Advanced Deep Learning

DATA.ML.230

Unsupervised learning
and
Generative Adversarial Networks

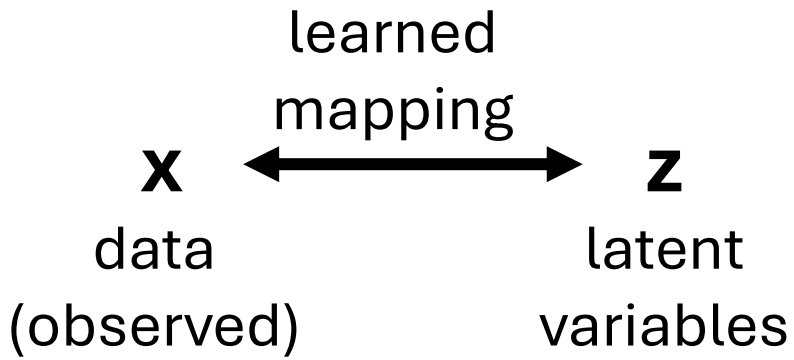
From: Simon J. D. Prince, Chapters 14 & 15 – Understanding Deep Learning, MIT Press (19 May 2025)

Unsupervised learning

- Unsupervised learning models are learned using a set of data $\{\mathbf{x}_i\}$ (no labels)
- Diverse set of unsupervised learning models:
  - Data generation to follow similar properties as those in the training set
  - Data denoising
  - Data compression
  - Data interpolation
  - Data grouping and prototypes calculation

Unsupervised learning

- Common strategy:



- Latent variables usually have a lower dimension than the original data

Unsupervised learning

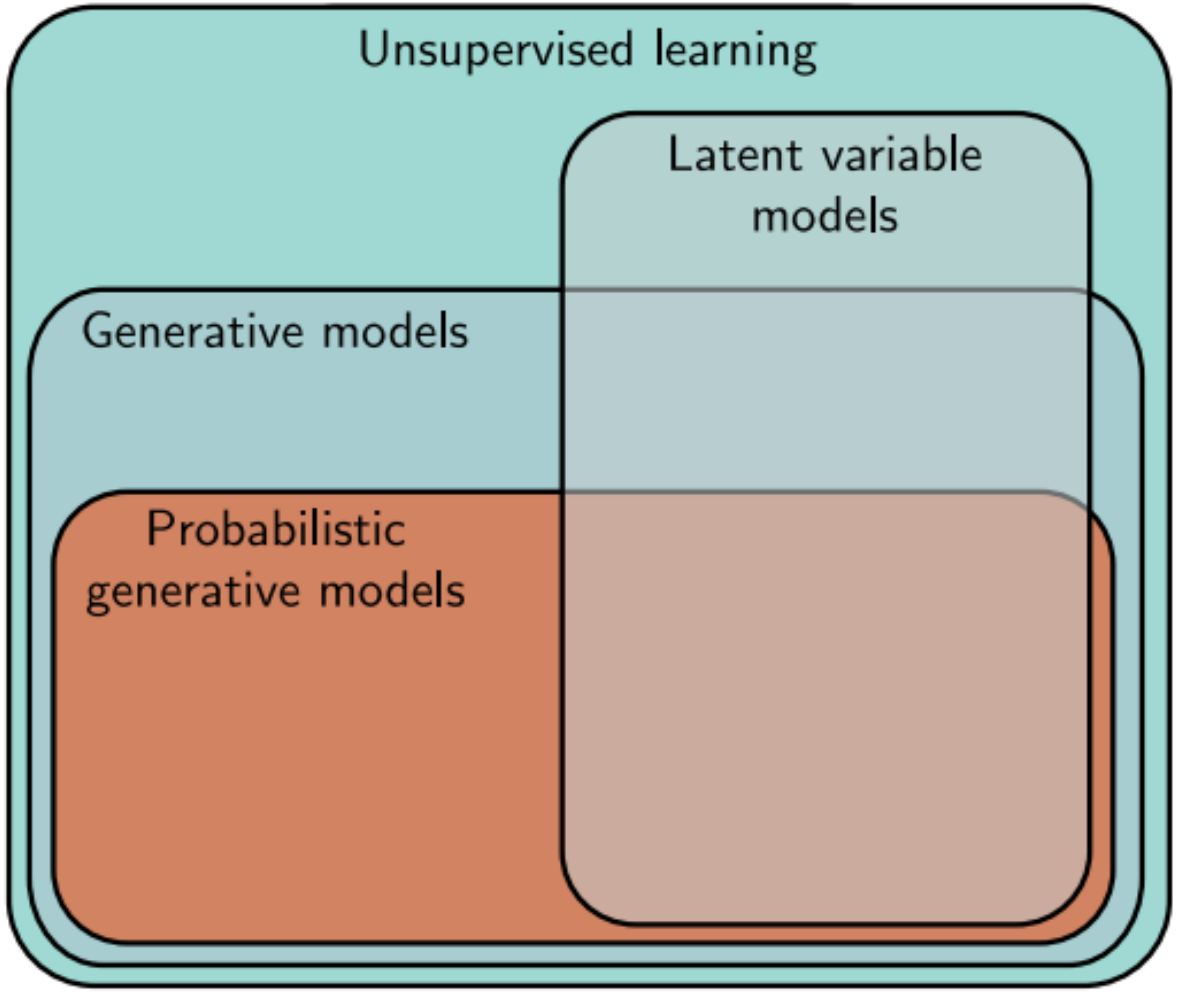
- Common strategy:

learned mapping

x → **z**

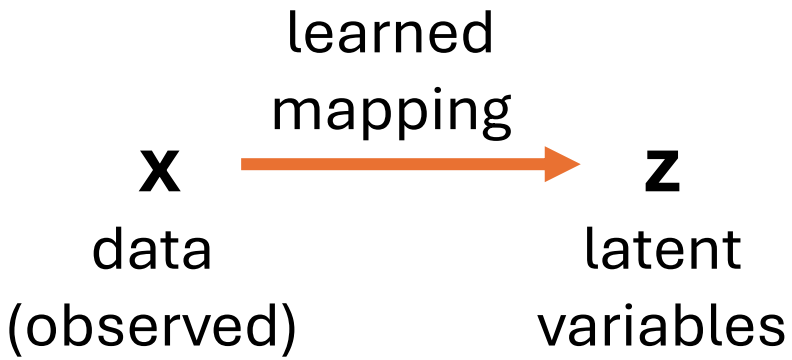
data (observed) → latent variables

- Latent variables usually have a lower dimension than the original data



Unsupervised learning

- Common strategy:



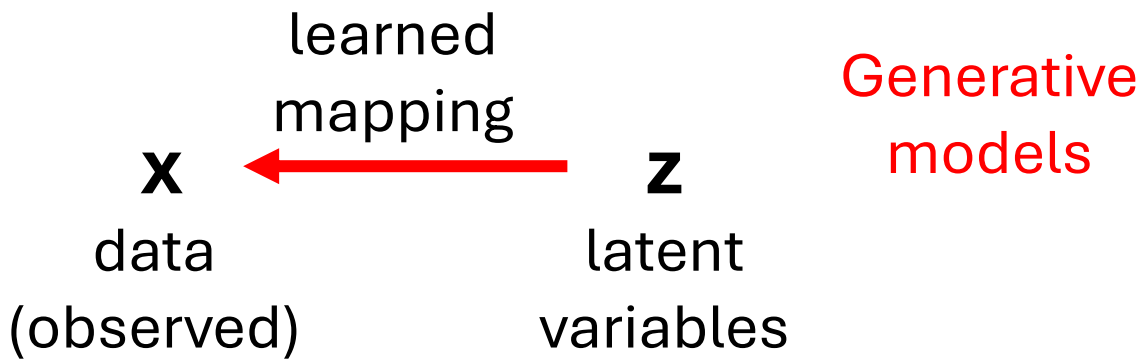
- Latent variables usually have a lower dimension than the original data

Examples:

- K-means clustering
- Principal Component Analysis
- Non-negative Matrix Factorization

Unsupervised learning

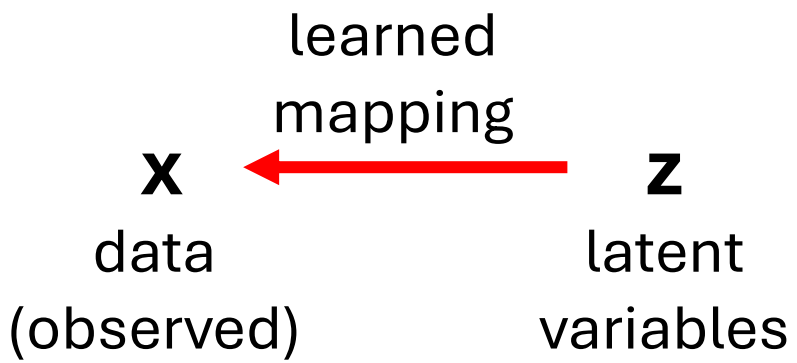
- Common strategy:



- Latent variables usually have a lower dimension than the original data

Unsupervised learning

- Common strategy:



Generative models

- Latent variables usually have a lower dimension than the original data

Examples:

- Generative Adversarial Networks
- Variational Autoencoders
- Diffusion models
- Normalizing Flows

Generative models

Desiderata:

- **Efficient sampling:** Generating samples from the model should be computationally inexpensive and take advantage of the parallelism of modern hardware

Generative models

Desiderata:

- **Efficient sampling**
- **High-quality sampling:** The samples should be indistinguishable from the real data with which the model was trained

Generative models

Desiderata:

- **Efficient sampling**
- **High-quality sampling**
- **Coverage:** Samples should represent the entire training distribution. It is insufficient to generate samples that all look like a subset of the train

Generative models

Desiderata:

- **Efficient sampling**
- **High-quality sampling**
- **Coverage**
- **Well-behaved latent space:** Every latent variable **z** corresponds to a plausible data example **x**. Smooth changes in **z** correspond to smooth changes in **x**

Generative models

Desiderata:

- **Efficient sampling**
- **High-quality sampling**
- **Coverage**
- **Well-behaved latent space**
- **Disentangled latent space:** Manipulating each dimension of **z** should correspond to changing an interpretable property of the data. For example, in a model of language, it might change the topic, tense, or verbosity

Generative models

Desiderata:

- **Efficient sampling**
- **High-quality sampling**
- **Coverage**
- **Well-behaved latent space**
- **Disentangled latent space**
- **Efficient likelihood computation:** If the model is probabilistic, we would like to be able to calculate the probability of new examples efficiently and accurately

Generative models

Desiderata:

Model	Efficient	Sample quality	Coverage	Well-behaved latent space	Disentangled latent space	Efficient likelihood
GANs	✓	✓	✗	✓	?	n/a
VAEs	✓	✗	?	✓	?	✗
Flows	✓	✗	?	✓	?	✓
Diffusion	✗	✓	?	✗	✗	✗

Figure 14.3 Properties of four generative models. Neither generative adversarial networks (GANs), variational autoencoders (VAEs), normalizing flows (Flows), nor diffusion models (diffusion) have the full complement of desirable properties.

Measuring performance of generative models

Test likelihood: Measure the likelihood of the data in the test set to belong to the distribution learned by the model.

Can be used only for probabilistic generative models

Measuring performance of generative models

Test likelihood: Measure the likelihood of the data in the test set to belong to the distribution learned by the model.

Inception score: Use of a pre-trained classification model (usually the “Inception” model).

Can be used only for images

Measuring performance of generative models

Test likelihood: Measure the likelihood of the data in the test set to belong to the distribution learned by the model.

Inception score: Use of a pre-trained classification model (usually the “Inception” model).

Can be used only for images

Two criteria:

- Each generated image **x*** should look like only one of the 1000 classes y in the ImageNet database

Measuring performance of generative models

Test likelihood: Measure the likelihood of the data in the test set to belong to the distribution learned by the model.

Inception score: Use of a pre-trained classification model (usually the "Inception" model).

Can be used only for images

Two criteria:

1. Each generated image **x*** should look like only one of the 1000 classes y in the ImageNet database

$Pr(y|\mathbf{x}_i^*)$ should be highly peaked

Measuring performance of generative models

Test likelihood: Measure the likelihood of the data in the test set to belong to the distribution learned by the model.

Inception score: Use of a pre-trained classification model (usually the “Inception” model).

Can be used only for images

Two criteria:

1. Each generated image **x*** should look like only one of the 1000 classes y in the ImageNet database
2. The entire set of generated images should not be concentrated to one/few classes

Measuring performance of generative models

Test likelihood: Measure the likelihood of the data in the test set to belong to the distribution learned by the model.

Inception score: Use of a pre-trained classification model (usually the "Inception" model).

Can be used only for images

Two criteria:

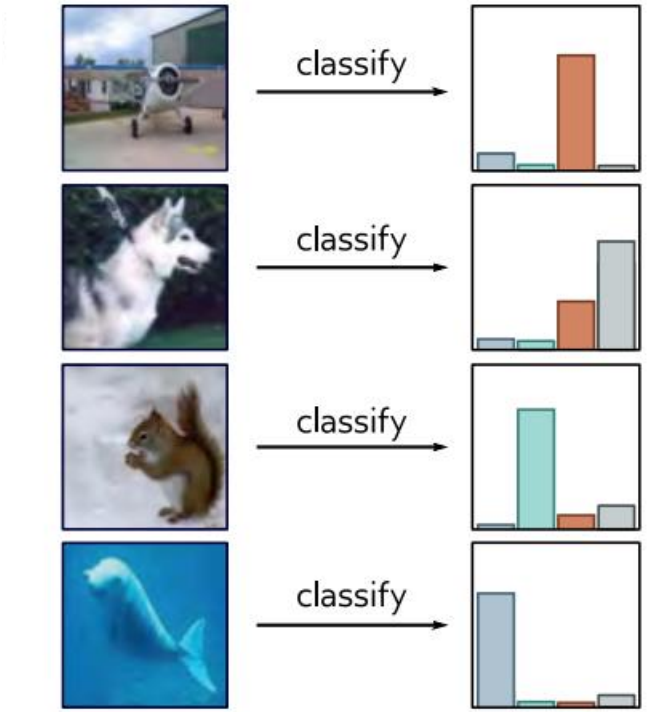
1. Each generated image **x*** should look like only one of the 1000 classes y in the ImageNet database
2. The entire set of generated images should not be concentrated to one/few classes

$Pr(y)$ should be flat when averaged over all generated examples

Measuring performance of generative models

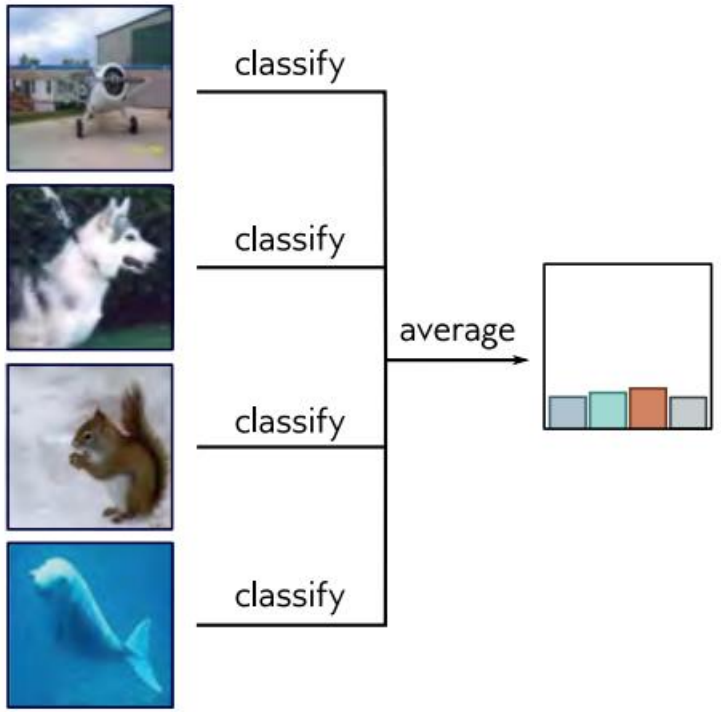
Inception score:

$$IS = \exp\left[\frac{1}{I}\sum_{i=1}^{I} D_{KL}\left[Pr(y|\mathbf{x}_i^*)||Pr(y)\right]\right]$$



Generated images $\{\mathbf{x}_i^*\}$

Classification $Pr(y|\mathbf{x}_i^*)$



Generated images $\{\mathbf{x}_i^*\}$

Marginal class distribution $Pr(y) = \frac{1}{I}\sum_{i=1}^{I} Pr(y|\mathbf{x}_i^*)$

$$Pr(y) = \frac{1}{I}\sum_{i=1}^{I} Pr(y|\mathbf{x}_i^*)$$

Measuring performance of generative models

Fréchet Inception distance: Computes a symmetric distance between the distributions of generated samples and real examples.

- Approximates both real data and generated distributions by multivariate Gaussians using the activations in the deepest layer of the Inception classification network
- Estimates the distance between them using the Fréchet distance

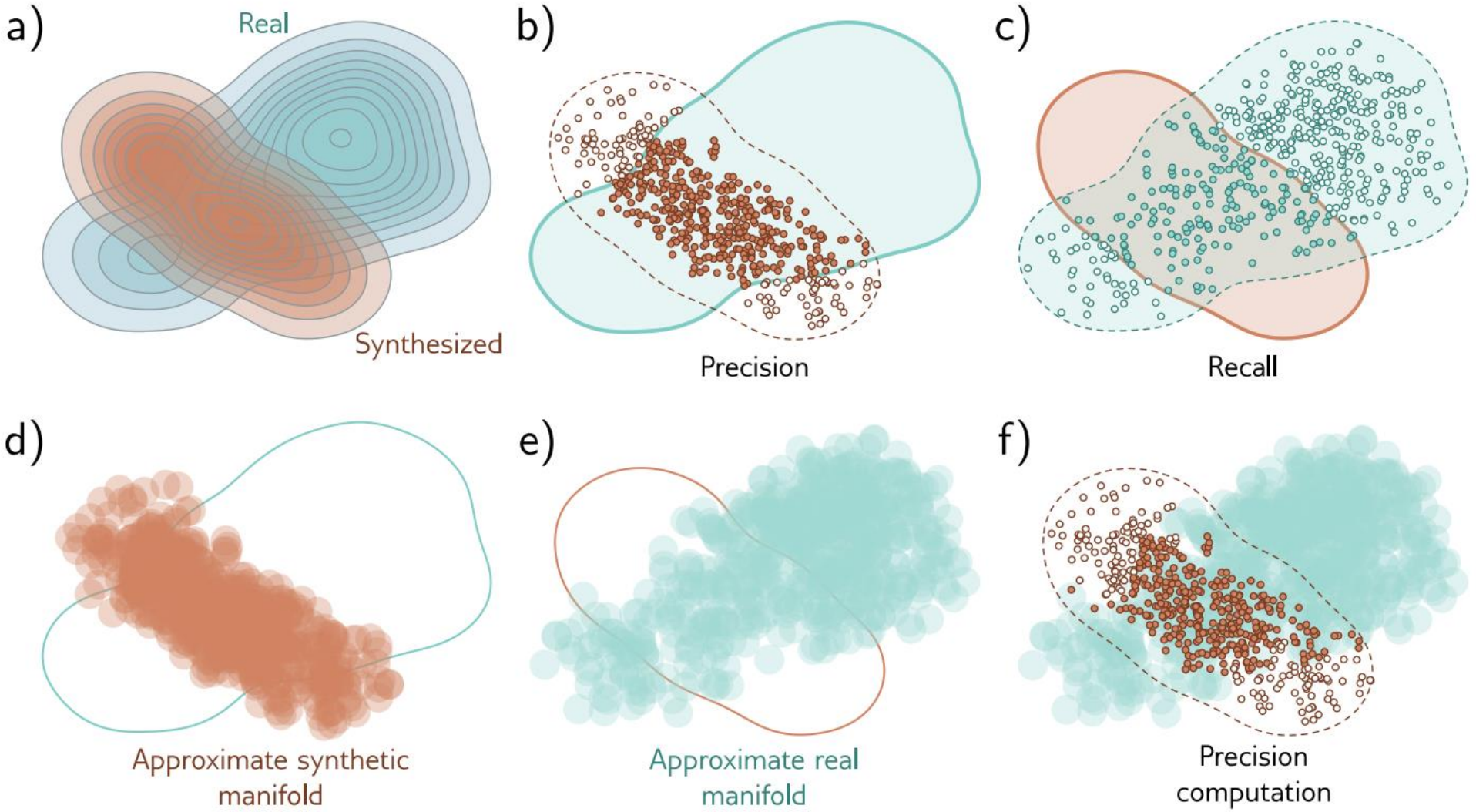
Manifold precision/recall: considers the overlap between the data manifold (subset of the data space where the real examples lie) and the model manifold (where the generated samples lie):

- Precision is the fraction of model samples that fall into the data manifold and measures the proportion of generated samples that are realistic.
- Recall is the fraction of data examples that fall within the model manifold and measures the proportion of the real data the model can generate

Measuring performance of generative models

How to determine the data manifold:

- For each real/synthetic data sample, use the distance to the k^{th} nearest neighbor to create a hypersphere centered on that sample.
- The manifold is decided to be the union of all the hyperspheres.



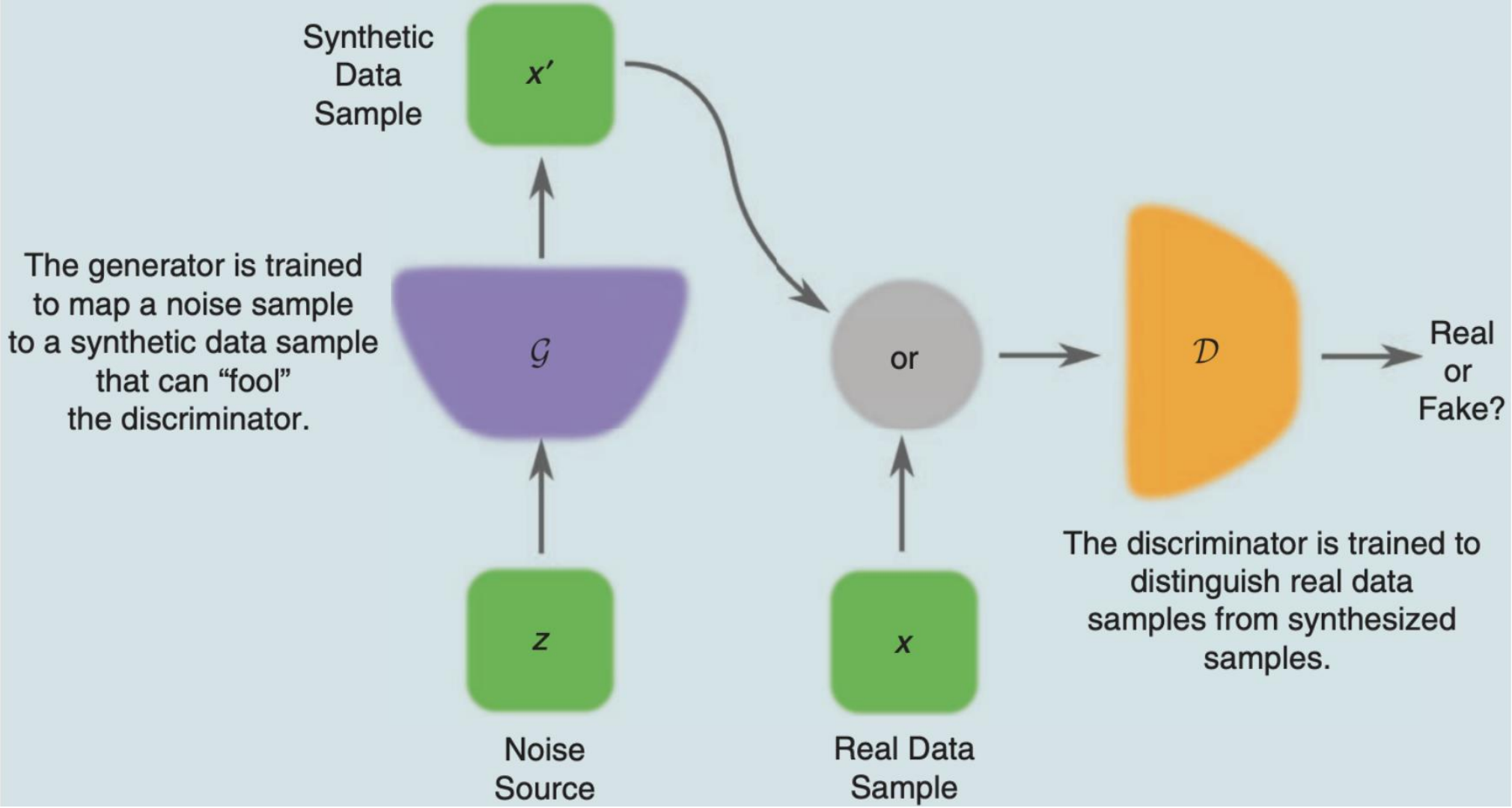
Generative Adversarial Networks

Goal: Generate new samples that are indistinguishable from a set of training examples

No probability distribution of the modeled data is learned

Mechanism/system creating new samples

Generative Adversarial Networks



Creswell, Antonia, et al. "Generative adversarial networks: An overview." IEEE Signal Processing Magazine, 2018

Generative Adversarial Networks

Two subgoals:

1. Generate new samples $\{\mathbf{x}_j^*\}$ that are drawn from the same distribution as a set of real training data $\{\mathbf{x}_i\}$:
   i. Choose a latent variable $\mathbf{z}_j$ from a simple base distribution (e.g., standard normal)
   ii. Pass $\mathbf{z}_j$ through a generator network g[•, **θ**] with parameters **θ** to generate $\mathbf{x}_j^*$ that *is similar to* the real data $\{\mathbf{x}_i\}$
2. Pass the generated samples $\{\mathbf{x}_j^*\}$ and real samples $\{\mathbf{x}_i\}$ through a discriminator network f[•, **ϕ**] with parameters **ϕ** and classify them to two classes so that y_i = real and y_j = generated

Estimate the parameters **θ** and **ϕ** until the generated data can fool the discriminator network

Generative Adversarial Networks

Example 1D

Generator network

$$x_j^* = g[z_j, \theta] = z_j + \theta$$

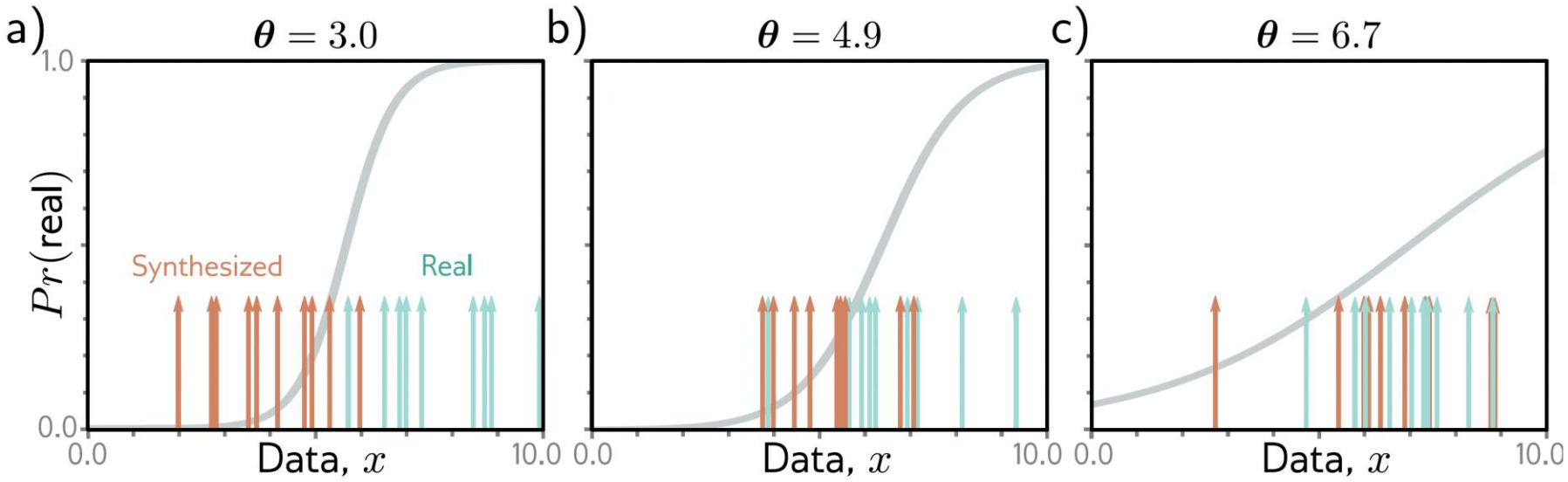
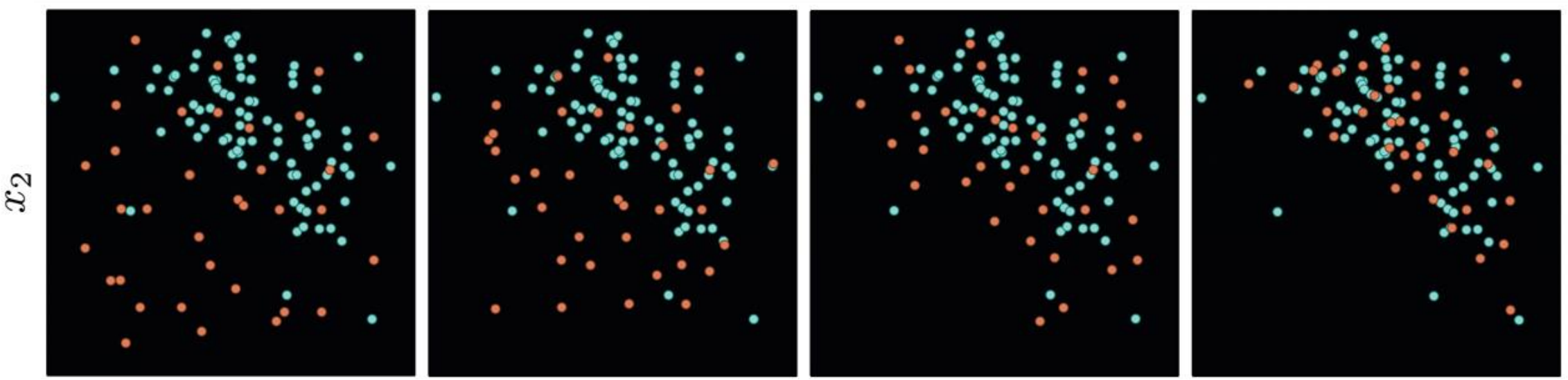


Figure 15.1 GAN mechanism. a) Given a parameterized function (a generator) that synthesizes samples (orange arrows) and a batch of real examples (cyan arrows), we train a discriminator to distinguish the real examples from the generated samples (sigmoid curve indicates the estimated probability that the data point is real). b) The generator is trained by modifying its parameters so that the discriminator becomes less confident the samples were synthetic (in this case, by moving the orange samples to the right). The discriminator is then updated. c) Alternating updates to the generator and discriminator cause the generated samples to become indistinguishable from real examples and the impetus to change the generator (i.e., the slope of the sigmoid function) to diminish.

Generative Adversarial Networks

Example 2D



Generative Adversarial Networks provide a mechanism for generating samples (orange points). As training proceeds (left to right), the loss function encourages these samples to become progressively less distinguishable from real examples (cyan points)

GAN loss function

Discriminator $\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$: Takes as input **x** and returns a scalar expressing its belief that it is a real example

GAN loss function

Discriminator $\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$: Takes as input **x** and returns a scalar expressing its belief that it is a real example

Binary classification problem
→ cross-entropy loss

GAN loss function

Discriminator $\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$: Takes as input **x** and returns a scalar expressing its belief that it is a real example:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_i -(1 - y_i) \log\left[1 - \mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\right] - y_i \log\left[\mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\right] \right]$$

$y_i \in \{0, 1\}$

Logistic sigmoid function

GAN loss function

Discriminator $\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$: Takes as input **x** and returns a scalar expressing its belief that it is a real example:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_i -(1-y_i) \log\Big[1 - \mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] - y_i \log\Big[\mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] \right]$$

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_j - \log\Big[1 - \mathrm{sig}[\mathrm{f}[\mathbf{x}_j^*, \boldsymbol{\phi}]]\Big] - \sum_i \log\Big[\mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] \right]$$

generated examples

real samples

GAN loss function

Discriminator $\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$: Takes as input **x** and returns a scalar expressing its belief that it is a real example:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_j -\log\left[1 - \mathrm{sig}[\mathrm{f}[\mathbf{x}_j^*, \boldsymbol{\phi}]]\right] - \sum_i \log\left[\mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\right] \right]$$

Substitute the generator $\mathbf{x}_j^* = \mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}]$:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\min_{\boldsymbol{\phi}} \left[\sum_j -\log\left[1 - \mathrm{sig}[\mathrm{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\right] - \sum_i \log\left[\mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\right] \right] \right]$$

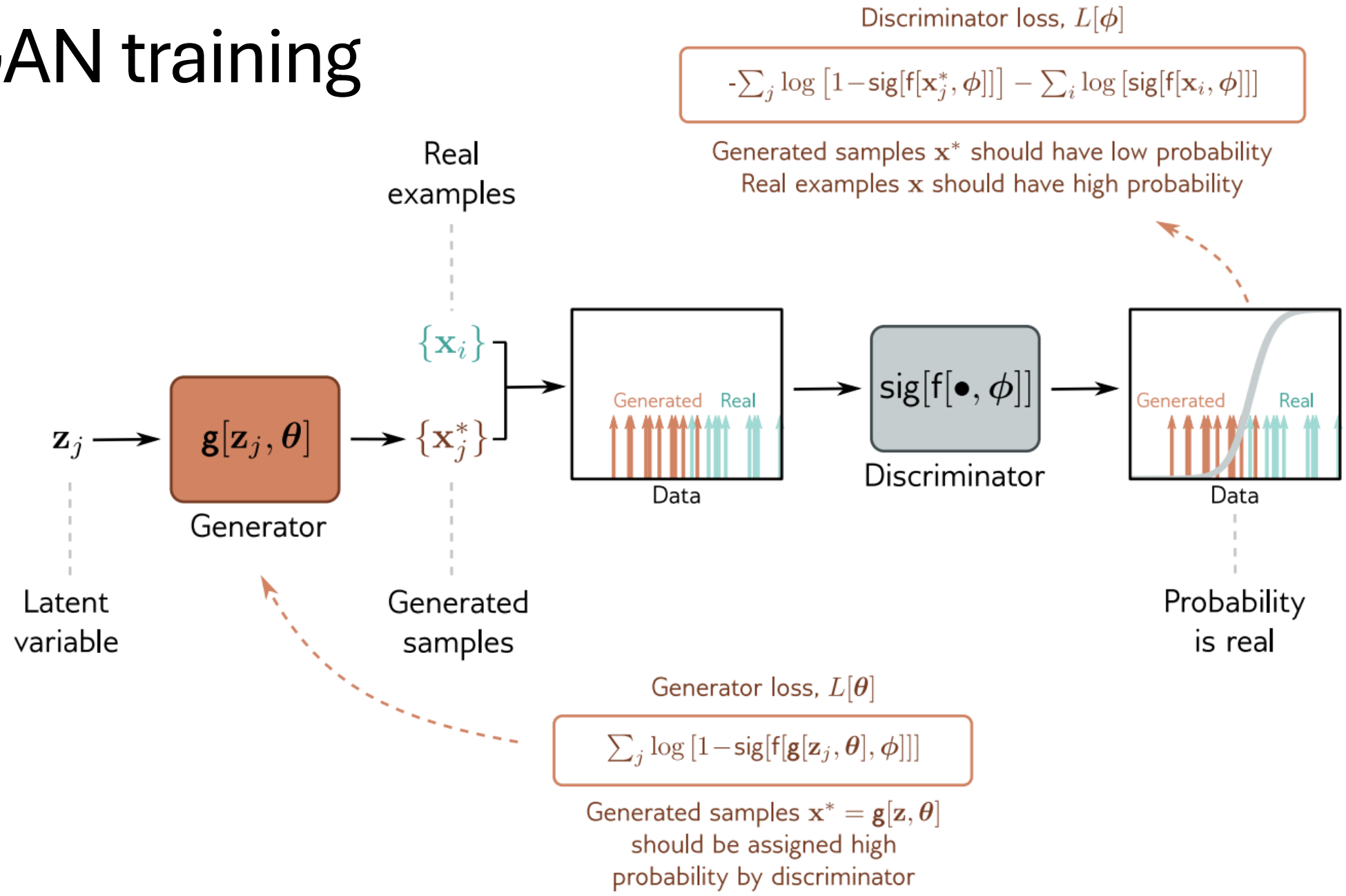
GAN training

Two inter-dependent loss functions:

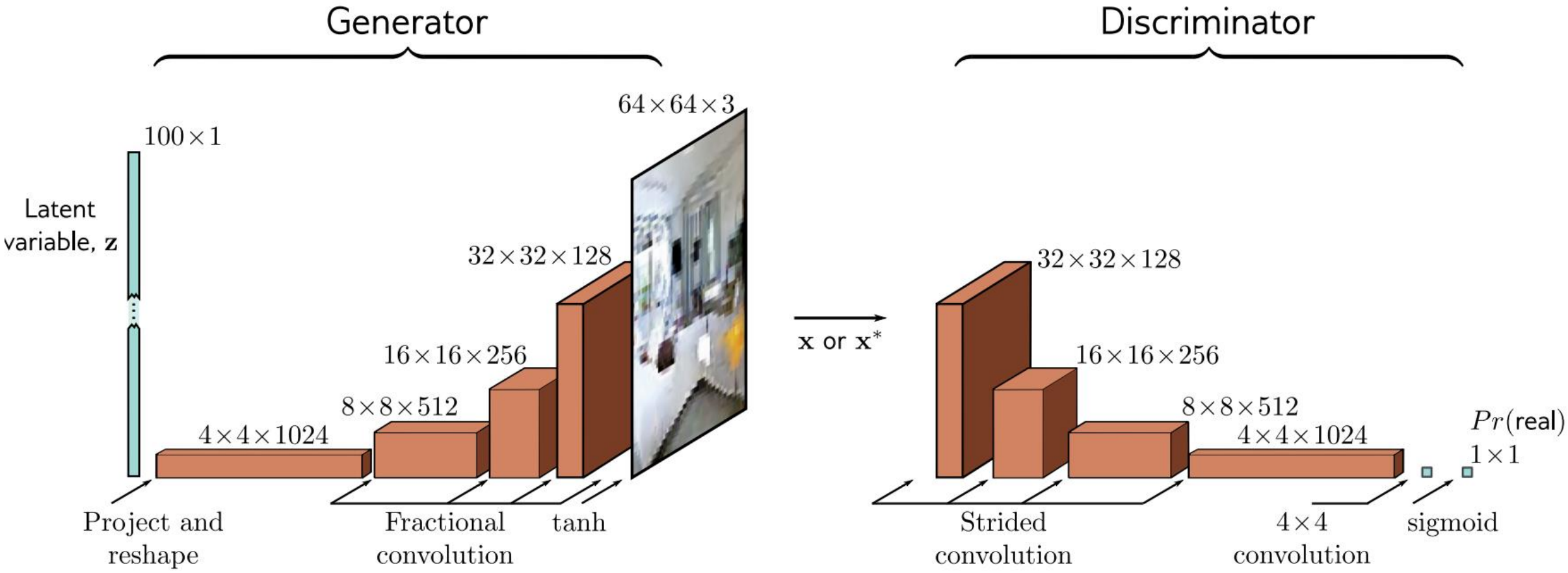
$$L[\boldsymbol{\phi}] = \sum_j -\log\Big[1 - \text{sig}[\text{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\Big] - \sum_i \log\Big[\text{sig}[\text{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big]$$

$$L[\boldsymbol{\theta}] = \sum_j \log\Big[1 - \text{sig}[\text{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\Big]$$

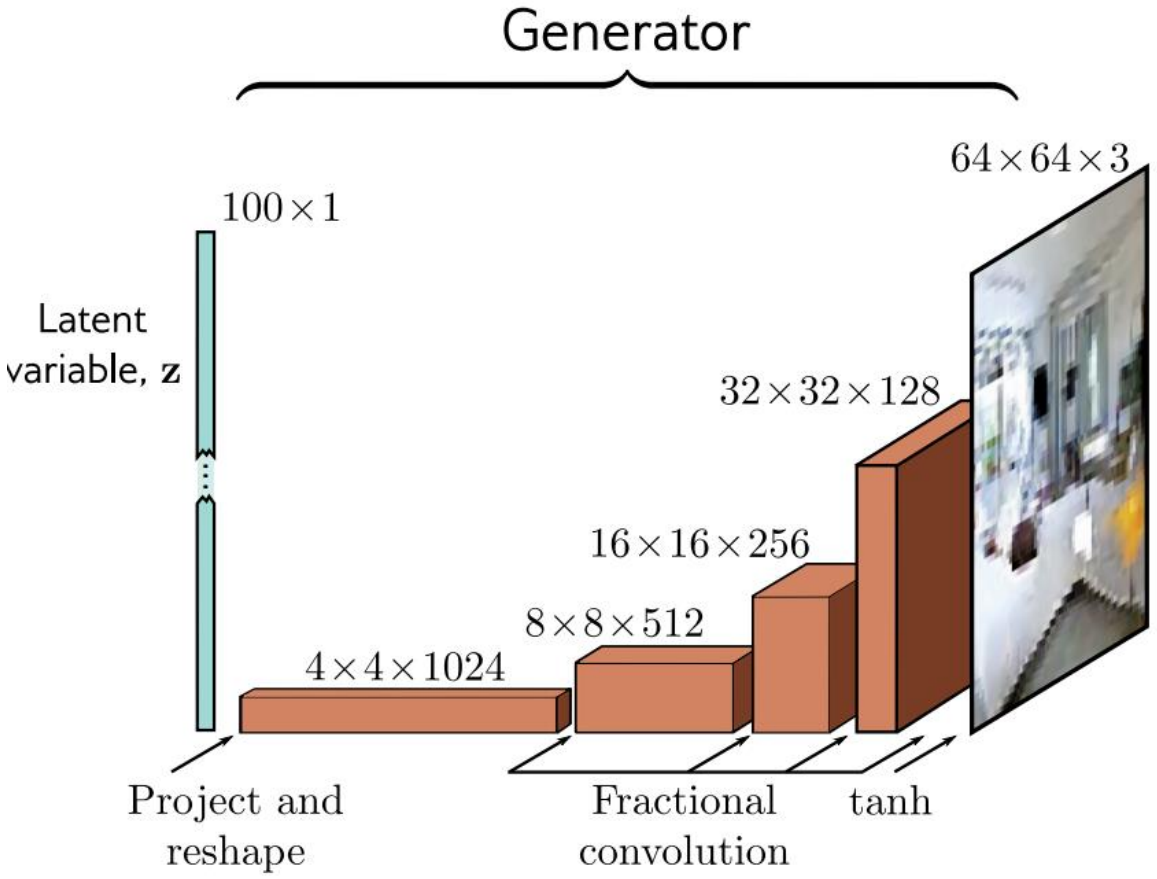
GAN training



Deep Convolutional GAN (DCGAN)



Deep Convolutional GAN (DCGAN)



After training ends, the Discriminator network is discarded

Deep Convolutional GAN (DCGAN)

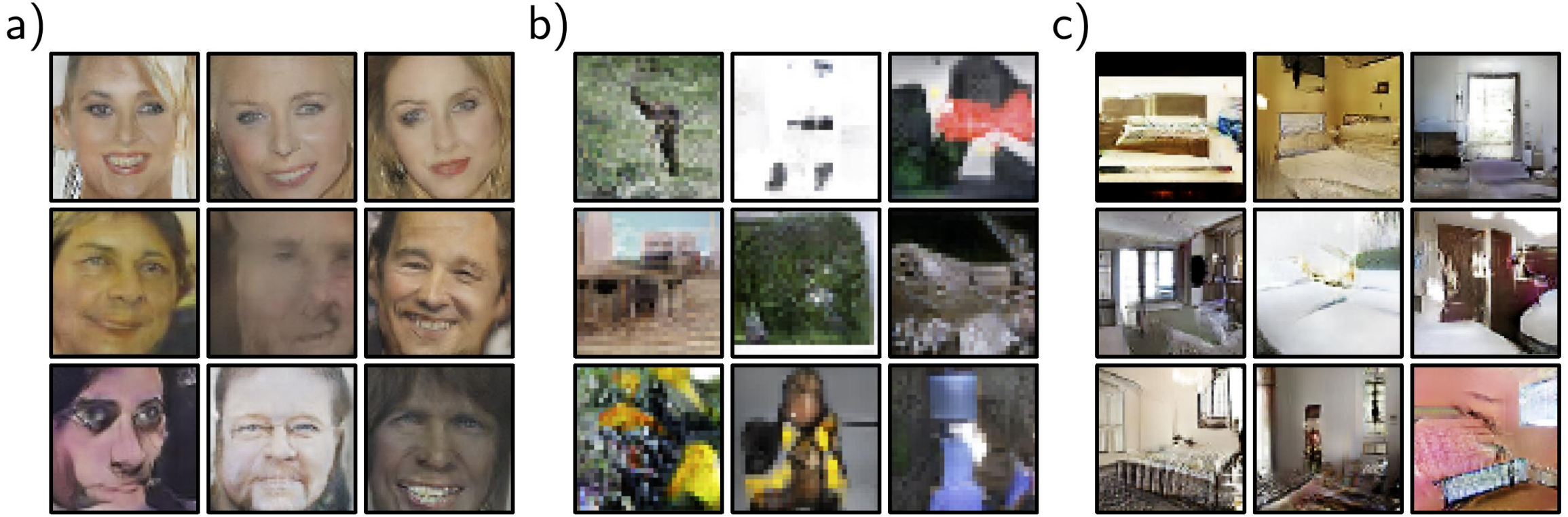


Figure 15.4 Synthesized images from the DCGAN model. a) Random samples drawn from DCGAN trained on a faces dataset. b) Random samples using the ImageNet database (see figure 10.15). c) Random samples drawn from the LSUN scene understanding dataset. Adapted from Radford et al. (2015).

Quality of generated images

Mode dropping: The generator makes plausible examples, but they only represent a subset of the data

Mode collapse: The generator collapses all samples to one or a few points

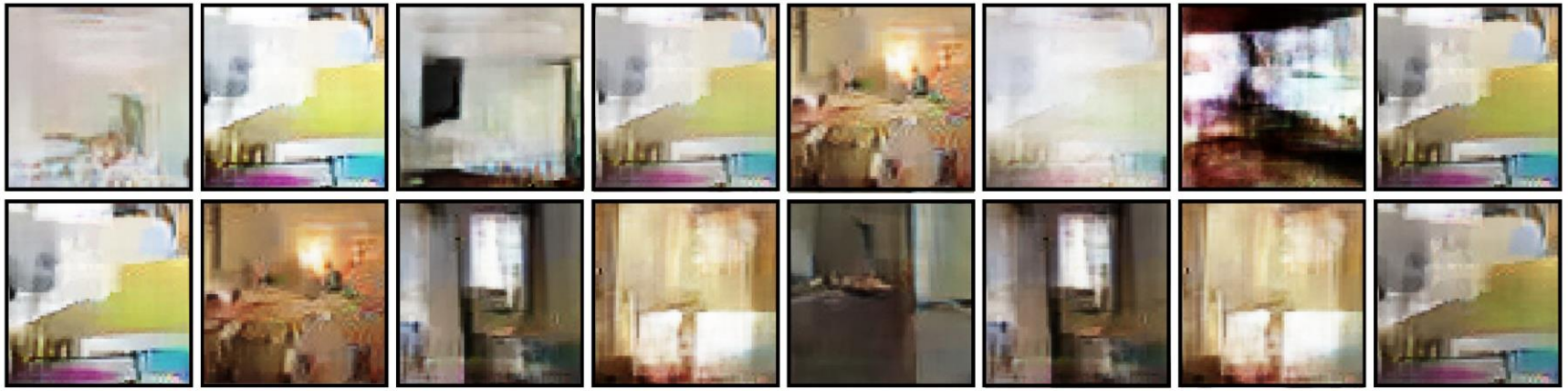


Figure 15.5 Mode collapse. Synthesized images from a GAN trained on the LSUN scene understanding dataset using an MLP generator with a similar number of parameters and layers to the DCGAN. The samples are low quality, and many are similar. Adapted from Arjovsky et al. (2017).

Quality of generated images

Loss function equivalent to the minimization of the Jensen-Shannon divergence between the synthetic data distribution $Pr(x^*)$ and the real data distribution $Pr(x)$:

$$
\begin{aligned}
& D_{JS}\left[Pr(\mathbf{x}^*) \,||\, Pr(\mathbf{x})\right] \\
& \quad = \frac{1}{2} D_{KL}\left[Pr(\mathbf{x}^*) \,\middle|\middle|\, \frac{Pr(\mathbf{x}^*) + Pr(\mathbf{x})}{2}\right] + \frac{1}{2} D_{KL}\left[Pr(\mathbf{x}) \,\middle|\middle|\, \frac{Pr(\mathbf{x}^*) + Pr(\mathbf{x})}{2}\right] \\
& \quad = \frac{1}{2}\int \underbrace{Pr(\mathbf{x}^*) \log\left[\frac{2Pr(\mathbf{x}^*)}{Pr(\mathbf{x}^*) + Pr(\mathbf{x})}\right] d\mathbf{x}^*}_{\text{quality}} + \underbrace{\frac{1}{2}\int Pr(\mathbf{x}) \log\left[\frac{2Pr(\mathbf{x})}{Pr(\mathbf{x}^*) + Pr(\mathbf{x})}\right] d\mathbf{x}}_{\text{coverage}}.
\end{aligned}
$$

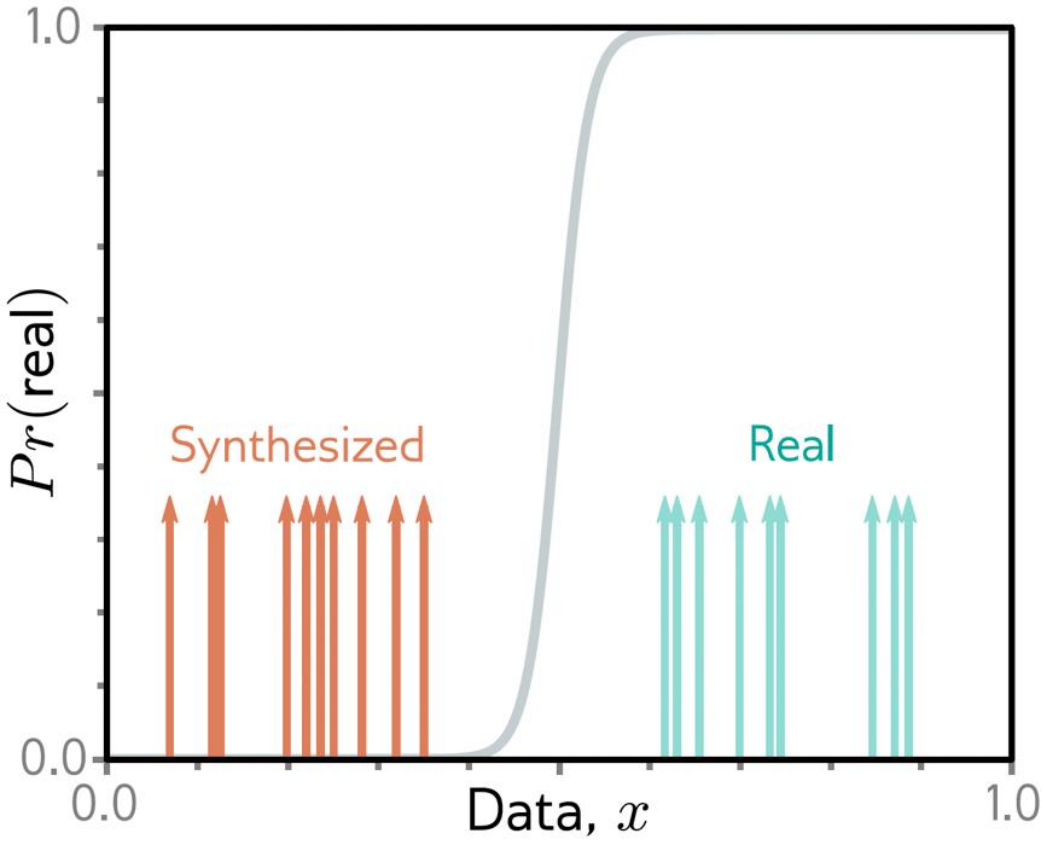
Penalizes regions with generated samples but not real samples

Does not depend on the generator
➔ Mode dropping

Quality of generated images

Changes in the generator will not decrease the loss when:

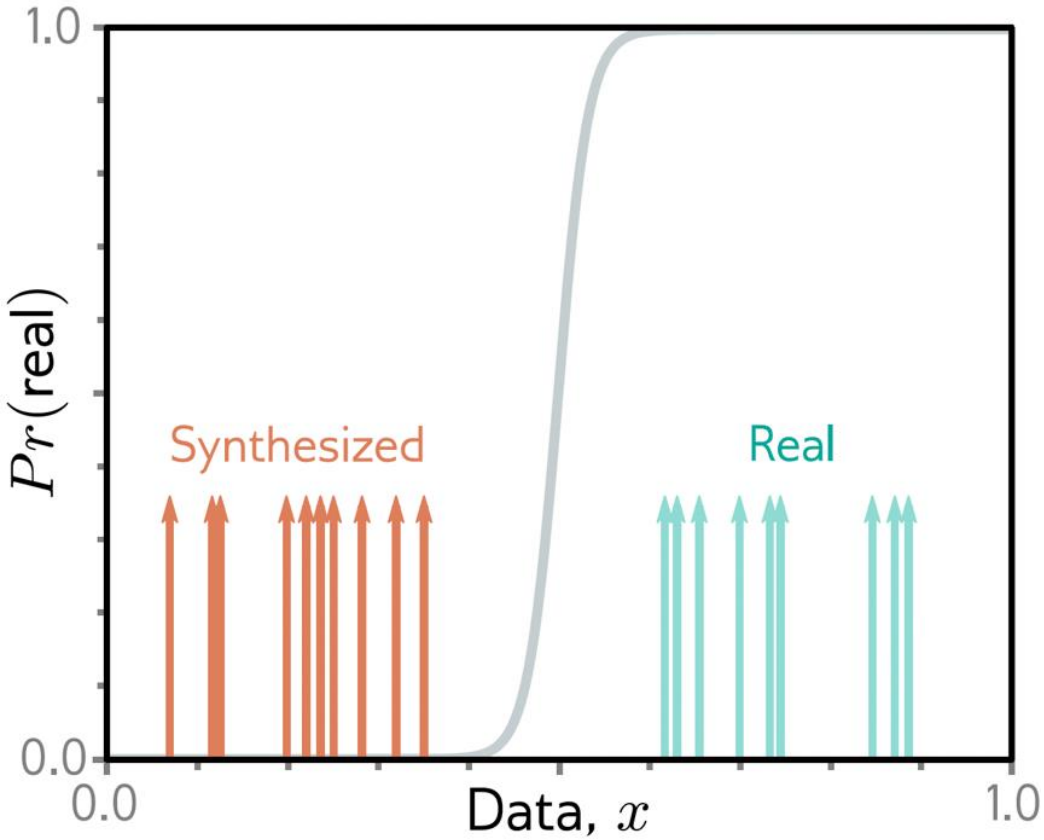
- The probability distributions of the real and generated samples are disjoint
- The discriminator can perfectly separate the generated and real samples



Quality of generated images

Changes in the generator will not decrease the loss when:

- The probability distributions of the real and generated samples are disjoint
- The discriminator can perfectly separate the generated and real samples



Solution: Choose a distance metric with better properties

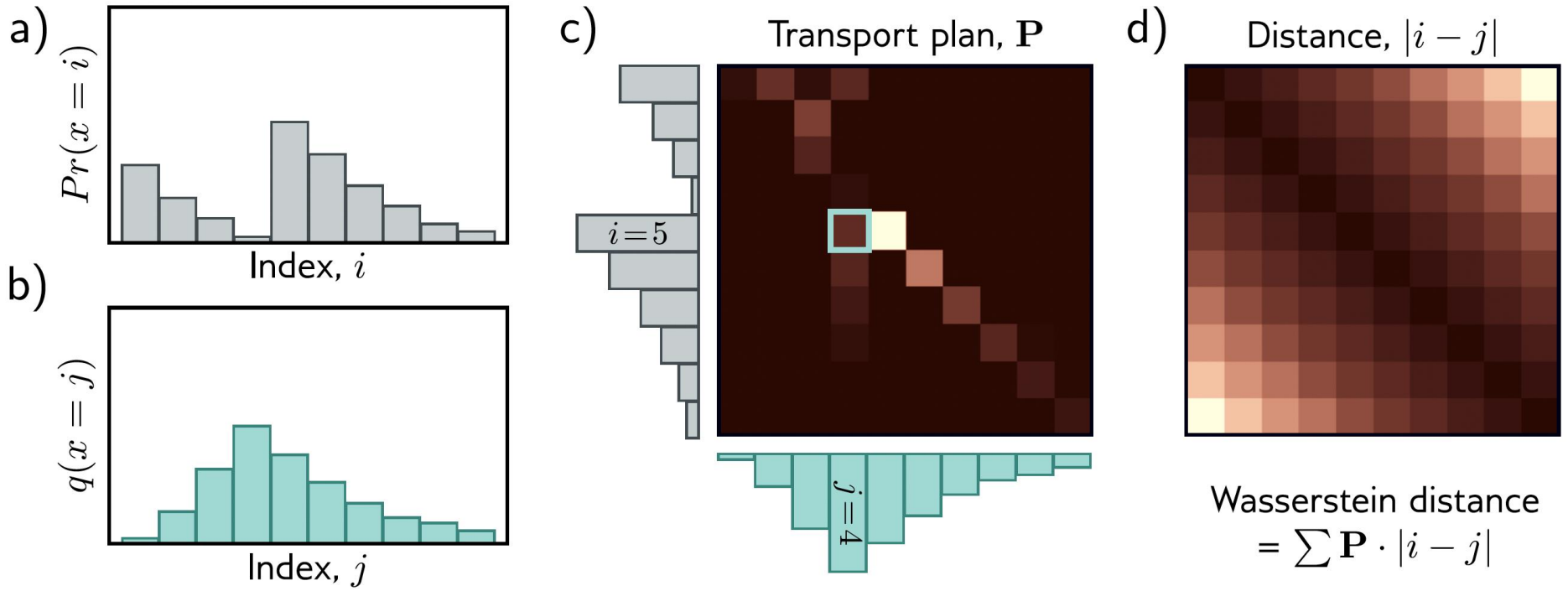


Figure 15.8 Wasserstein or earth mover's distance. a) Consider the discrete distribution $Pr(x=i)$. b) We wish to move the probability mass to create the target distribution $q(x=j)$. c) The transport plan $\mathbf{P}$ identifies how much mass will be moved from i to j. For example, the cyan highlighted square p_{54} indicates how much mass will be moved from $i=5$ to $j=4$. The elements of the transport plan must be non-negative, the sum over j must be $Pr(x=i)$, and the sum over i must be $q(x=j)$. Hence $\mathbf{P}$ is a joint probability distribution. d) The distance matrix between elements i and j. The optimal transport plan $\mathbf{P}$ minimizes the sum of the pointwise product of $\mathbf{P}$ and the distance matrix (termed the Wasserstein distance). Hence, the elements of $\mathbf{P}$ tend to lie close to the diagonal where the distance cost is lowest. Adapted from Hermann (2017).

Wasserstein distance

Solution of a linear programming problem:

$$D_w\left[Pr(x)||q(x)\right] = \min_{\mathbf{P}}\left[\sum_{i,j} P_{ij}\cdot|i-j|\right],$$

subject to:

$$\begin{aligned}\textstyle\sum_j P_{ij} &= Pr(x=i) \\ \textstyle\sum_i P_{ij} &= q(x=j) \\ P_{ij} &\geq 0\end{aligned}$$

Initial distribution of Pr(x)

Initial distribution of q(x)

Non-negativity

Wasserstein distance

Dual problem is easier to solve:

$$D_w\Big[Pr(x)||q(x)\Big] = \max_{\mathbf{f}}\left[\sum_i Pr(x=i)f_i - \sum_j q(x=j)f_j\right]$$

subject to: $|f_{i+1} - f_i| < 1$

Wasserstein loss function

$$
\begin{aligned}
L[\boldsymbol{\phi}] &= \sum_j \mathrm{f}[\mathbf{x}_j^*, \boldsymbol{\phi}] - \sum_i \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}] \\
&= \sum_j \mathrm{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}] - \sum_i \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]
\end{aligned}
$$

with the constraint that: $\left| \frac{\partial \mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]}{\partial \mathbf{x}} \right| < 1$

Wasserstein loss function

$$L[\boldsymbol{\phi}] = \sum_j \mathrm{f}[\mathbf{x}_j^*, \boldsymbol{\phi}] - \sum_i \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]$$

$$= \sum_j \mathrm{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}] - \sum_i \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]$$

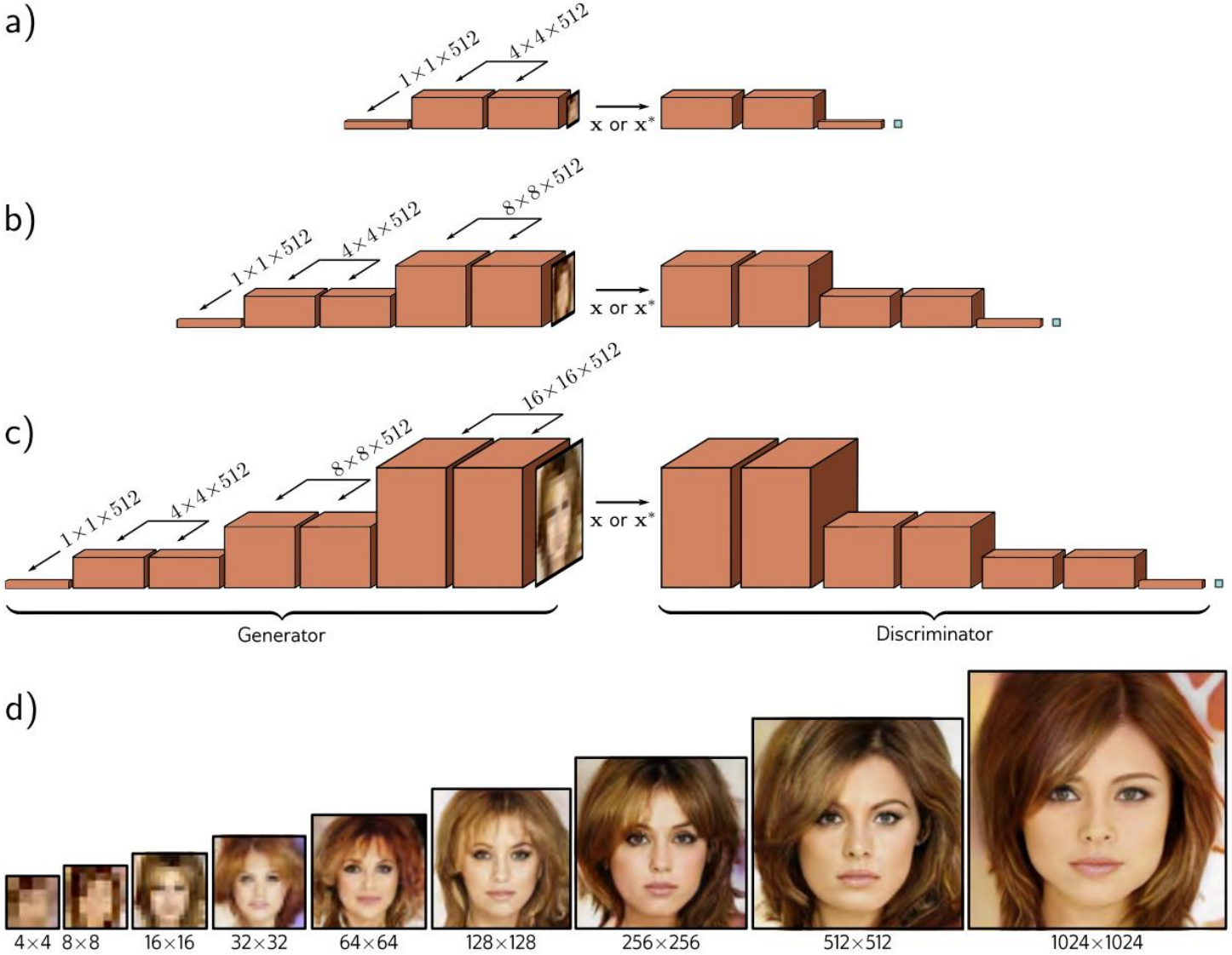
with the constraint that: $\left|\frac{\partial \mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]}{\partial \mathbf{x}}\right| < 1$ ←

Ways to enforce the constraint:

1. Weight clipping to have small range (e.g., ±0.01)
2. Add a regularization term to constrain the gradient norm deviations from 1

Processes for high-quality image generation

- Progressive growing



Processes for high-quality image generation

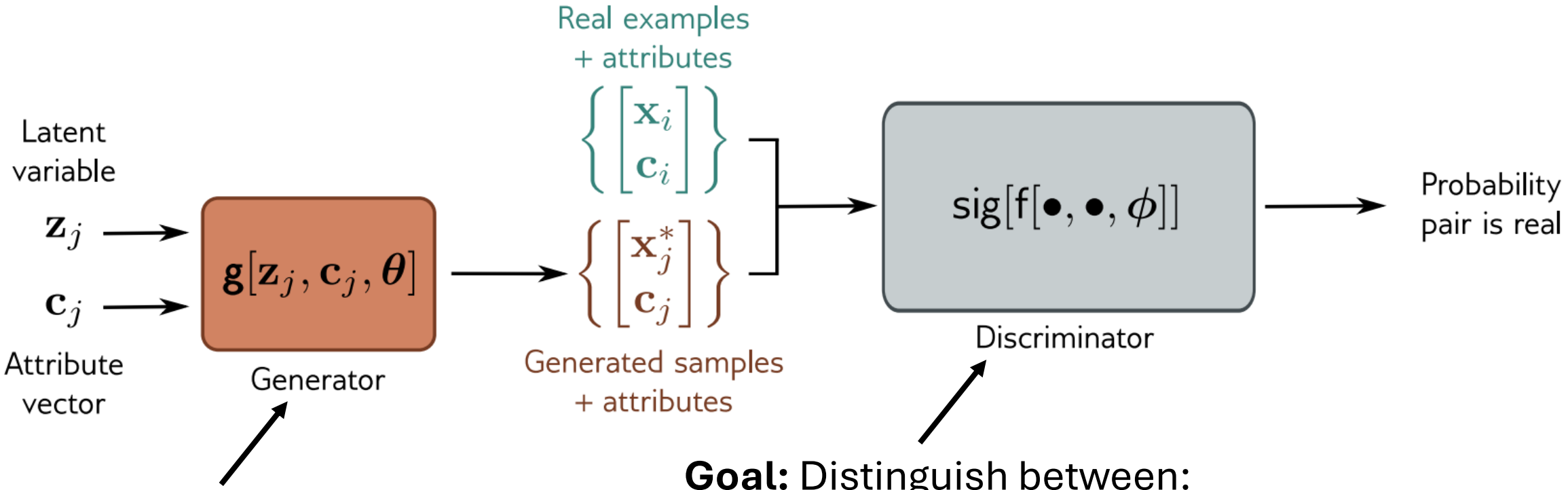
- Progressive growing
- Mini-batch discrimination: making the samples to have sufficient variety (to prevent mode collapse)

Processes for high-quality image generation

- Progressive growing
- Mini-batch discrimination
- Truncation: choose latent variables **z** that have high probability (close to the mean)

Conditional GAN

- Pass a vector **c** of attributes to both the generator and discriminator

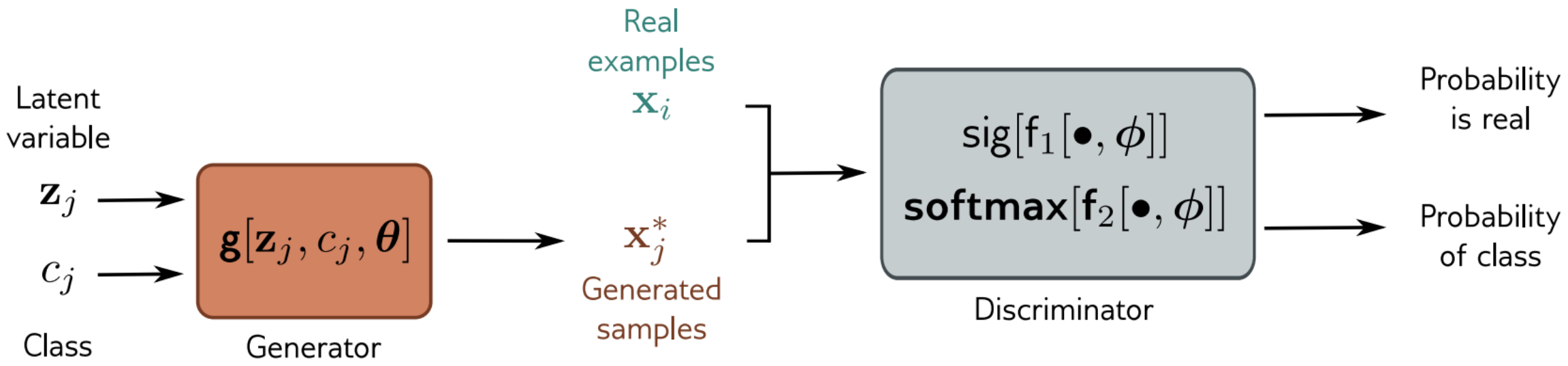


Goal: transform **z** to a data sample **x** with the correct attribute **c**

Goal: Distinguish between:
(i) the generated sample with the target attribute
(ii) A real sample with the real attribute

Auxiliary classifier GAN (ACGAN)

- Requires the discriminator to predict the attribute



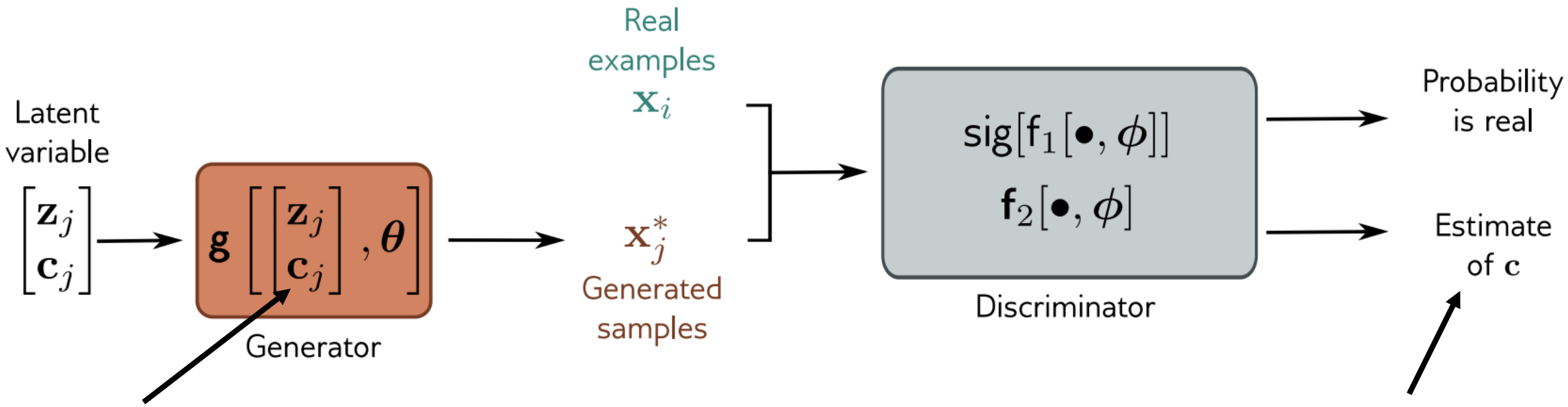
Auxiliary classifier GAN (ACGAN)

Figure 15.14 Auxiliary classifier GAN. The generator takes a class label as well as the latent vector. The discriminator must both identify if the data point is real *and* predict the class label. This model was trained on ten ImageNet classes. Left to right: generated examples of monarch butterflies, goldfinches, daisies, redshanks, and gray whales. Adapted from Odena et al. (2017).

InfoGAN

- Attempts to identify important attributes automatically

Idea: interpretable real-world characteristics should be easy to predict



Random attribute variable $\mathbf{c}_j$:

- salient aspects of the data
- the latent space is disentangled

If c is discrete → binary or multi-class entropy loss

If c is continuous → least squares loss

InfoGAN

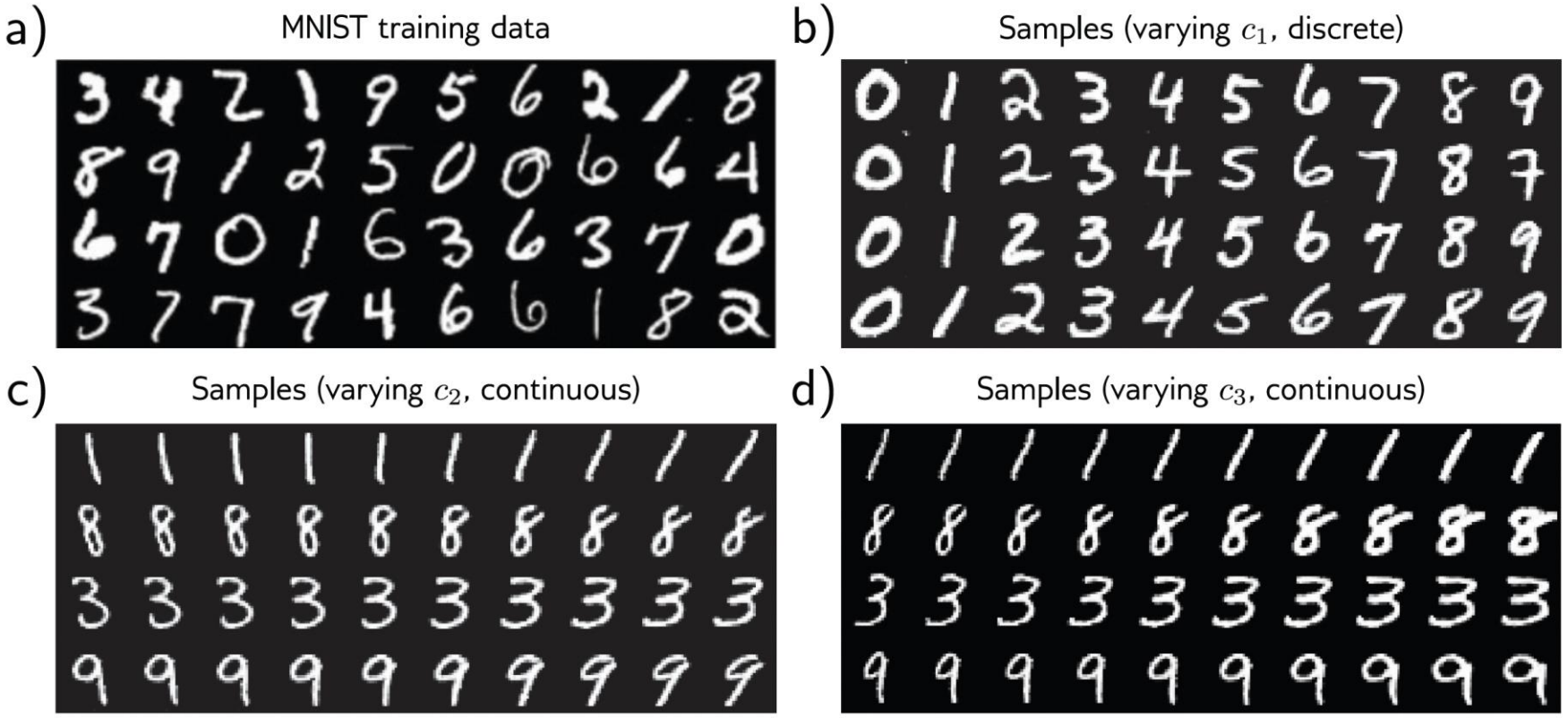
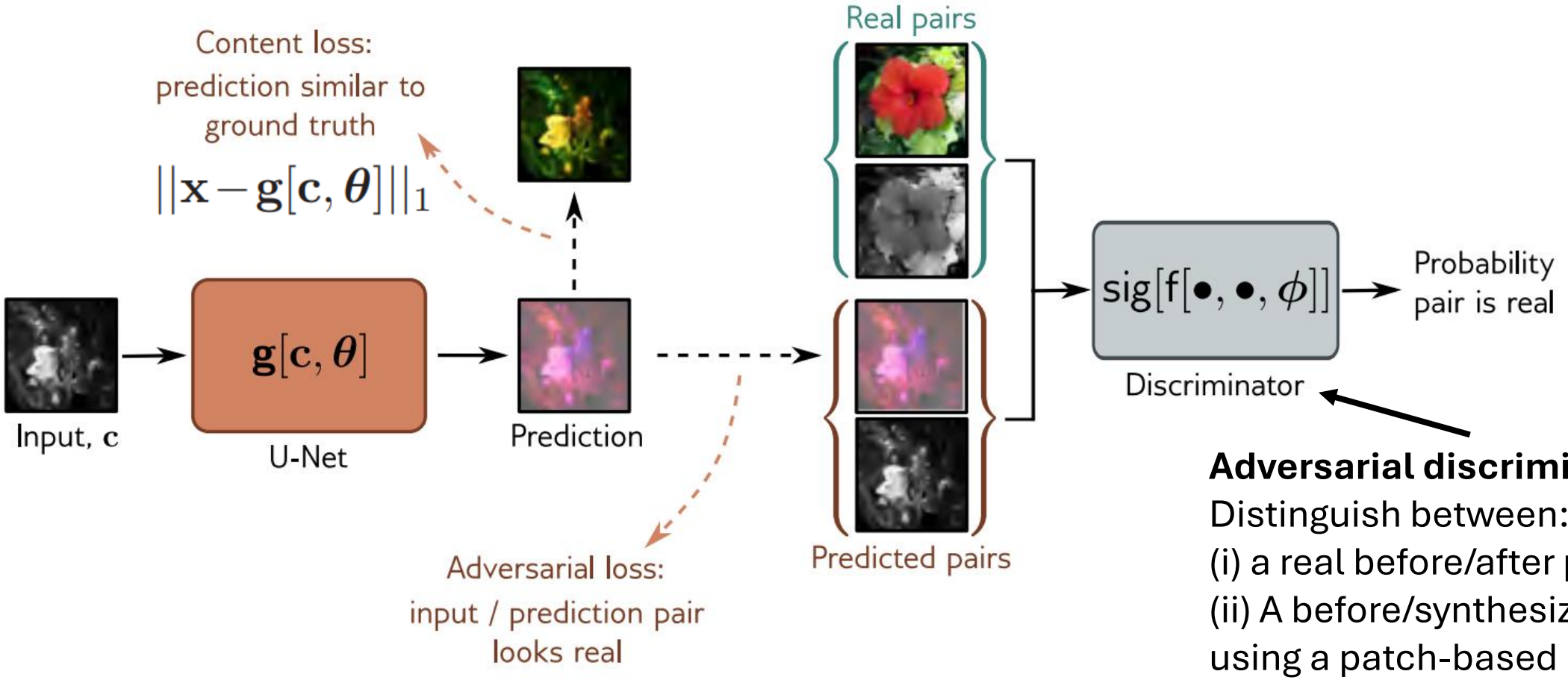


Figure 15.15 InfoGAN for MNIST. a) Training examples from the MNIST database, which consists of 28×28 pixel images of handwritten digits. b) The first attribute c_1 is categorical with 10 categories; each column shows samples generated with one of these categories. The InfoGAN recovers the ten digits. The attribute vectors c_2 and c_3 are continuous. c) Moving from left to right, each column represents a different value of c_2 while keeping the other latent variables constant. This attribute seems to correspond to the orientation of the character. d) The third attribute seems to correspond to the thickness of the stroke. Adapted from Chen et al. (2016b).

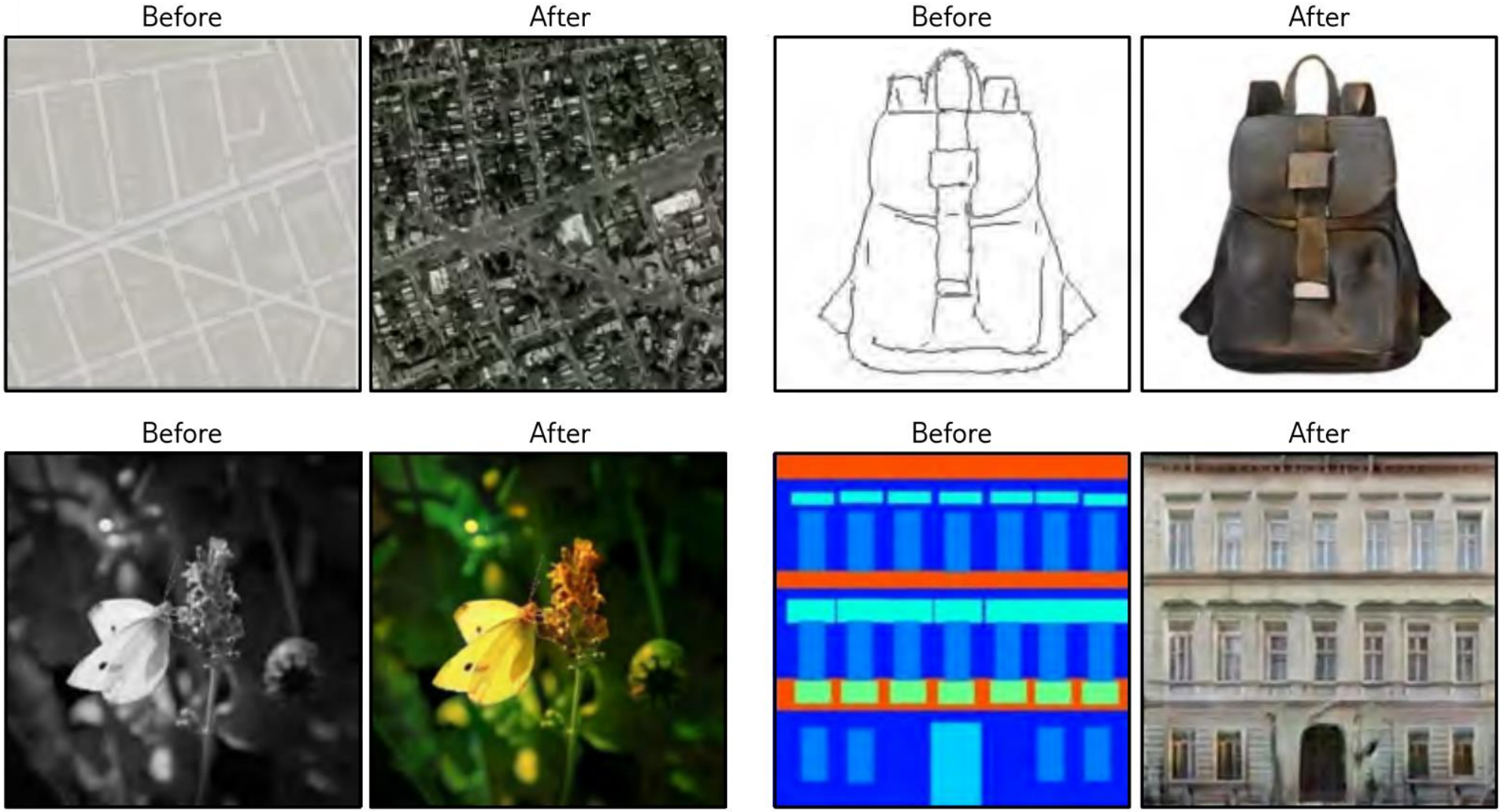
Pix2Pix

- Performs image translation using before/after pairs for training



Adversarial discriminator
Distinguish between:
(i) a real before/after pair and
(ii) A before/synthesized paper
using a patch-based loss (PatchGAN)

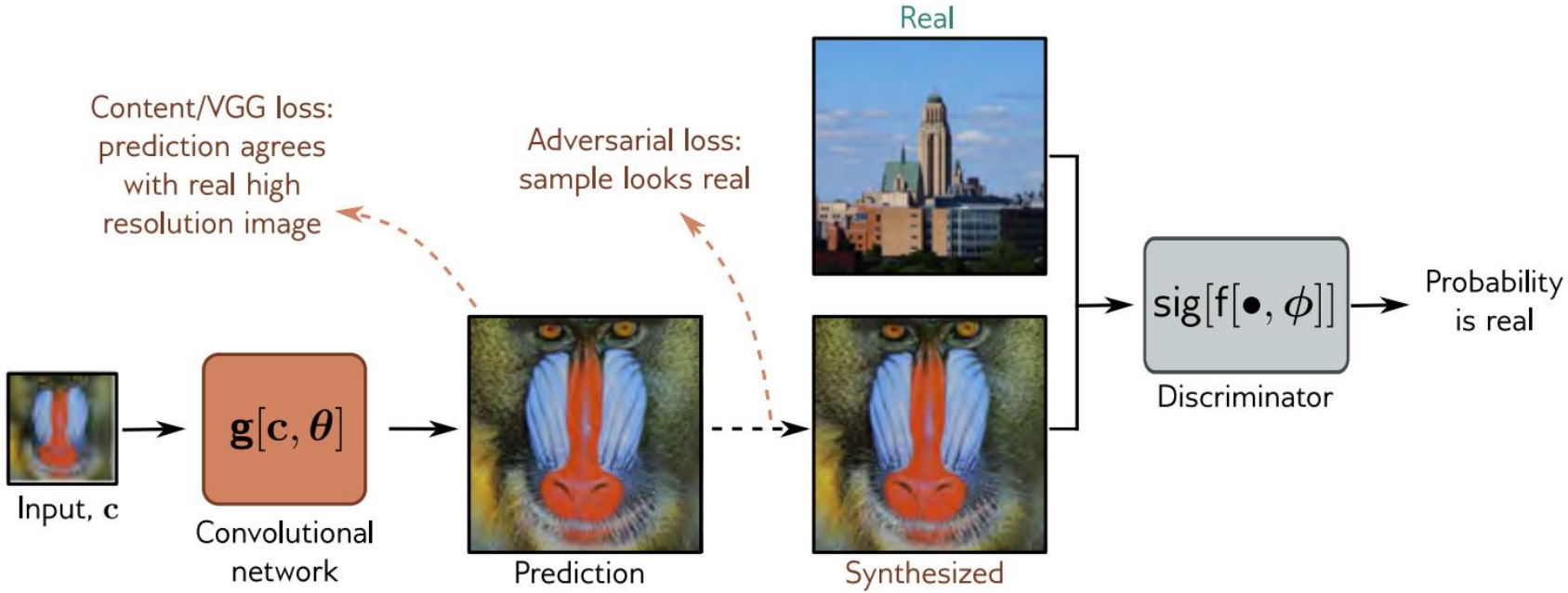
Pix2Pix



Super-resolution GAN (SRGAN)

Three losses:

- Content loss: squared difference between the output and the true high-resolution image
- VGG/perceptual loss: squared difference between VGG output for the synthesized and ground truth images
- Adversarial loss: real high-resolution image or upsampled image?

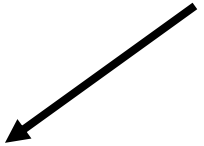


CycleGAN

- Uses two sets of photos with different styles (not image pairs)

CycleGAN

E.g. there exist many photos and many Monet paintings, but no correspondence between them.



- Uses two sets of photos with different styles (not image pairs)

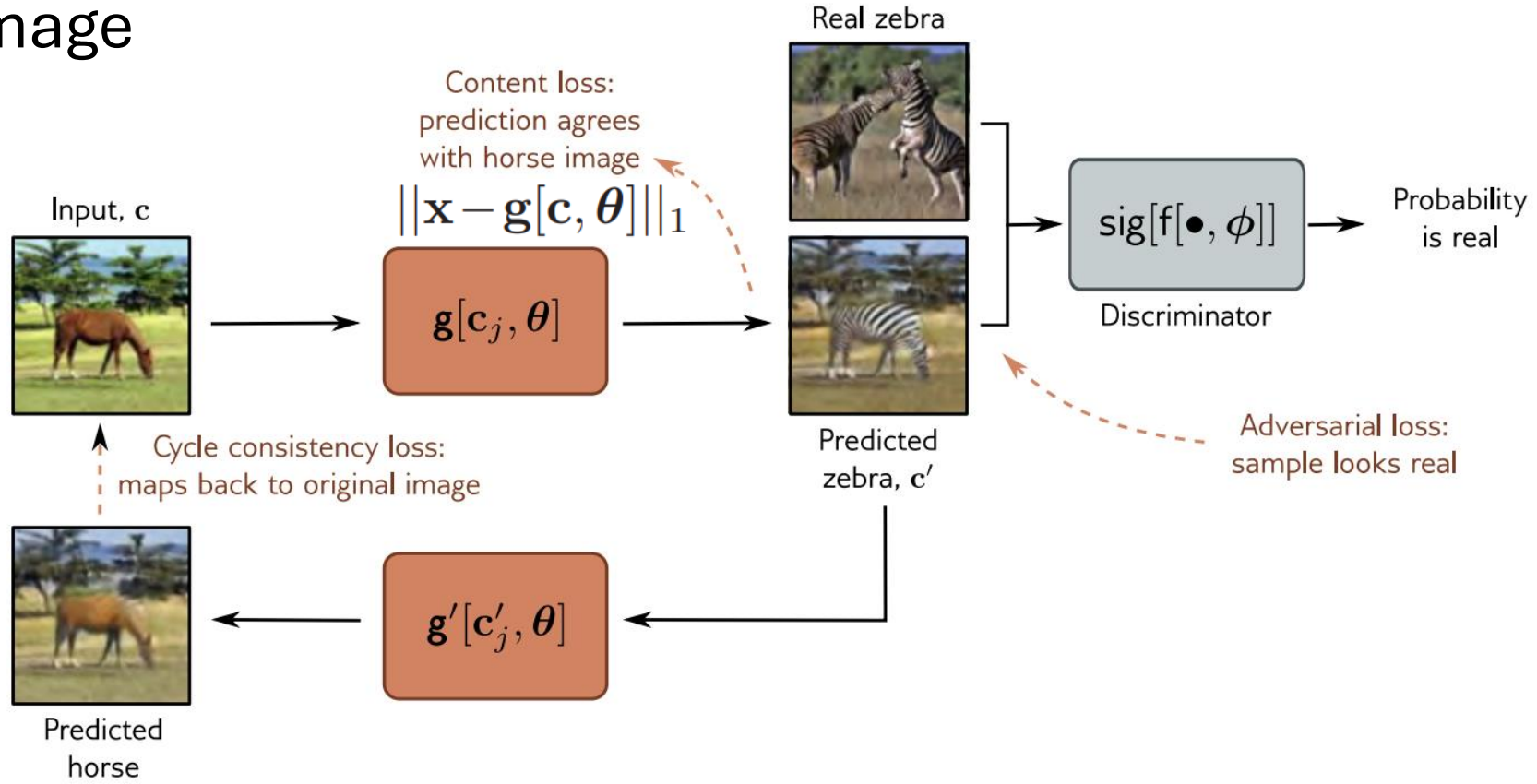
CycleGAN

- Uses two sets of photos with different styles (not image pairs)

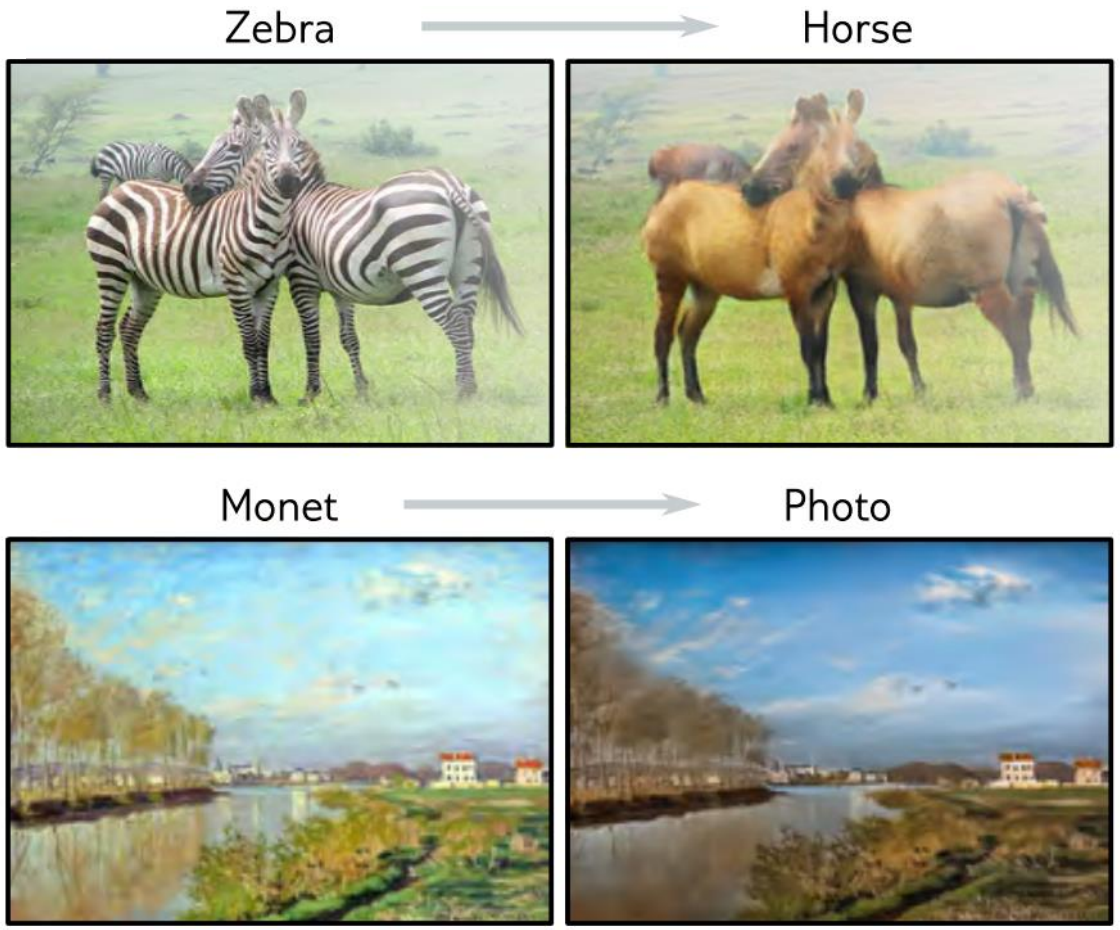
Idea: converting an image in one direction and then back should lead to the original image

Three losses:

- Content loss: before and after images should be similar
- Adversarial loss: the output should be similar to real examples of the target domain
- Cycle-consistency loss: the mapping should be reversible



CycleGAN



StyleGAN

- Controls the output image at different scales and separates style from noise
- Uses many latent variables z:
  - Introduced at different levels of the architecture
  - Model styles at different scales

Idea: Style corresponds to aspects of the image that are salient to humans, while noise aspects are unimportant variations

StyleGAN

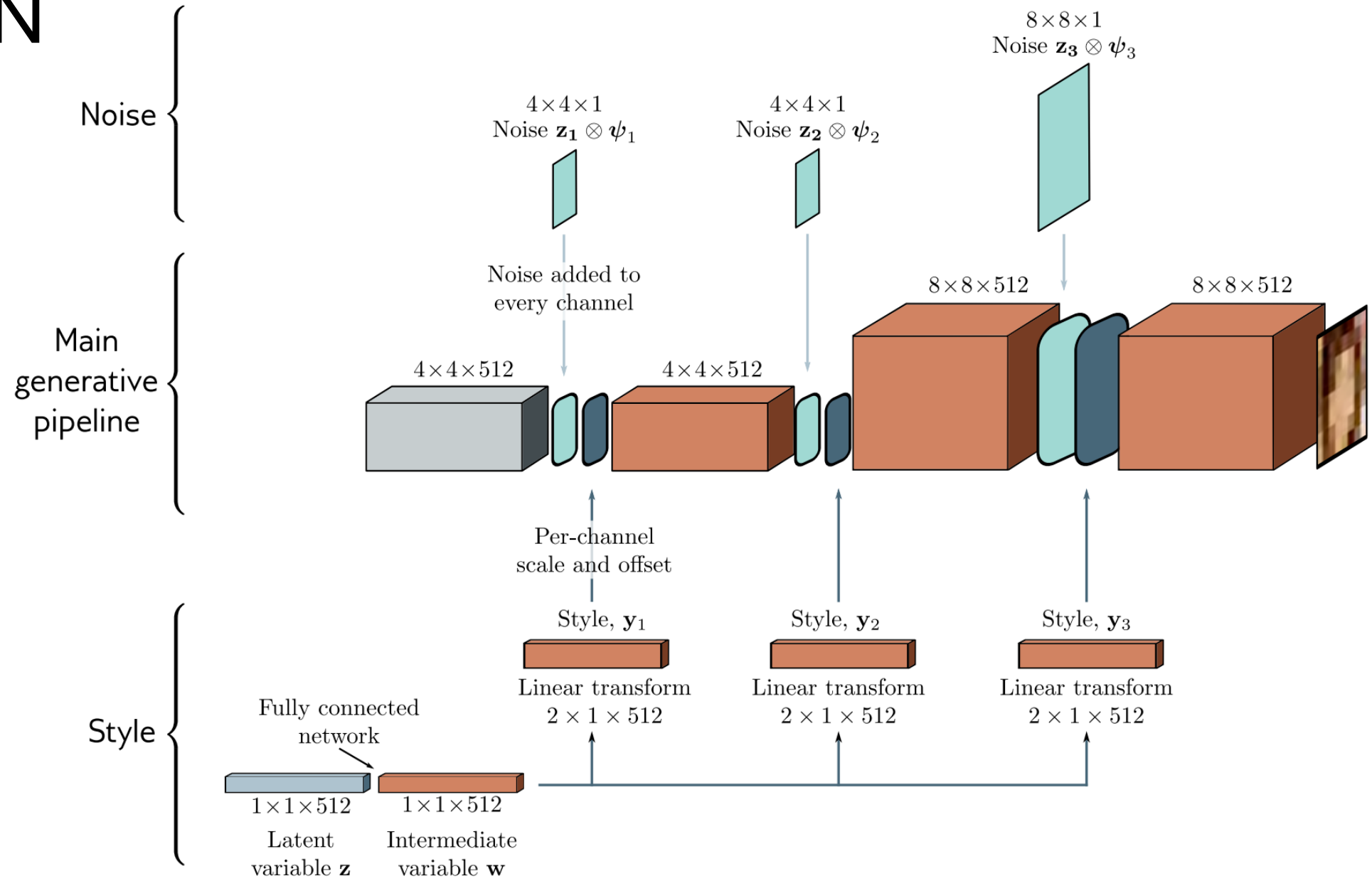
- Controls the output image at different scales and separates style from noise
- Uses many latent variables z:
  - Introduced at different levels of the architecture
  - Model styles at different scales

Idea: Style corresponds to aspects of the image that are salient to humans, while noise aspects are unimportant variations

Example: human faces

- Styles: face shape, head pose, shape & details of facial features
- Noise: exact placement of hair, stubble, freckles, skin pores

StyleGAN


Noise
Main generative pipeline
Style
4×4×1
Noise $\mathbf{z_1} \otimes \psi_1$
4×4×1
Noise $\mathbf{z_2} \otimes \psi_2$
8×8×1
Noise $\mathbf{z_3} \otimes \psi_3$
Noise added to every channel
4×4×512
4×4×512
8×8×512
8×8×512
Per-channel scale and offset
Style, $\mathbf{y}_1$
Style, $\mathbf{y}_2$
Style, $\mathbf{y}_3$
Linear transform $2 \times 1 \times 512$
Linear transform $2 \times 1 \times 512$
Linear transform $2 \times 1 \times 512$
Fully connected network
1×1×512
1×1×512
Latent variable $\mathbf{z}$
Intermediate variable $\mathbf{w}$

StyleGAN

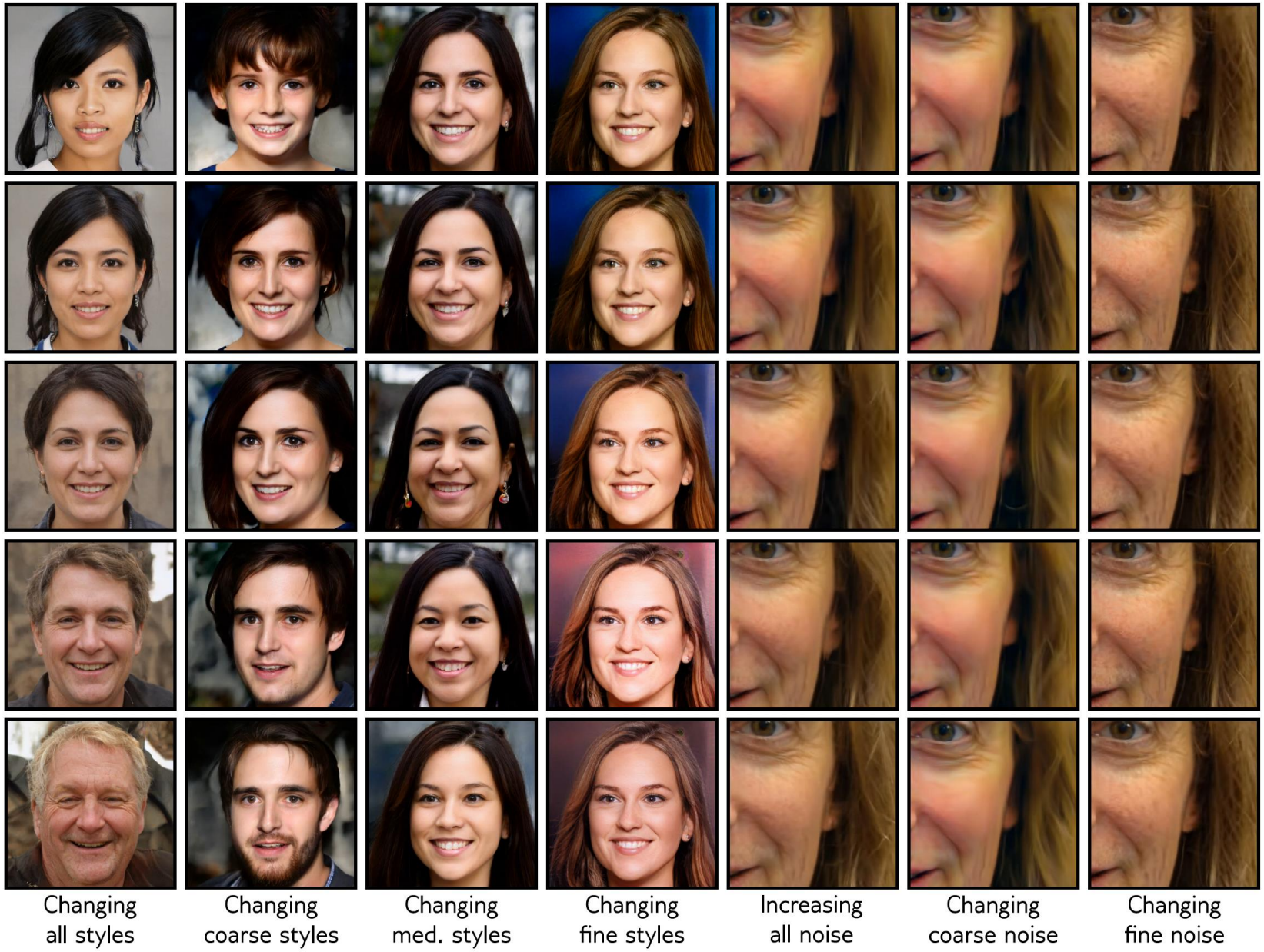


Figure 15.20 StyleGAN results. First four columns show systematic changes in style at various scales. Fifth column shows the effect of increasing noise magnitude. Last two columns show different noise vectors at two different scales.