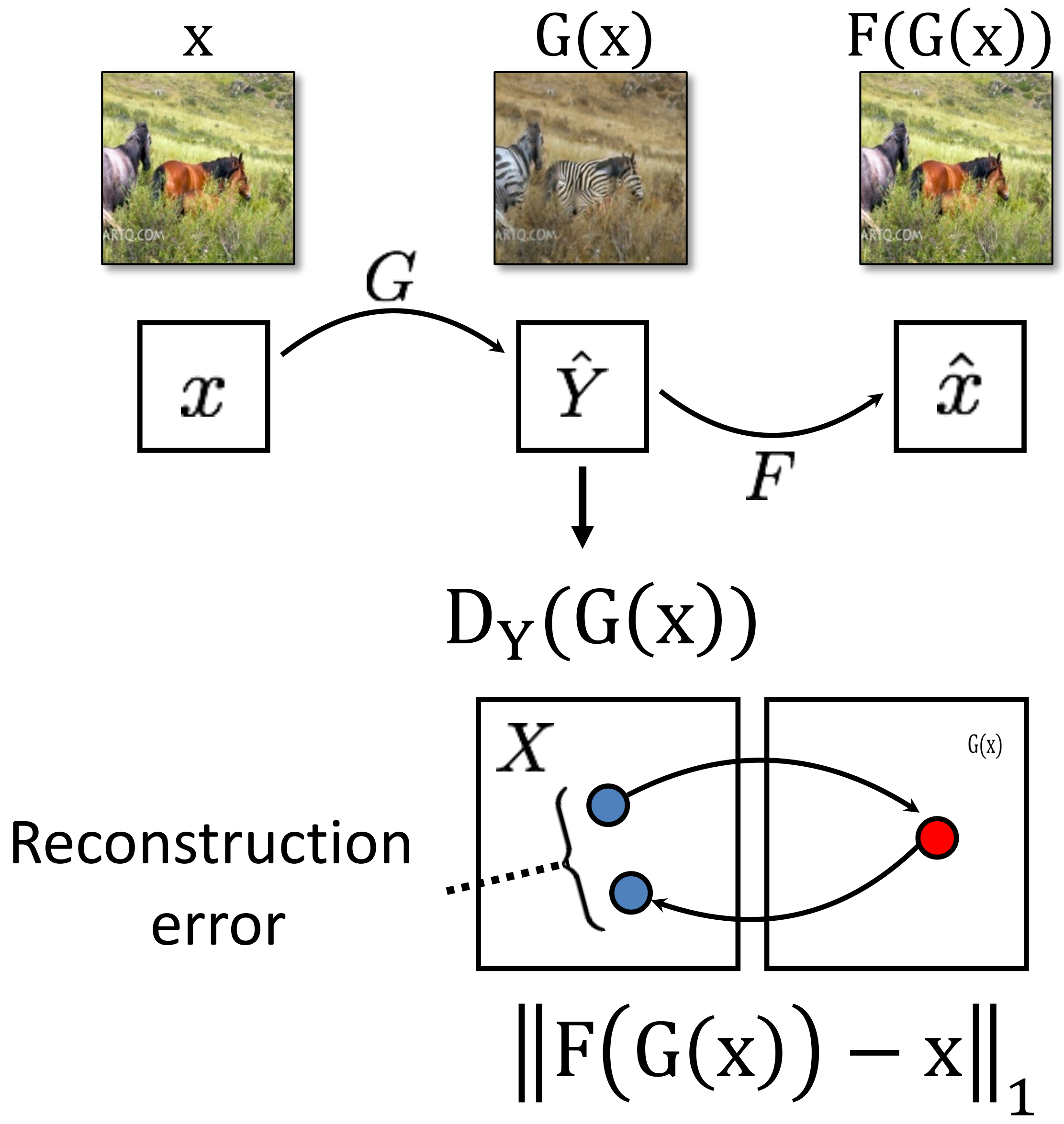
[Zhu*, Park*, Isola, and Efros, ICCV 2017]

Cycle-Consistent Adversarial Networks



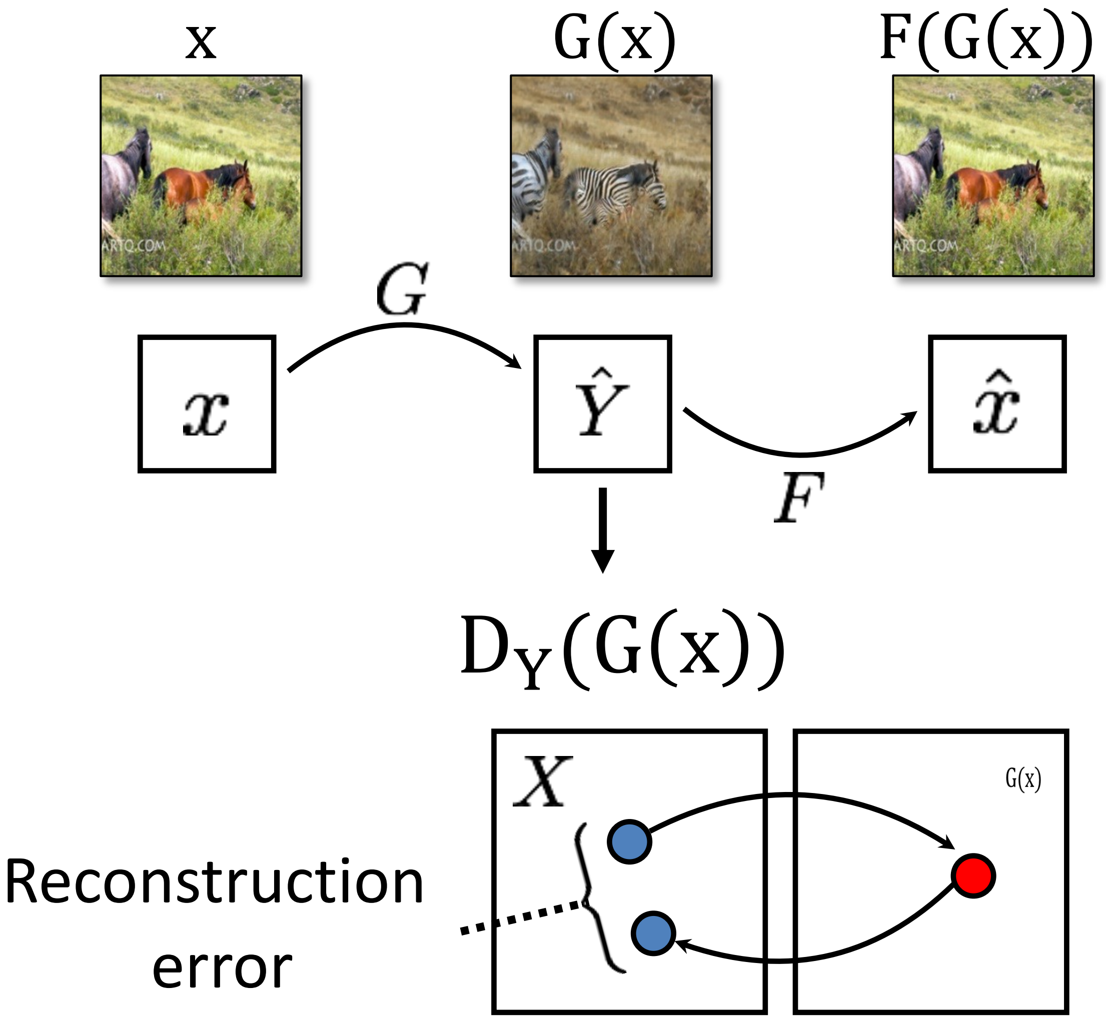
[Zhu*, Park*, Isola, and Efros, ICCV 2017]

Cycle-Consistent Adversarial Networks

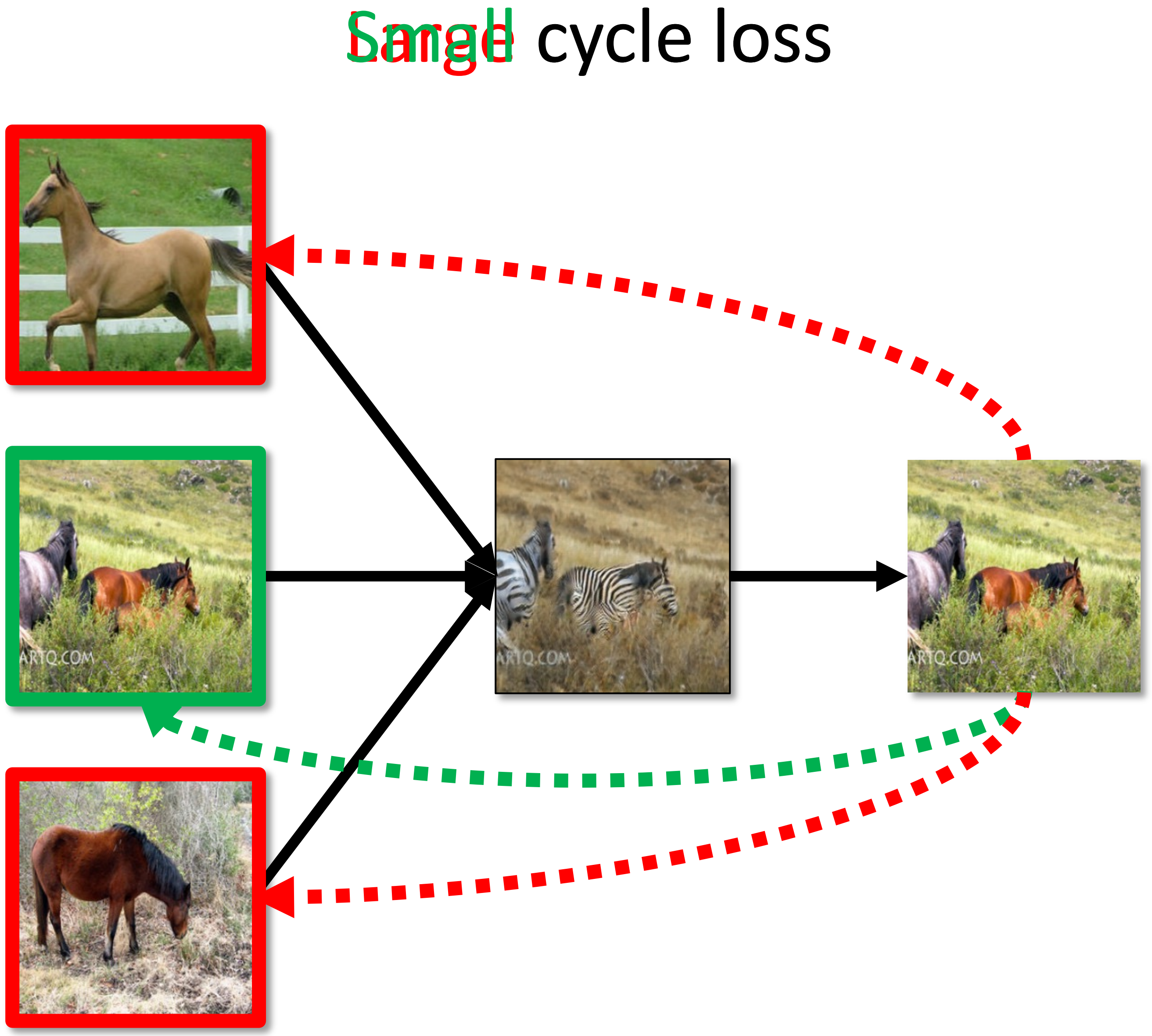


[Zhu*, Park*, Isola, and Efros, ICCV 2017]

Cycle Consistency Loss

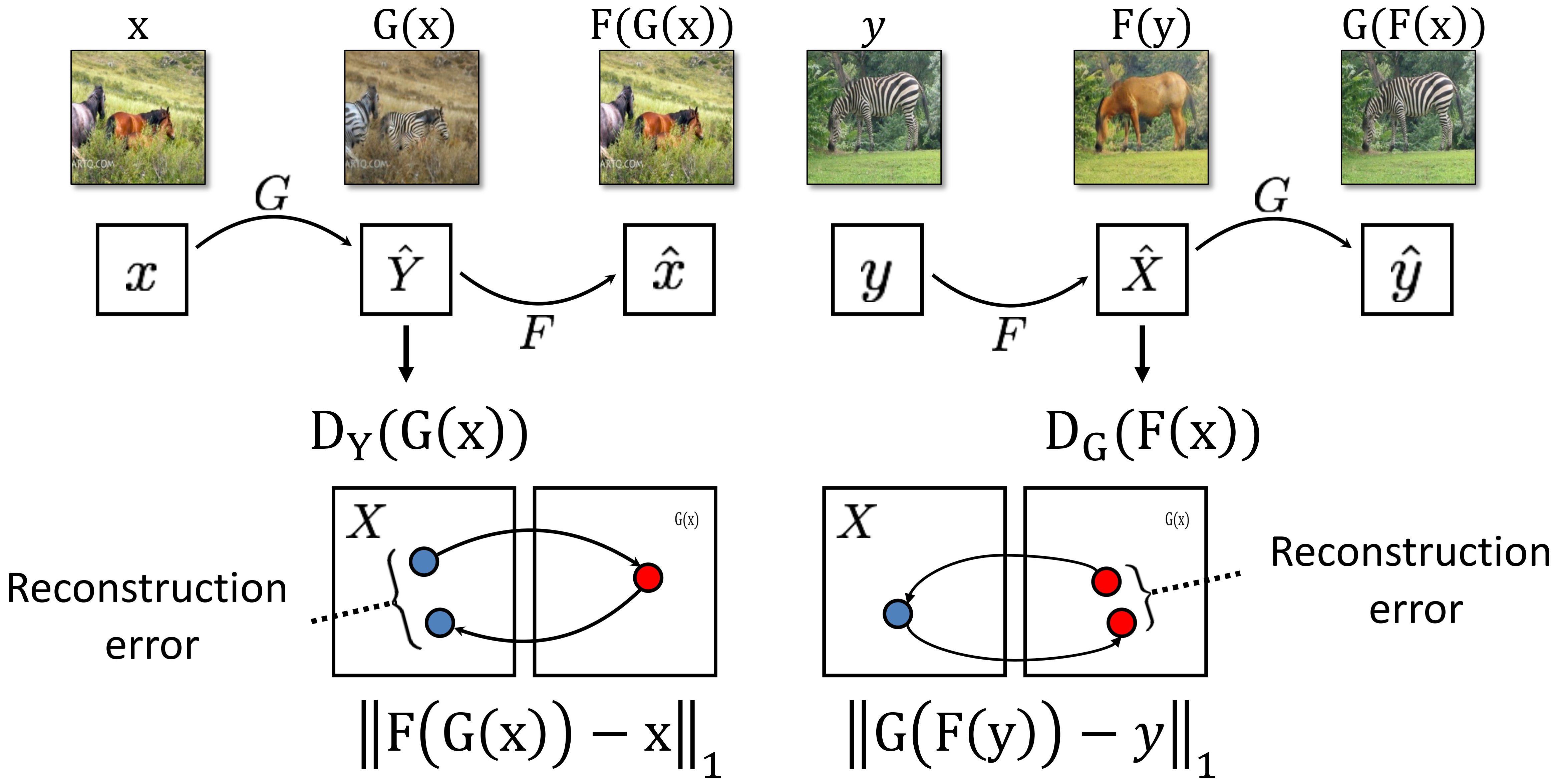


$$\|F(G(x)) - x\|_1$$



[Zhu*, Park*, Isola, and Efros, ICCV 2017]

Cycle Consistency Loss



See similar formulations [Yi et al. 2017], [Kim et al. 2017]   [Zhu*, Park*, Isola, and Efros, ICCV 2017]

Results


Collection Style Transfer

Photograph
@ Alexei Efros

Monet

Van Gogh

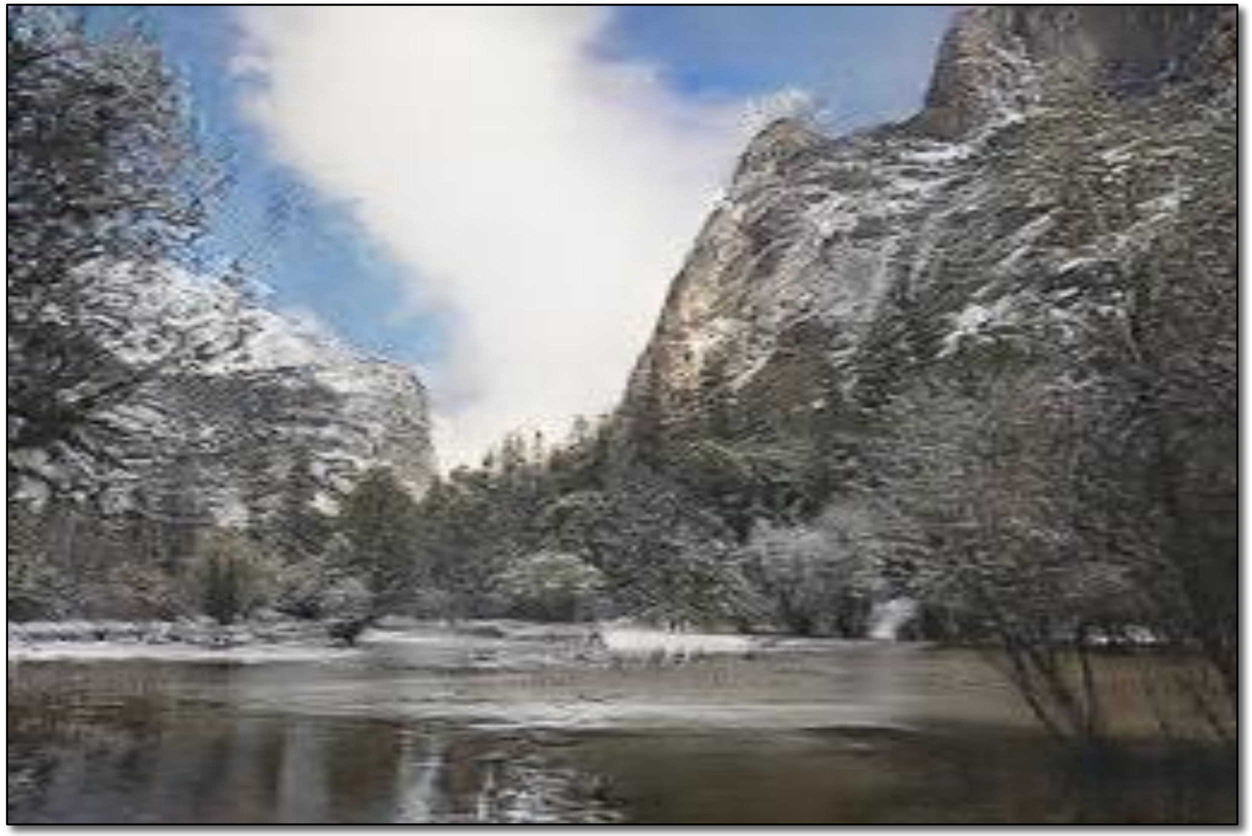
Cezanne

Ukiyo-e


Input
Monet
Van Gogh
Cezanne
Ukiyo-e

Monet's paintings → photos

Monet's paintings → photos

Loss	Map → Photo % Turkers labeled *real*	Photo → Map % Turkers labeled *real*
CoGAN [30]	0.6% ± 0.5%	0.9% ± 0.5%
BiGAN/ALI [8, 6]	2.1% ± 1.0%	1.9% ± 0.9%
SimGAN [45]	0.7% ± 0.5%	2.6% ± 1.1%
Feature loss + GAN	1.2% ± 0.6%	0.3% ± 0.2%
CycleGAN (ours)	**26.8% ± 2.8%**	**23.2% ± 3.4%**

AMT 'real vs fake' test on maps ↔ aerial

Loss	Per-pixel acc.	Per-class acc.	Class IOU
CoGAN [30]	0.40	0.10	0.06
BiGAN/ALI [8, 6]	0.19	0.06	0.02
SimGAN [45]	0.20	0.10	0.04
Feature loss + GAN	0.06	0.04	0.01
CycleGAN (ours)	**0.52**	**0.17**	**0.11**

FCN scores on cityscapes labels→ photos

Loss	Per-pixel acc.	Per-class acc.	Class IOU
CoGAN [30]	0.45	0.11	0.08
BiGAN/ALI [8, 6]	0.41	0.13	0.07
SimGAN [45]	0.47	0.11	0.07
Feature loss + GAN	0.50	0.10	0.06
CycleGAN (ours)	**0.58**	**0.22**	**0.16**

Classification performance of photo→labels

CycleGAN implementations

PyTorch

pytorch-CycleGAN-and-pix2pix

Image-to-image translation in PyTorch (e.g., horse2zebra, edges2cats, and more)

Python ★ 4.3k 970

Torch

CycleGAN

Software that can generate photos from paintings, turn horses into zebras, perform style transfer, and more.

Lua ★ 6.5k 940

20+ implementations by researchers/developers:

- Tensorflow, Chainer, mxnet, Lasagne, Keras…

Generative Modeling approaches

- Autoencoders
- Auto-regressive models
- GANs
- Diffusion model

Denoising Diffusion-based Generative Modeling: Foundations and Applications

Karsten Kreis

Ruiqi Gao

Arash Vahdat

NVIDIA

CVPR JUNE 19-24 2022 NEW ORLEANS LOUISIANA

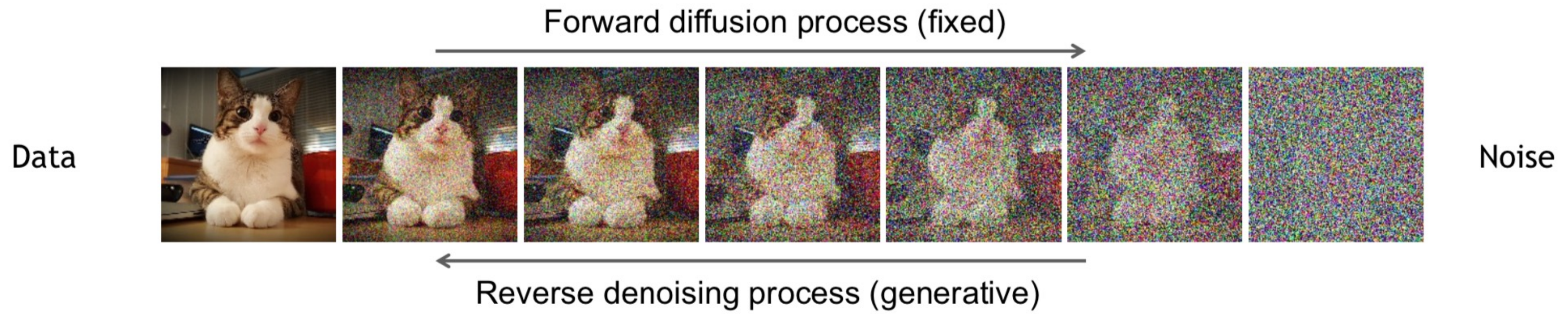
Google AI

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

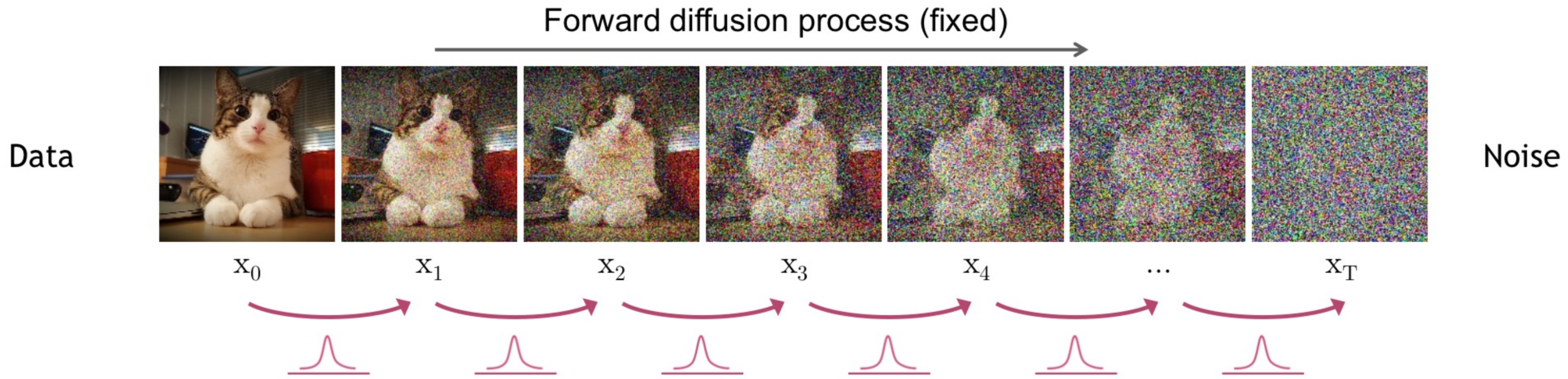


Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

[Kreis, Gao & Vahdat, CVPR22]

Forward Diffusion Process

The formal definition of the forward process in T steps:

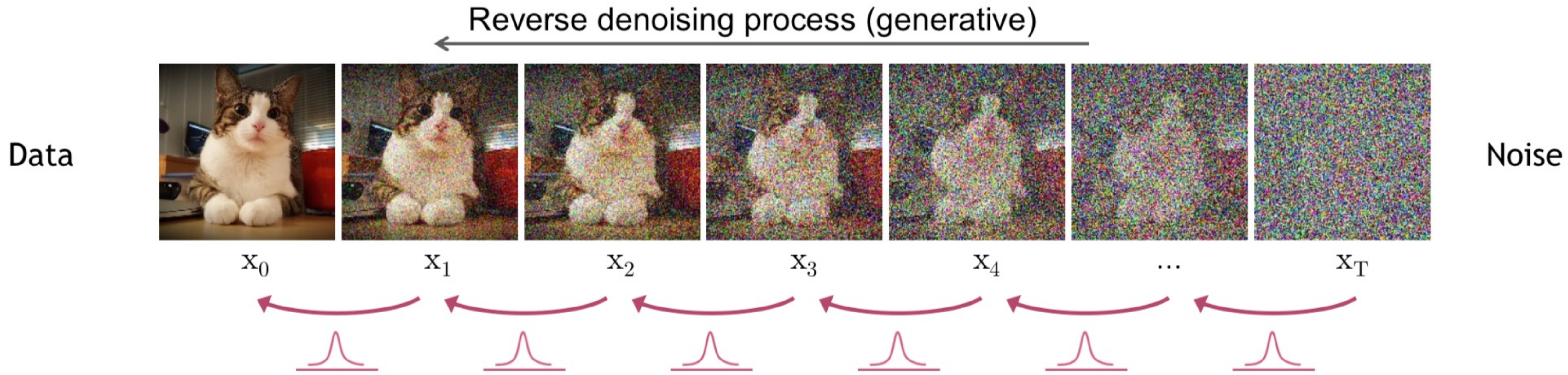


$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad \text{(joint)}$$

[Kreis, Gao & Vahdat, CVPR22]

Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



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

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

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

[Kreis, Gao & Vahdat, CVPR22]

Learning Denoising Model

Variational upper bound

For training, we can form variational upper bound that is commonly used for training variational autoencoders:

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] =: L$$

Sohl-Dickstein et al. ICML 2015 and Ho et al. NeurIPS 2020 show that:

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

where $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ is the tractable posterior distribution:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0), \tilde{\beta}_t\mathbf{I}),$$

$$\text{where } \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

[Kreis, Gao & Vahdat, CVPR22]

Parameterizing the Denoising Model

Since both $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are Normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}||\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)||^2\right] + C$$

Recall that $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$. Ho et al. NeurIPS 2020 observe that:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

They propose to represent the mean of the denoising model using a *noise-prediction* network:

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

With this parameterization

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})}\left[\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon}_{\mathbf{x}_t}, t)||^2\right] + C$$

Training Objective Weighting

Trading likelihood for perceptual quality

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})} \Bigg[\underbrace{\frac{\beta_t^2}{2\sigma_t^2 (1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, t)||^2 \Bigg]$$

The time dependent λ_t ensures that the training objective is weighted properly for the maximum data likelihood training.

However, this weight is often very large for small t's.

Ho et al. NeurIPS 2020 observe that simply setting $\lambda_t = 1$ improves sample quality. So, they propose to use:

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I}), t \sim \mathcal{U}(1,T)} \Big[||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon}_{\mathbf{x}_t}, t)||^2 \Big]$$

For more advanced weighting see Choi et al., Perception Prioritized Training of Diffusion Models, CVPR 2022.

Summary

Training and Sample Generation

Algorithm 1 Training

repeat

  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$

  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$

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

  Take gradient descent step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\boxed{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}, t\right) \right\|^2$$

until converged

Algorithm 2 Sampling

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

for $t = T, \dots, 1$ **do**

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

$$\mathbf{x}_{t-1} = \boxed{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)} + \sigma_t \mathbf{z}$$

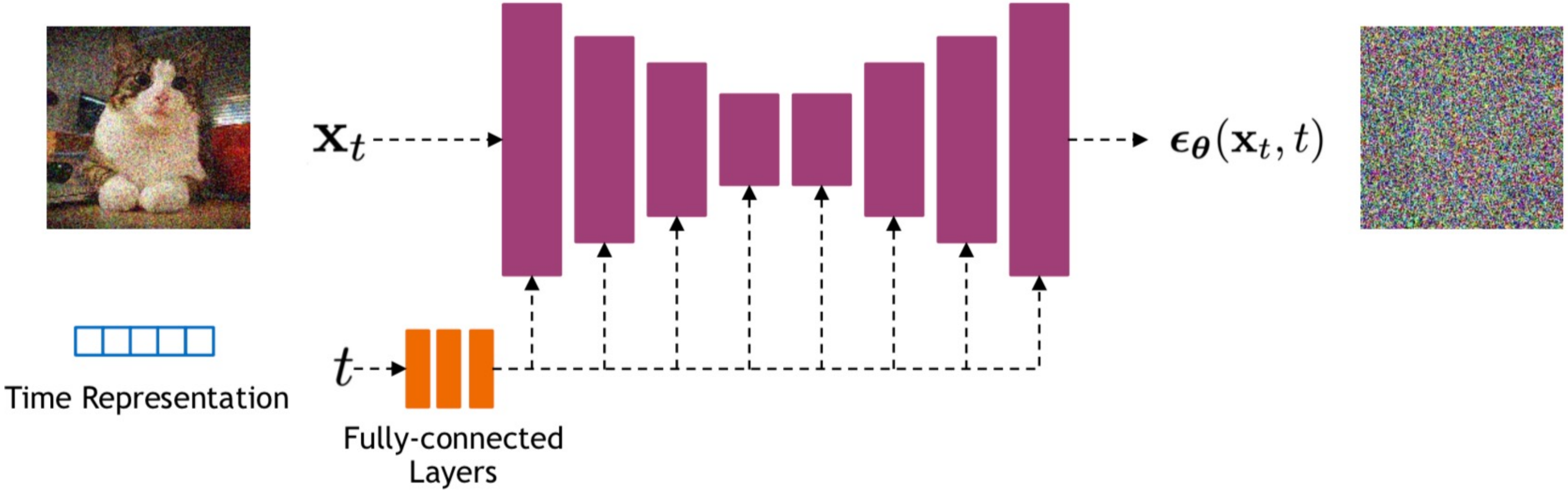
end for

return $\mathbf{x}_0$

Implementation Considerations

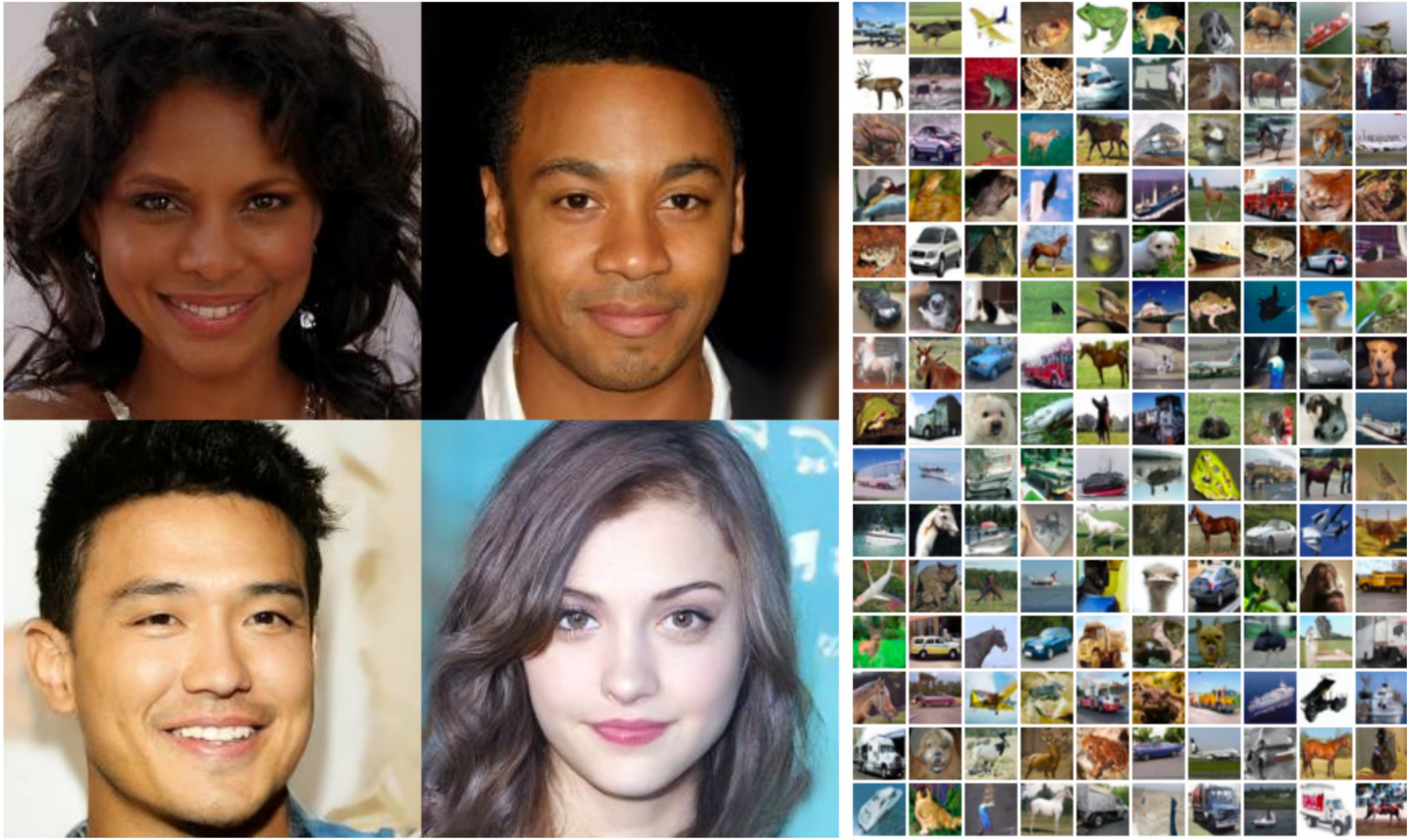
Network Architectures

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



Time representation: sinusoidal positional embeddings or random Fourier features.

Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see [Dharivwal and Nichol NeurIPS 2021](#))

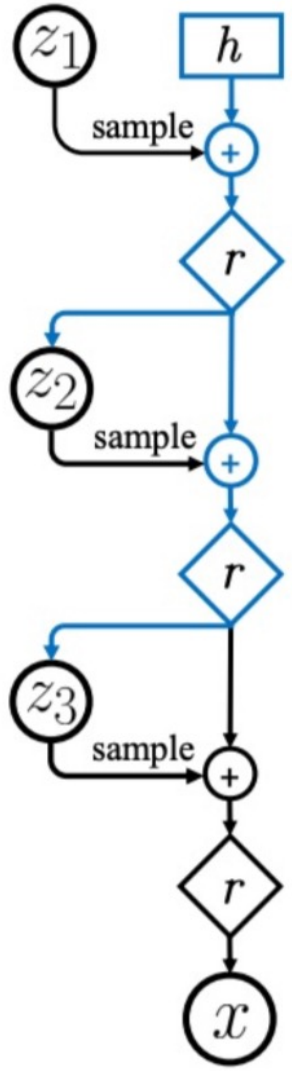


Connection to VAEs

Diffusion models can be considered as a special form of hierarchical VAEs.

However, in diffusion models:

- The encoder is fixed
- The latent variables have the same dimension as the data
- The denoising model is shared across different timestep
- The model is trained with some reweighting of the variational bound.



[Vahdat and Kautz, NVAE: A Deep Hierarchical Variational Autoencoder, NeurIPS 2020](#)
[Sønderby, et al.. Ladder variational autoencoders, NeurIPS 2016.](#)

Comparison of Generative Models

Method	Key idea	Pros	Cons
Variational Autoencoders (VAE)	Encoder + Decoder, trained by max lower-bound on LLH	Easy inference/sampling LLH available	Blurry samples
Generative Adversarial Networks (GAN)	Generator + Discriminator trained by adversarial game	High-quality generations	Unstable training; no direct LLH
Autoregressive Models (e.g. NLM)	Factored representation of joint into product of per-dimension conditionals	Simple; good LLH numbers	Inefficient sampling
Diffusion Models (e.g. DALLE-2)	Sequence of incremental denoising operations	Current SOTA; simple	Inefficient sampling Continuous data only

References

Beutel et al. *Data decisions and theoretical implications when adversarially learning fair representations*. arXiv:1707.00075, 2017.

Denton and Birodkar. *Unsupervised Learning of Disentangled Representations from Video*. NIPS, 2017.

Donahue et al. *Adversarial Feature Learning*. ICLR, 2017.

Dumoulin et al. *Adversarially Learned Inference.* ICLR, 2017

Edwards & Storkey. *Censoring Representations with an Adversary*. ICLR, 2016.

Ganin and Lempitsky. *Unsupervised domain adaptation by backpropagation.* ICML, 2015.

Kim and Mnih. *Disentangling by Factorising*. ICML, 2018.

Madras et al. *Learning Adversarially Fair and Transferable Representations*. ICML, 2018.

Makhzani et al. *Adversarial Autoencoders*. ICLR Workshop, 2016.

Mescheder et al. *Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks*. ICML, 2017.

Schmidhuber. *Learning factorial codes by predictability minimization.* Neural Computation, 1992.

Tzeng et al. S*imultaneous deep transfer across domains and tasks.* ICCV, 2015.

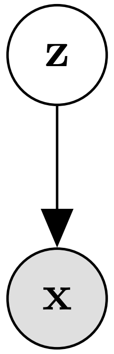
Tzeng et al. *Adversarial discriminative domain adaptation.* CVPR, 2017.

Villegas, et al. D*ecomposing motion and content for natural video sequence prediction.* In ICLR, 2017.

Zhang et al. M*itigating Unwanted Biases with Adversarial Learning*. AIES, 2018.

More detailed Varational Auto-encoder derivation

Directed graphical models



- We assume data is generated by:

$$z \sim p(z) \qquad x \sim p(x|z)$$

- z is latent/hidden x is observed (image)
- Use θ to denote parameters of the generative model

Parameter estimation

- Given dataset $\{x_1, ..., x_n\}$, maximize likelihood of data under model:

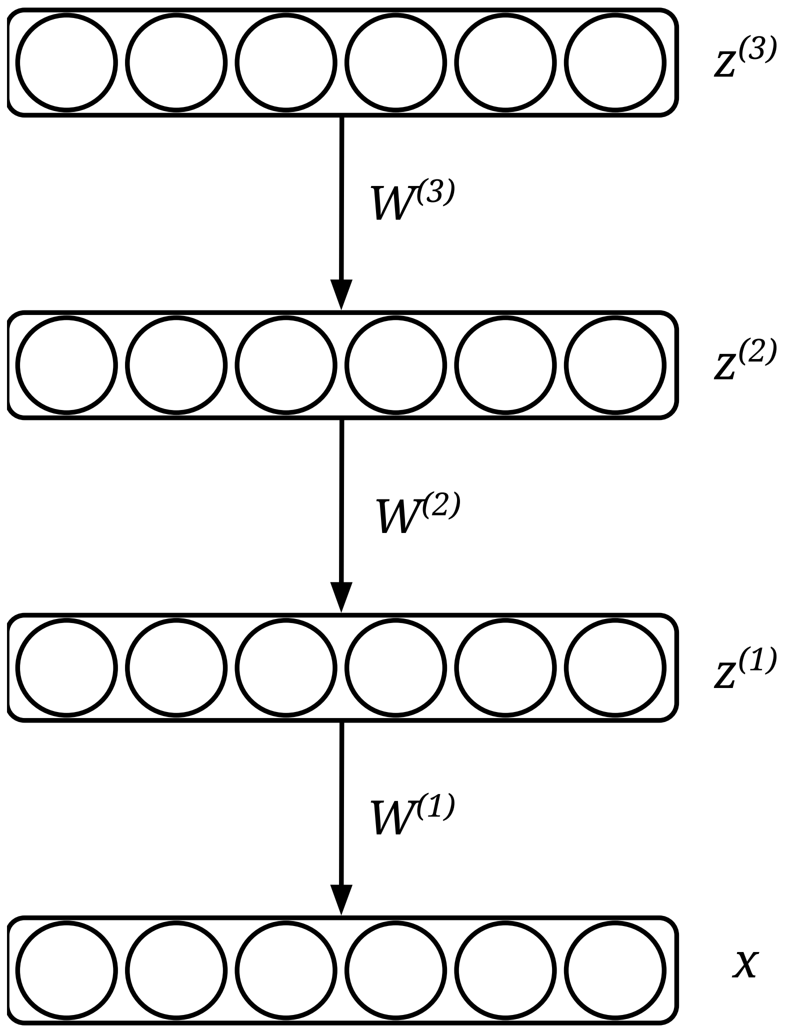
$$\max_\theta \sum_{i=1}^{n} \log p(x_i;\theta) = \max_\theta \sum_{i=1}^{n} \sum_z \log p(x_i, z;\theta)$$

- This quantity often intractable, difficult to optimize directly

- Can be optimized with iterative Expectation Maximization (EM) algorithm
  - Fix parameters and compute log likelihood wrt $p(z|x;\theta^t)$
  - Fix z find parameters $\theta^{(t+1)}$ to maximize log likelihood

Parameter estimation

- Standard EM requires access to posterior $p(z|x)$
- For the deep neural net models we care about this is infeasible
- Solution: introduce *variational* approximation $q(z;\phi)$ to $p(z|x)$
- Will give bound on log likelihood



Bounding the marginal likelihood

Recall Jenson's inequality: When f is concave, $f(\mathbb{E}[x]) \geq \mathbb{E}[f(x)]$

$$\begin{aligned}
\log p(x) &= \log \int_z p(x,z) \\
&= \log \int_z q(z) \frac{p(x,z)}{q(z)} \\
&\geq \int_z q(z) \log \frac{p(x,z)}{q(z)} = L(x;\theta,\phi) \qquad \text{(by Jensons inequality)} \\
&= \int_z q(z) \log p(x,z) - \int_z q(z) \log q(z) \\
&= \underbrace{\mathbb{E}_{q(z)}[\log p(x,z)]}_{\text{Expectation of joint distribution}} + \underbrace{\mathrm{H}(q(z))}_{\text{Entropy}}
\end{aligned}$$

Learning directed graphical models

- Maximize bound on likelihood of data:

$$\max_{\theta} \sum_{i=1}^{N} \log p(x_i; \theta) \geq \max_{\theta, \phi_1, \ldots, \phi_N} \sum_{i=1}^{N} L(x_i; \theta, \phi_i)$$

- Historically, used different ϕ_i for every data point
  - But we'll move away from this soon..
- Can still use EM style algorithm to iteratively optimize
- For more info see Blei *et al.* (2003)

New method of learning: approximate inference model

- Instead of having different variational parameters for each data point, fit a conditional parametric function
- The output of this function will be the parameters of the variational distribution $q(z|x)$
- Instead of q(z) we have $q_\phi(z|x)$

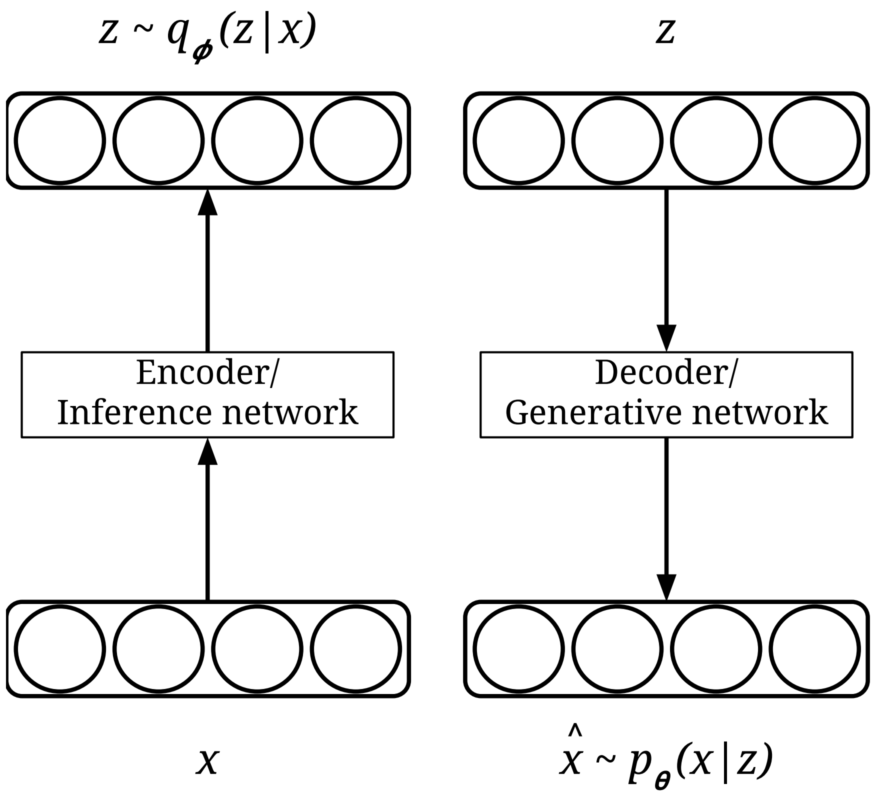
Evidence Lower BOund (ELBO)

- ELBO becomes:

$$L(x;\theta,\phi) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x,z)]}_{\text{Expectation of joint distribution}} + \underbrace{\mathrm{H}(q_\phi(z|x))}_{\text{Entropy}}$$

Variational autoencoder

- *Encoder* network maps from image space to latent space
  - Outputs parameters of $q_\phi(z|x)$
- *Decoder* maps from latent space back into image space
  - Outputs parameters of $p_\theta(x|z)$

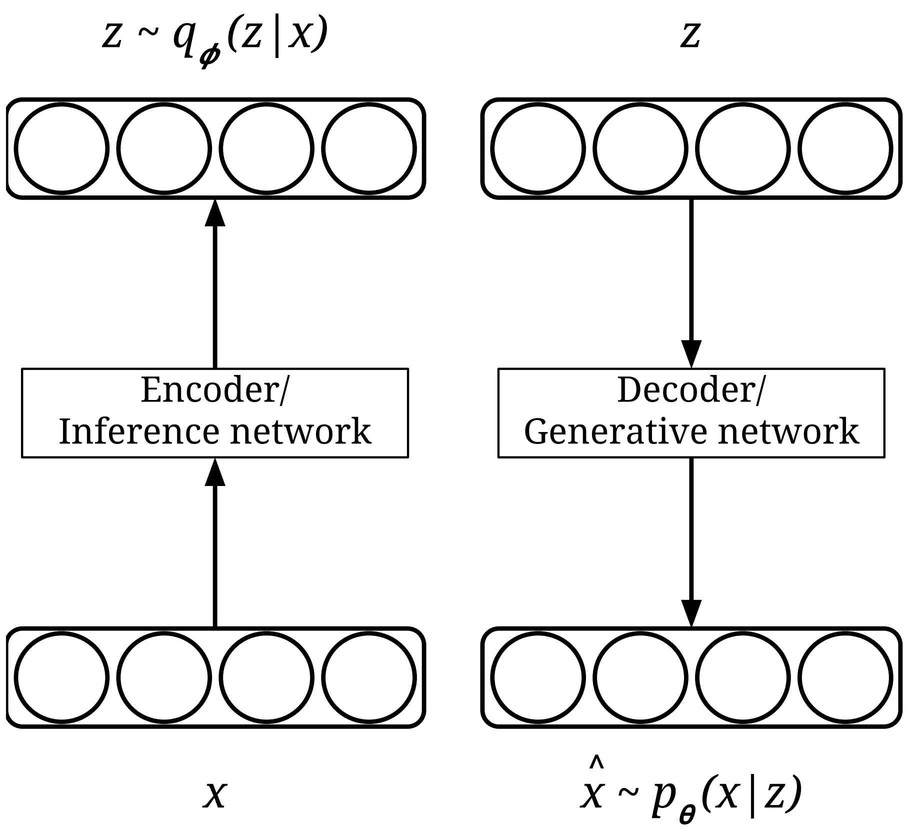


[Kingma & Welling (2013)]

Example

- *Encoder* network outputs mean and variance of Normal distribution
    - $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$

- *Decoder* network outputs mean (and optionally variance) of Normal distribution
    - $p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \mathbf{I})$



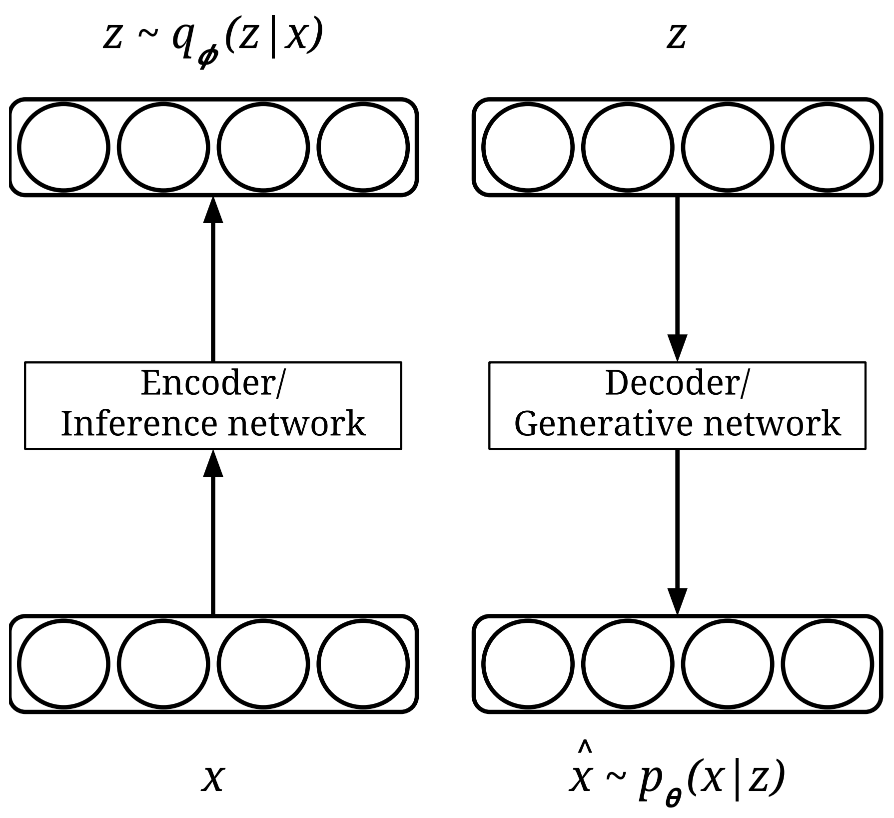
[Kingma & Welling (2013)]

- Rearranging the ELBO:

$$
\begin{aligned}
L(x;\theta,\phi) &= \int_z q(z|x)\log\frac{p(x,z)}{q(z|x)} \\
&= \int_z q(z|x)\log\frac{p(x|z)p(z)}{q(z|x)} \\
&= \int_z q(z|x)\log p(x|z) + \int_z q(z|x)\log\frac{p(z)}{q(z|x)} \\
&= \mathbb{E}_{q(z|x)}\log p(x|z) - \mathbb{E}_{q(z|x)}\log\frac{q(z|x)}{p(z)} \\
&= \underbrace{\mathbb{E}_{q(z|x)}\log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}
\end{aligned}
$$

Variational autoencoder

- Inference network outputs parameters of $q_\phi(z|x)$
- Generative network outputs parameters of $p_\theta(x|z)$
- Optimize θ and ϕ jointly by maximizing ELBO:



$$L(x;\theta,\phi) = \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}$$

Stochastic gradient variation bayes (SGVB) estimator

- Reparameterization trick : re-parameterize $z \sim q_\phi(z|x)$ as

$$z = g_\phi(x, \epsilon) \text{ with } \epsilon \sim p(\epsilon)$$

- For example, with a Gaussian can write $z \sim \mathcal{N}(\mu, \sigma^2)$ as

$$z = \mu + \epsilon\sigma^2 \text{ with } \epsilon \sim \mathcal{N}(0, 1)$$

[Kingma & Welling (2013); Rezende *et al.* (2014)]

Stochastic gradient variation bayes (SGVB) estimator

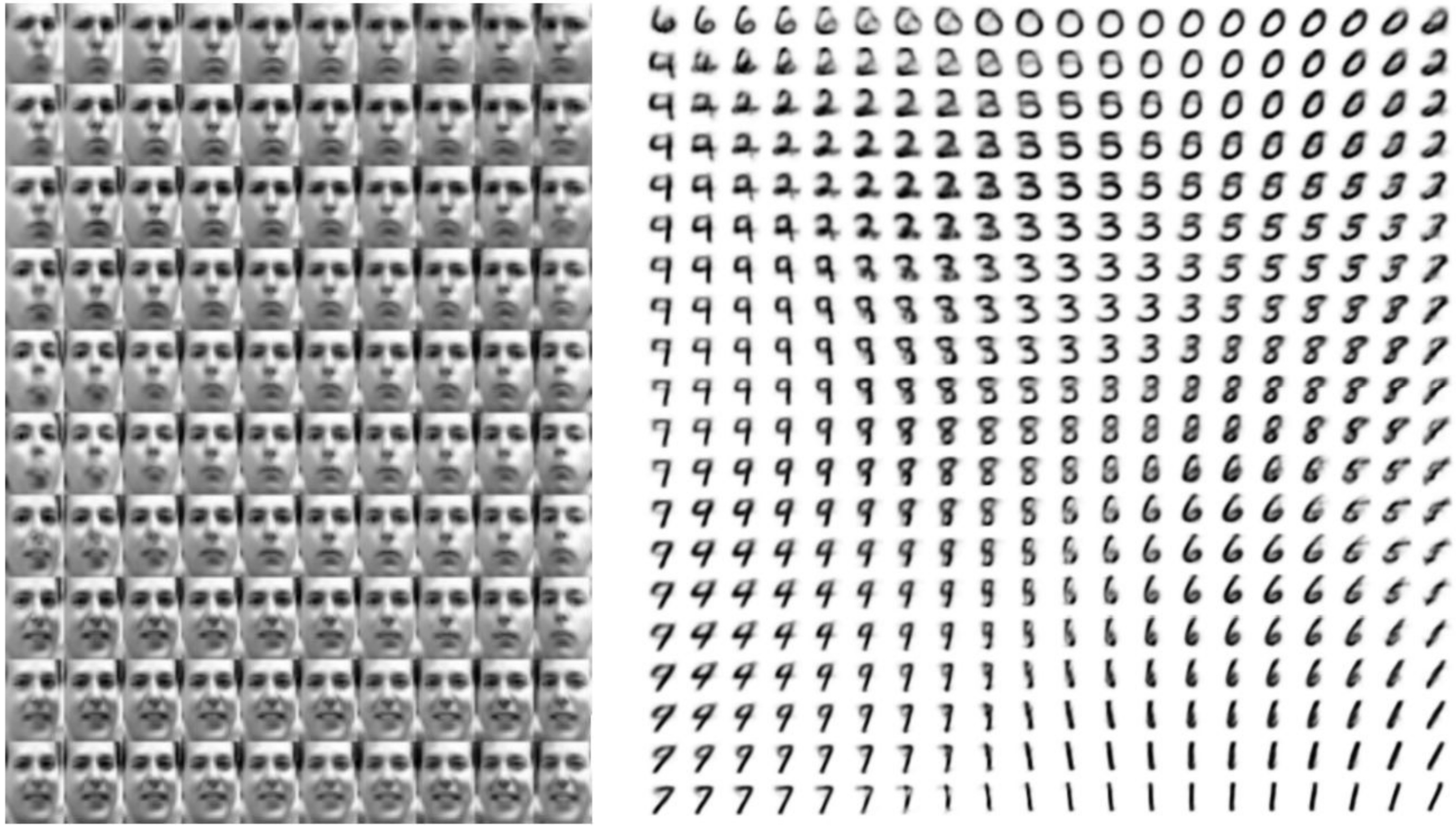
$$L(x;\theta,\phi) = \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}$$

- Using reparameterization trick we form Monte Carlo estimate of reconstruction term:

$$\begin{aligned} \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) &= \mathbb{E}_{p(\epsilon)} \log p_\theta(x|g_\phi(x,\epsilon)) \\ &\simeq \frac{1}{L}\sum_{i=1}^{L} \log p_\theta(x|g_\phi(x,\epsilon)) \qquad \text{where } \epsilon \sim p(\epsilon) \end{aligned}$$

- KL divergence term can often be computed analytically (eg. Gaussian)

VAE learned manifold



[Kingma & Welling (2013)]

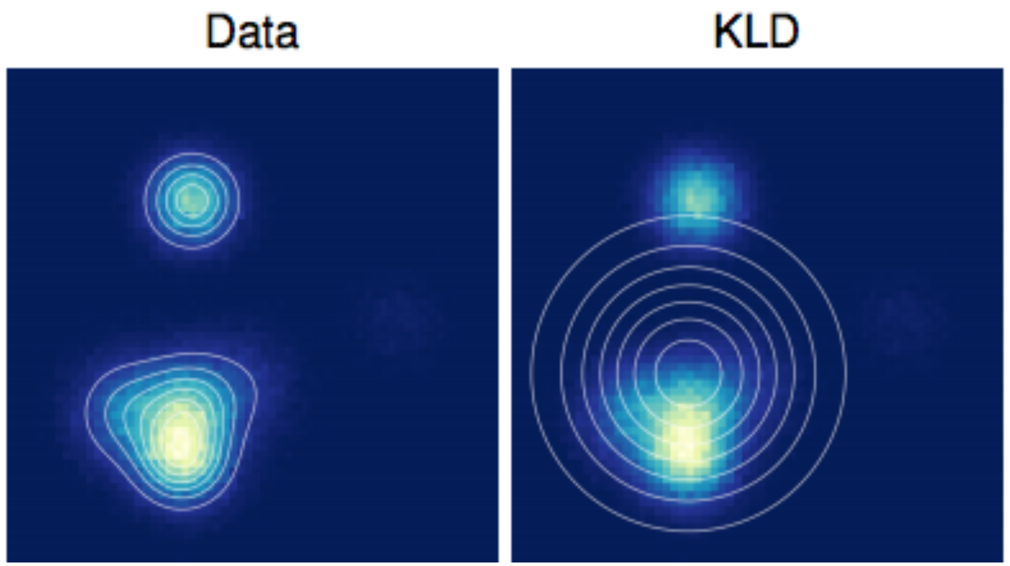
VAE samples

(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

[Kingma & Welling (2013)]

VAE tradeoffs

- Pros:
  - Theoretically pleasing
  - Optimizes bound on likelihood
  - Easy to implement
- Cons:
  - Samples tend to be blurry
    - Maximum likelihood minimizes $D_{KL}(p_{data}||p_{model})$



[Theis *et al.* (2016)]