







# Unsupervised Learning & Generative Adversarial Networks

DL4DS – Spring 2025

# April Dates

| Sunday | Monday           | Tuesday  | Wednesday             | Thursday  | Friday  | Saturday |
|--------|------------------|--|-----------------------|---|---|----------|
|        | April 1          | 2  | 3                     | 4<br>GANs                              | 5   | 6        |
| 7      | 8                | 9<br>VAEs  | 10<br>Discussion      | 11<br>Diffusion Models  | 12  | 13       |
| 14     | 15               | 16<br>Graph Neural<br>Nets   | 17<br>Discussion      | 18<br>Reinforcement<br>Learning   | 19  | 20       |
| 21     | 22               | 23<br>TBD/Overflow   | 24<br>Discussion      | 25<br>★ Project <br>Presentations 1 ★ | 26  | 27       |
| 28     | 29               | 30<br>★ Project <br>Presentations 2 ★ | May 1<br>Discussion?? | 2<br>Study Period   | 3<br>Study Period   | 4        |
| 5      | 6<br>Final Exams | 7  | 8                     | 9   | 10<br>Final report<br>& Repo **  | 11       |

\*\* Might be earlier. Depends on when grades are due.

# Project Presentations

Looking for volunteers for April 25.  
Then I will randomly draw remainder of April 25 spots.

Format:

≤ 3 minutes screencast/video

≤ 2 minutes additional presentation

~2 minutes Q&A

## April 25 – 75 minutes

- Slot 1
- Slot 2
- Slot 3
- Slot 4
- Slot 5
- Slot 6
- Slot 7
- Slot 8

## April 30 – 75 minutes

- Slot 9
- Slot 10
- Slot 11
- Slot 12
- Slot 13
- Slot 14
- Slot 15
- Slot 16
- Slot 17

Up to this point...

- we looked at *discriminative supervised learning* models
- Exceptions:
  - Transformers pretrained *unsupervised* (then usually finetuned *supervised*)
  - and the Transformer decoder which *generated* text

**Supervised**            **Unsupervised**

**Discriminative**            **Generative**



# Supervised vs. Self/Unsupervised Learning

## Supervised Learning

**Data:**  $(x, y)$

$x$  is data,  $y$  is a label

**Goal:** Learn function to map  
 $x \rightarrow y$

**Applications:** Classification, regression, object detection, semantic segmentation, etc.

## Self/Unsupervised Learning

**Data:**  $x$

$x$  is data, no labels!

**Goal:** Learn the hidden or underlying structure of the data.

**Applications:** Clustering, dimensionality reduction, compression, find outliers, generating new examples, denoising, interpolating between data points, etc.

# Supervised vs. Self/Unsupervised Learning

## Supervised Learning

**Data:**  $(x, y)$

$x$  is data,  $y$  is a label

**Goal:** Learn function to map  
 $x \rightarrow y$

**Applications:** Classification, regression, object detection, semantic segmentation, etc.

## Self/Unsupervised Learning

**Data:**  $x$

$x$  is data, no labels! Or labels part of the data

**Goal:** Learn the hidden or underlying structure of the data.

**Applications:** Clustering, dimensionality reduction, compression, find outliers, generating new examples, denoising, interpolating between data points, etc.

# We'll consider two attributes of models

- Probabilistic Models
  
- Latent Variable Models

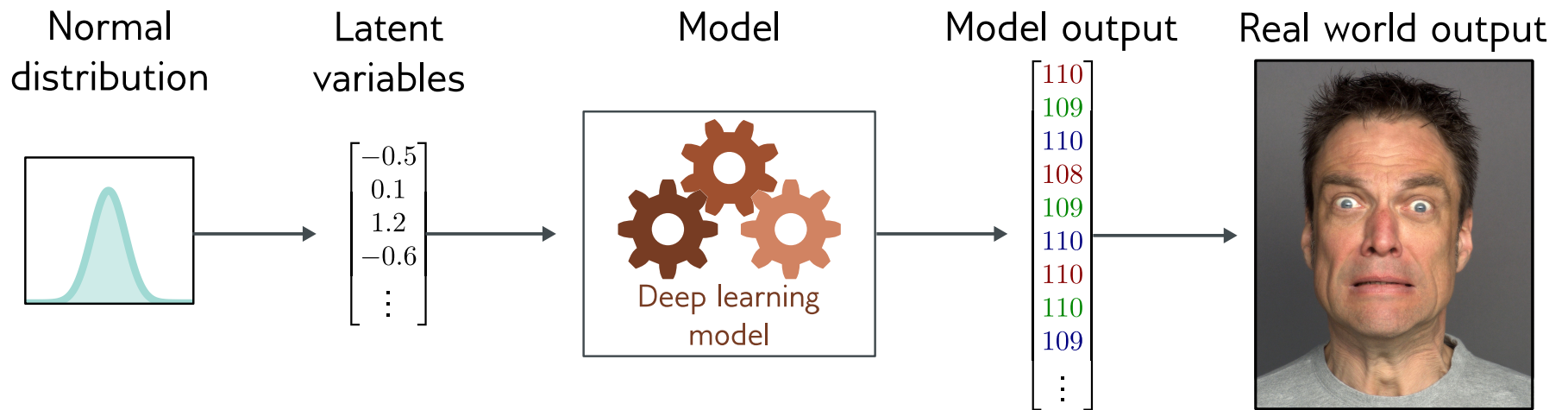
# Probabilistic models

- Maximize log likelihood of training data

$$\hat{\phi} = \operatorname{argmax}_{\phi} \left[ \sum_{i=1}^I \log[\operatorname{Pr}(x_i | \phi)] \right]$$

- Find the parameters,  $\phi$ , of some parametric probability distribution so that the training data is most likely under that distribution

# Latent variable models

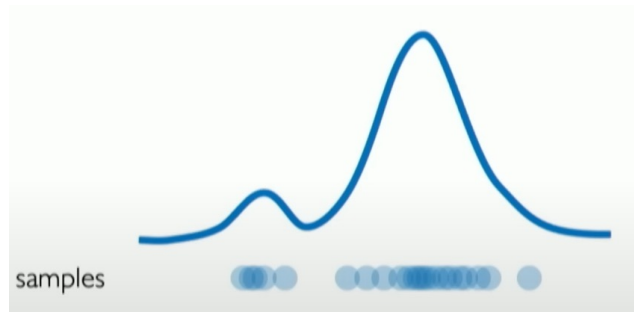


Latent variable models map a random “latent” variable to create a new data sample


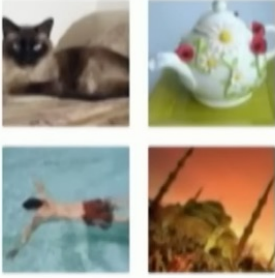
# Generative Modeling

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

## Probability Density Estimation



## Sample Generations

|   |   |
|---|---|
|  |  |
| Input samples   | Generated samples   |
| Training data $\sim P_{data}(x)$  | Generated $\sim P_{model}(x)$   |

How can we learn  $P_{model}(x)$  similar to  $P_{data}(x)$ ?

# Types of unsupervised generative model

- Generative adversarial networks (GANs) (LV)
- Variational auto-encoders (VAEs) (P, LV)
- Diffusion models (P, LV)
- Normalizing flows (P, LV)
- Energy models (P)
- Autoregressive models (P)

# Decoder model: GPT3

- One job: predict the next word in a sequence
- More formally builds an **autoregressive** probability model

$$Pr(t_1, t_2, \dots, t_N) = Pr(t_1) \prod_{n=2}^N Pr(t_n | t_1 \dots t_{n-1})$$

- Doesn't use latent variables, but is probabilistic and generative
  - Can generate new examples
  - Can assign a probability to new data



# Why generative models? Debiasing

Capable of uncovering **underlying features** in a dataset



Homogeneous skin color, pose

VS



Diverse skin color, pose, illumination

How can we use this information to create fair and representative datasets?

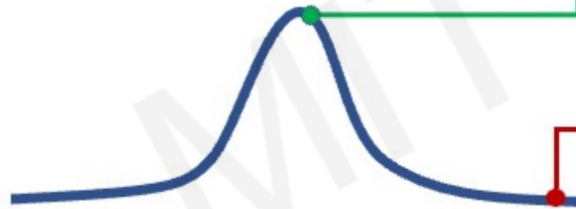
# Why generative models? Outlier detection

- **Problem:** How can we detect when we encounter something new or rare?
- **Strategy:** Leverage generative models, detect outliers in the distribution
- Use outliers during training to improve even more!

**95% of Driving Data:**  
(1) sunny, (2) highway, (3) straight road



Detect outliers to avoid unpredictable behavior when training



Edge Cases



Harsh Weather



Pedestrians

# More outlier examples

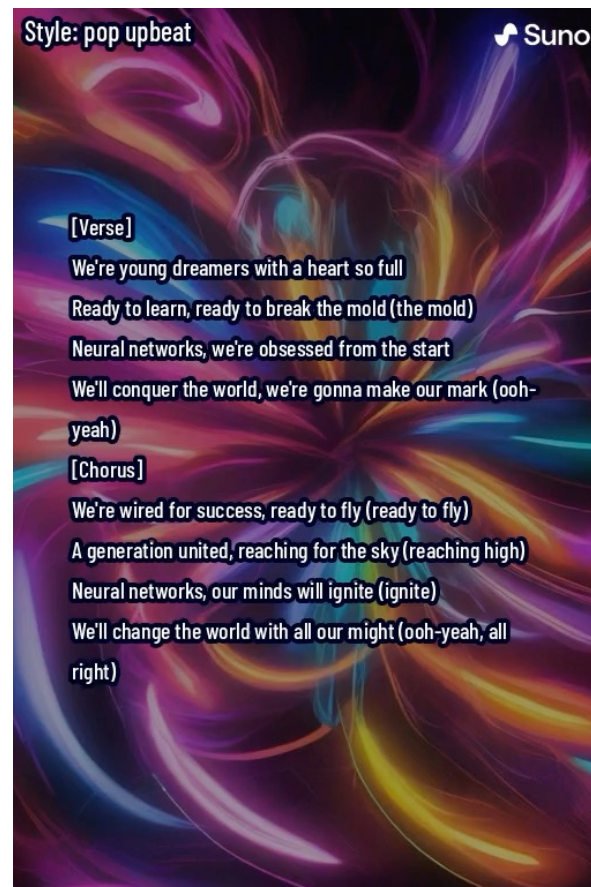


YouTube Video, Feb. 2020 -- <https://www.youtube.com/watch?v=hx7BXih7zx8&t=514s>

## Why generative models? image, video and audio creation



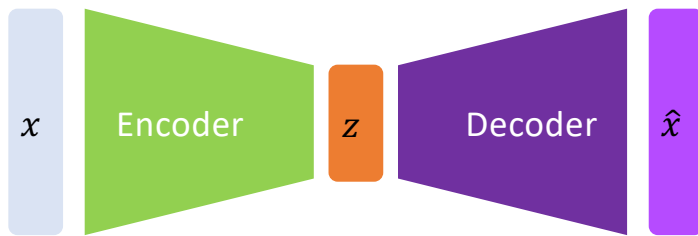
A teenage superhero fighting crime in an urban setting shown in the style of claymation.



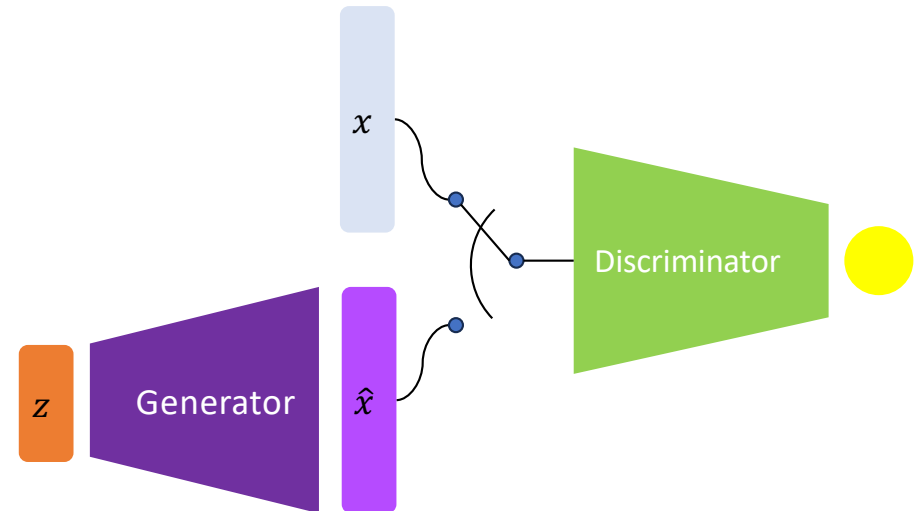
Write a short pop song about students wanting to learn about neural networks and do great things with them.

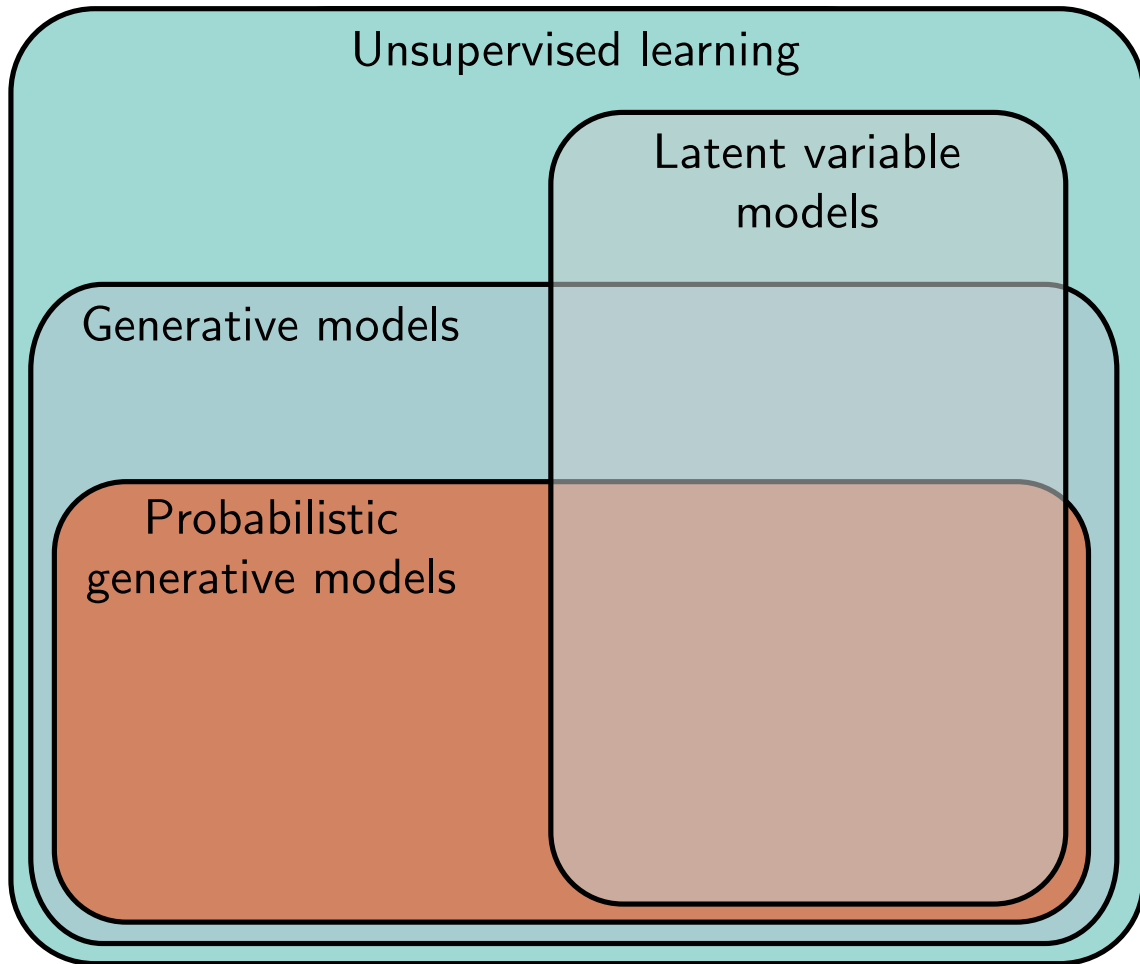
# Latent Variable Models

**Autoencoders and Variational Autoencoders (VAEs)**



**Generative Adversarial Networks**

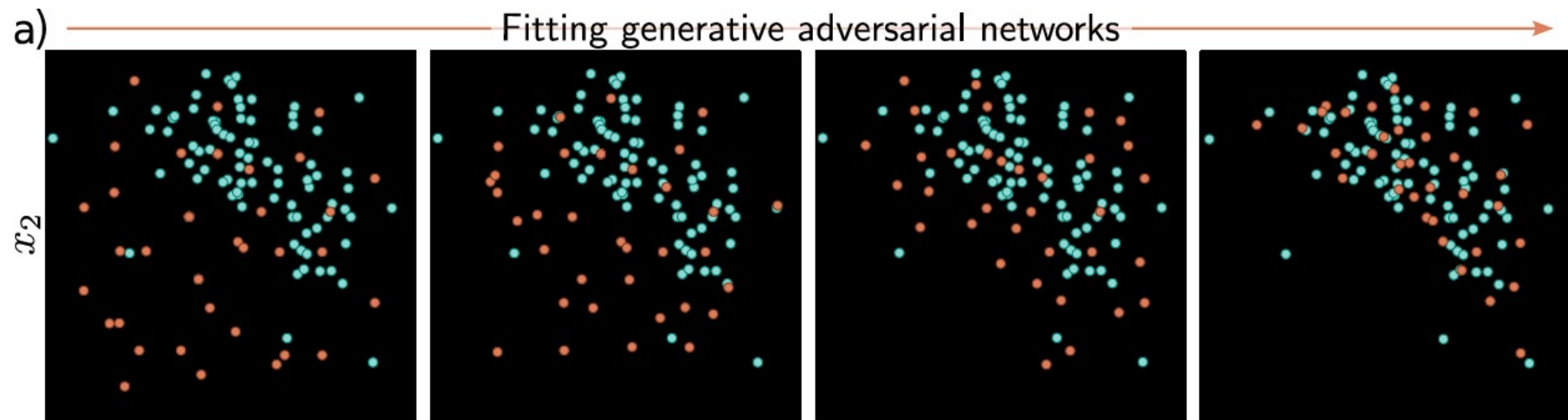




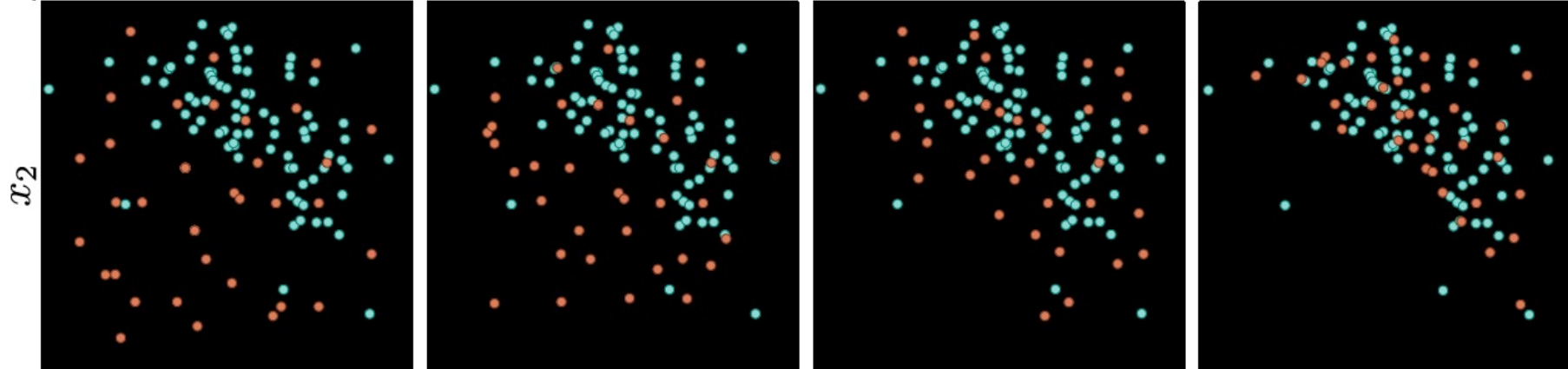
Generative = can generate new examples

Probabilistic = can assign probability to data examples

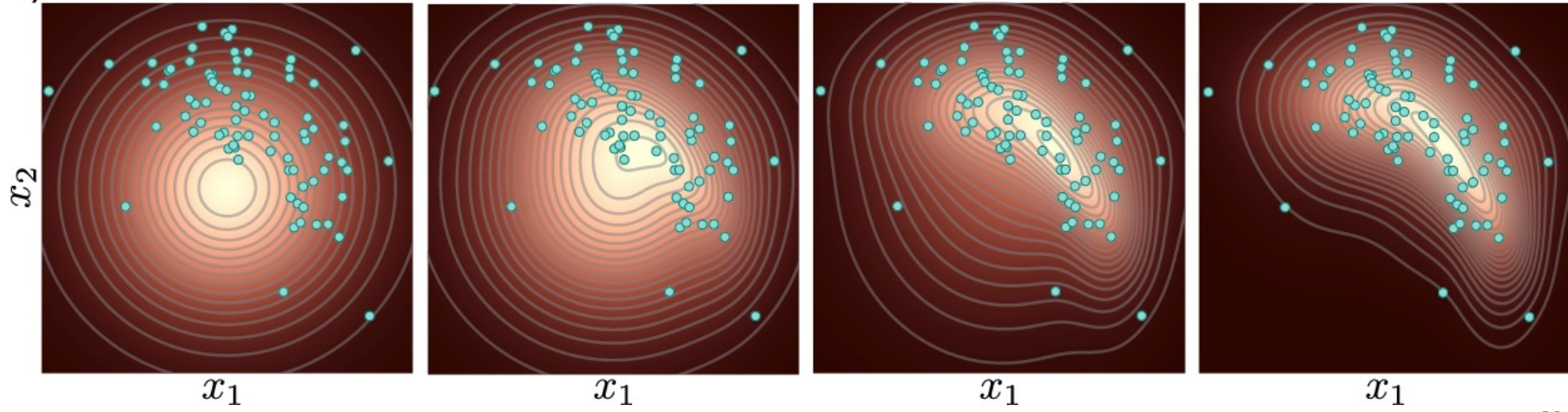




a)  Fitting generative adversarial networks



b)  Fitting normalizing flows, variational auto-encoders, diffusion models





# What makes a good model?

- **Efficient sampling:** Generating samples from the model should be computationally inexpensive and take advantage of the parallelism of modern hardware.
- **High-quality sampling:** The samples should be indistinguishable from the real data that the model was trained with.
- **Coverage:** Samples should represent the entire training distribution. It is insufficient to only generate samples that all look like a subset of the training data.
- **Well-behaved latent space:** Every latent variable  $z$  should correspond to a plausible data example  $x$  and smooth changes in  $z$  should correspond to smooth changes in  $x$ .
- **Interpretable 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 degree of verbosity.
- **Efficient likelihood computation:** If the model is probabilistic, we would like to be able to calculate the probability of new examples efficiently and accurately

# Do we have good models?

|                            | <b>GANs</b> | <b>VAEs</b> | <b>Flows</b> | <b>Diffusion</b> |
|----------------------------|-------------|-------------|--------------|------------------|
| Efficient sampling         | ✓           | ✓           | ✓            | ✗                |
| High quality               | ✓           | ✗           | ✗            | ✓                |
| Coverage                   | ✗           | ?           | ?            | ?                |
| Well-behaved latent space  | ✓           | ✓           | ✓            | ✗                |
| Interpretable latent space | ?           | ?           | ?            | ✗                |
| Efficient likelihood       | n/a         | ✗           | ✓            | ✗                |

How to measure performance within or between categories?

- Open research area.

# “Generative adversarial networks”, Goodfellow et al



RESEARCH-ARTICLE OPEN ACCESS

Generative adversarial networks

Authors: Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio [Authors Info & Claims](#)

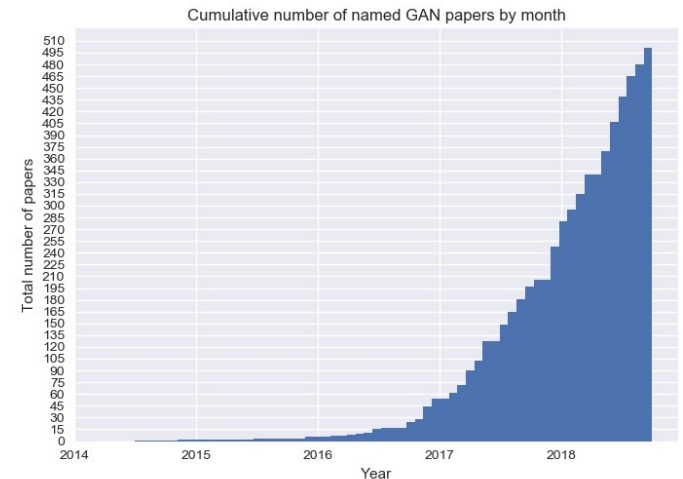
Communications of the ACM, Volume 63, Issue 11 • pp 139–144 • <https://doi.org/10.1145/3422622>

## Ian Goodfellow

PhD ML, U de Montréal 2014

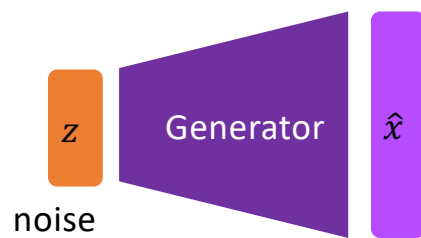
- Google (TensorFlow, Google Brain)
- OpenAI
- Google Staff/Sr. Staff Research Scientist
- Apple Director of ML
- Google Deep Mind, Research Scientist

| TITLE  | CITED BY | YEAR |
|--|----------|------|
| <b>Generative adversarial networks</b><br>I Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, ...<br>Advances in neural information processing systems 27 | 75073 *  | 2014 |
| <b>Deep learning</b><br>I Goodfellow, Y Bengio, A Courville<br>MIT press   | 63352    | 2016 |
| <b>TensorFlow: Large-scale machine learning on heterogeneous systems</b><br>M Abadi, A Agarwal, P Barham, E Brevdo, Z Chen, C Citro, GS Corrado, ...                         | 24052 *  | 2015 |
| <b>Explaining and Harnessing Adversarial Examples</b><br>I Goodfellow, J Shlens, C Szegedy<br>ICLR   | 19914    | 2014 |



[The GAN Zoo \(Github\)](#)

# General Idea of GANs

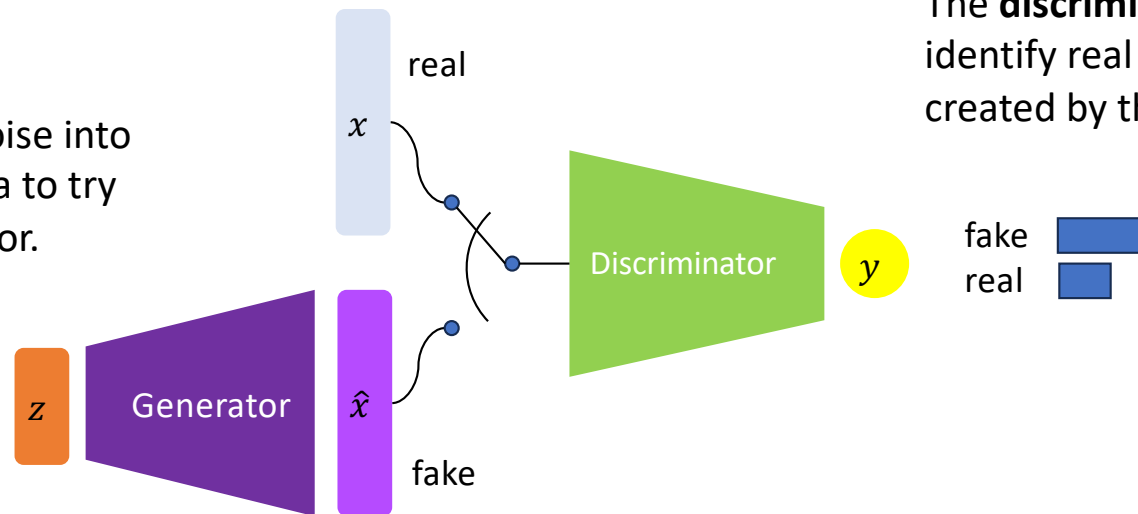


- Don't try to build a probability model directly
- Learn a transformation from a sample of noise to look similar to training data distribution

# Generative Adversarial Networks

Train a generative model to try to fool a “discriminator” model.

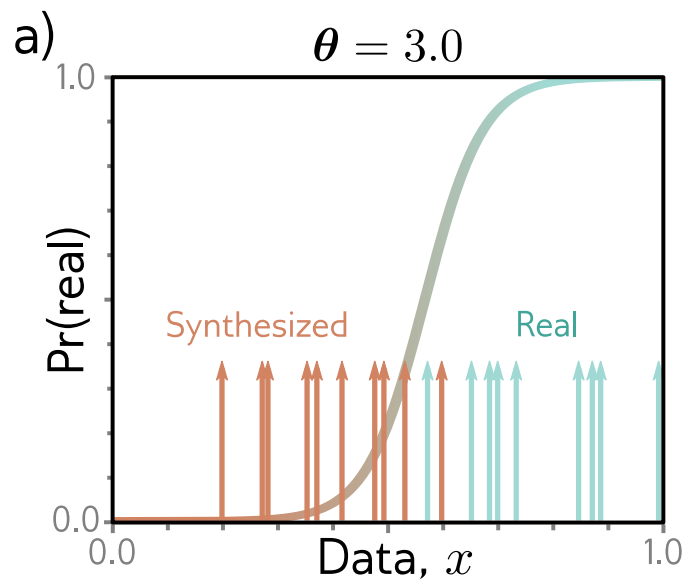
The **generator** turns noise into an imitation of the data to try to trick the discriminator.



The **discriminator** tries to identify real data from fakes created by the generator.

# GAN example

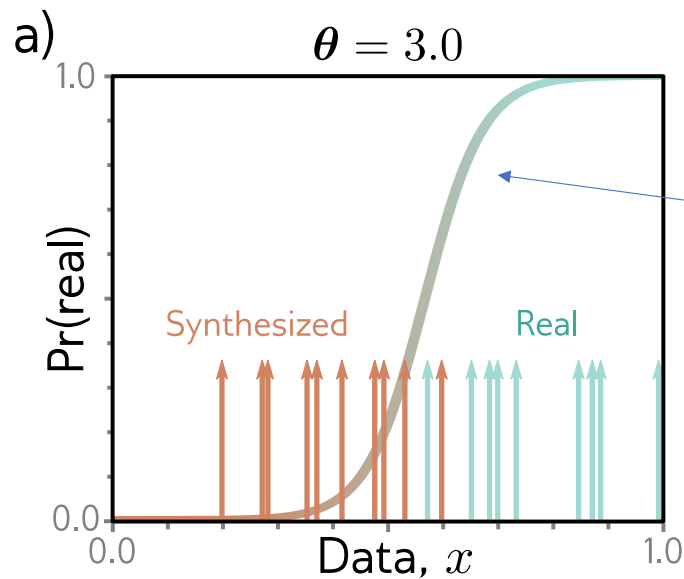
$$x_j^* = g[z_j, \theta] = z_j + \theta$$



- We take examples from a **real** distribution (e.g. shifted standard gaussian)
- We generate **synthesized samples**,  $z_j$ , from a standard gaussian and shift by  $\theta$ .
- Train a classifier on the data

# GAN example

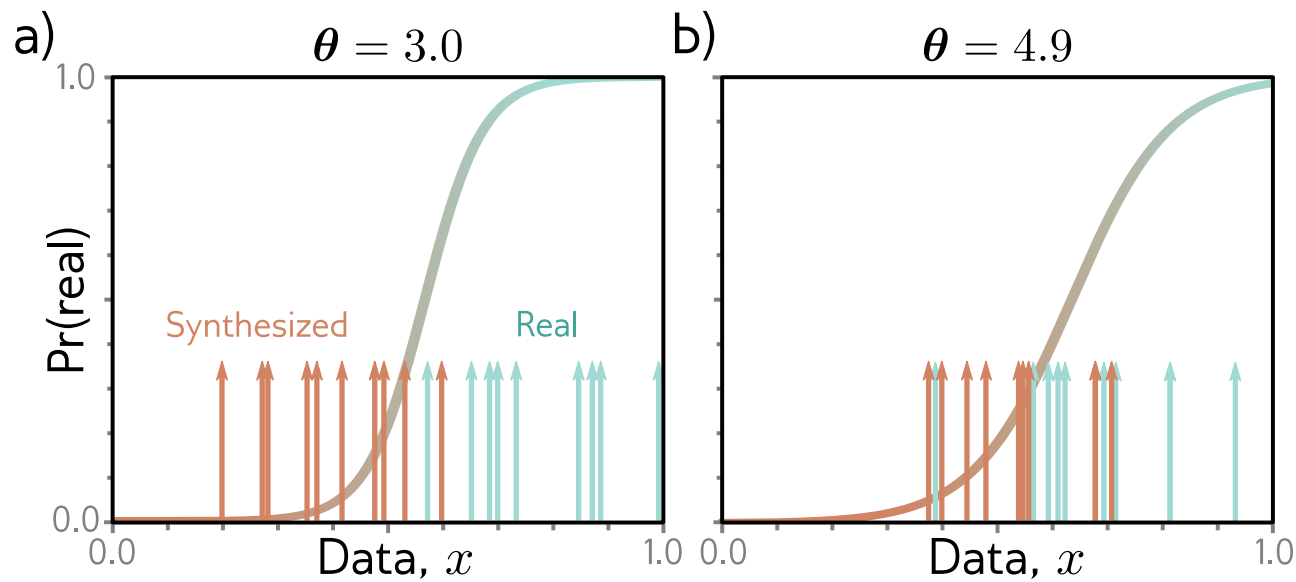
$$x_j^* = g[z_j, \theta] = z_j + \theta$$



- Train the **discriminator**
- using logistic regression parameterized by  $\phi$
- as a binary classifier on the data
- e.g.  $\begin{cases} \text{real if } f[\cdot] \geq .5 \\ \text{fake if } f[\cdot] < .5 \end{cases}$

# GAN example

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



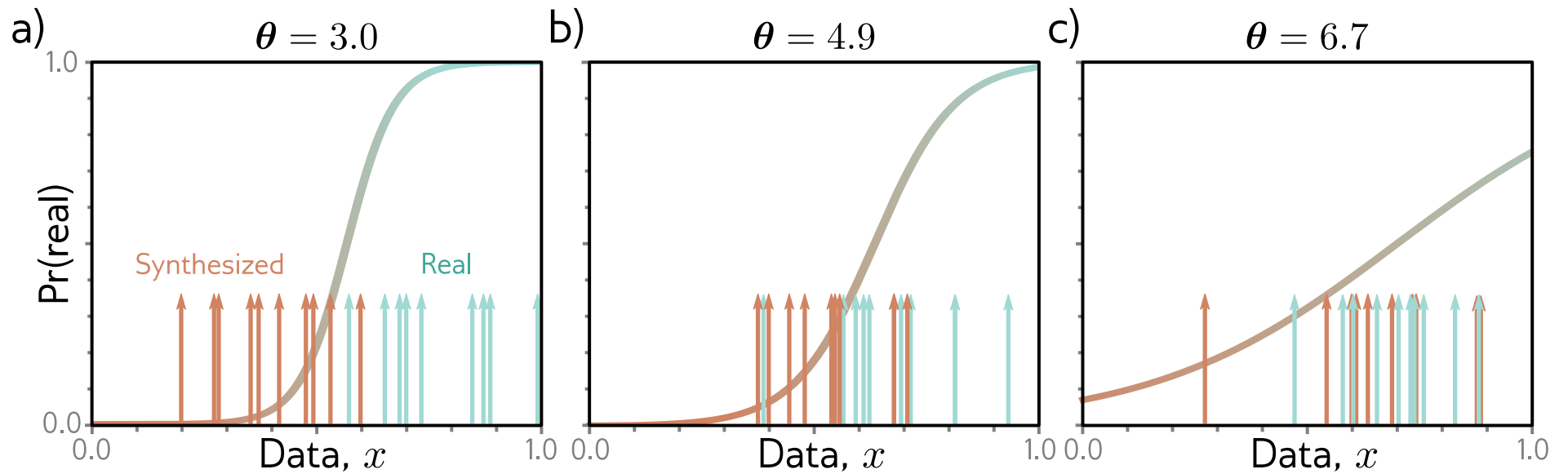
- Train the **generator** to update  $\theta$  in order to *increase* the loss on the discriminator
- Then train the **discriminator** to *decrease* the loss



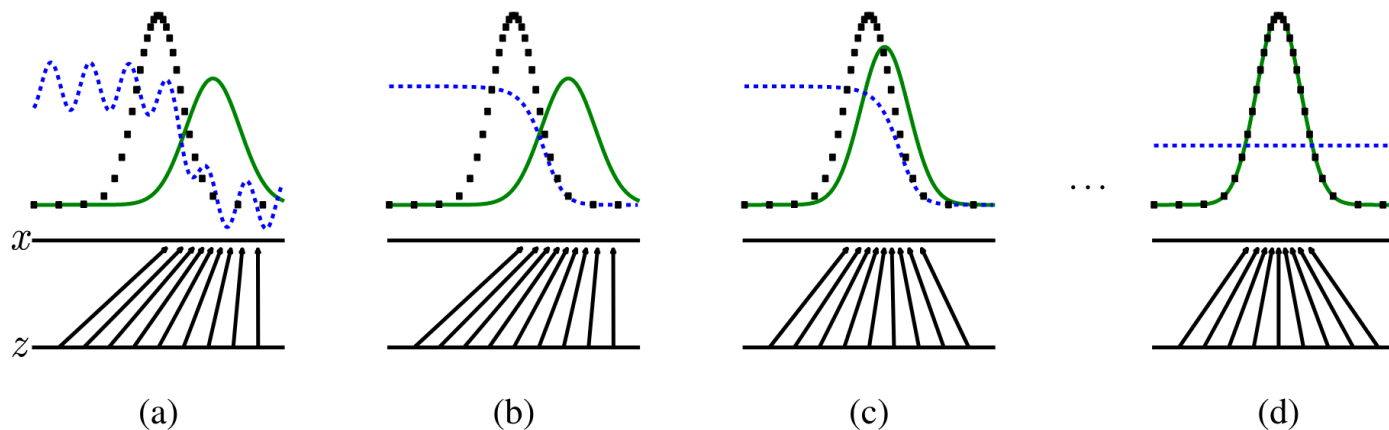
# GAN example

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

- Keep repeating till the discriminator does no better than random chance



# Trained to completion



- $z$ : uniform latent variable
- $x$ : samples according to a (green solid) generative distribution
- black dotted curve: real data distribution
- blue dashed curve: discriminator

## 4.1 Global Optimality of $p_g = p_{\text{data}}$

We first consider the optimal discriminator  $D$  for any given generator  $G$ .

**Proposition 1.** For  $G$  fixed, the optimal discriminator  $D$  is

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}$$

# GANs

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

# GAN cost function

**Discriminator** uses standard cross entropy loss (see Section 5.4 – binary classification loss):

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

# GAN cost function

**Discriminator** uses standard cross entropy loss (see Section 5.4 – binary classification loss):

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

Generated samples,  $\mathbf{x}_i^*$ ,  $y_i = 0$ , and for real examples,  $\mathbf{x}_i$ ,  $y_i = 1$  :

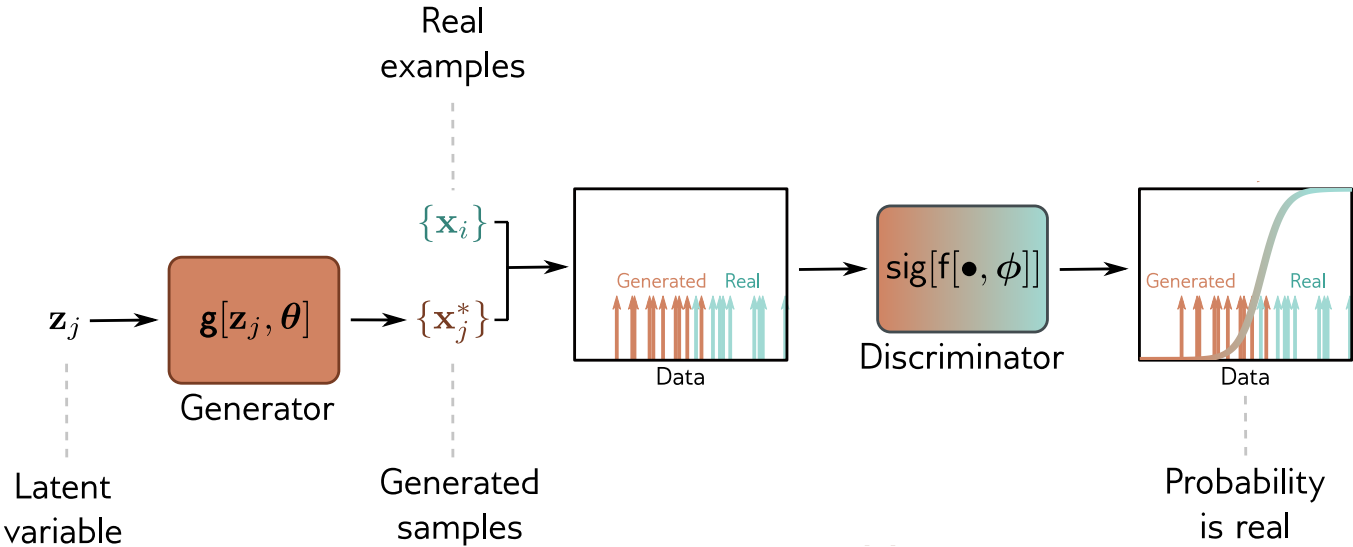
$$\hat{\phi} = \operatorname{argmin}_{\phi} \left[ \sum_j -\log [1 - \operatorname{sig}[f[\mathbf{x}_j^*, \phi]]] - \sum_i \log [\operatorname{sig}[f[\mathbf{x}_i, \phi]]] \right]$$

These are *generated*  
samples so  $y_j = 0$

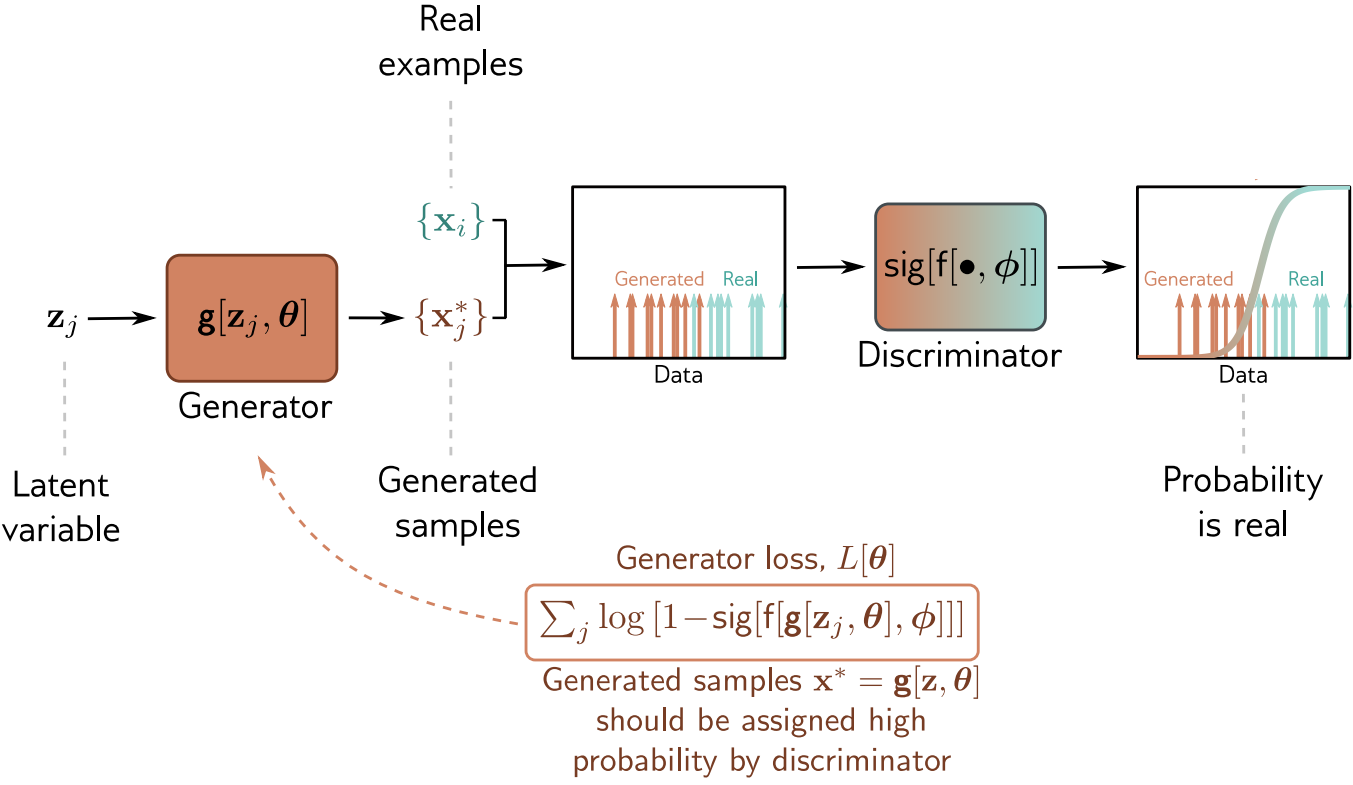
These are *real* samples  
so  $y_i = 1$

We can separate into two summations that separately index over the generated samples and the real samples.

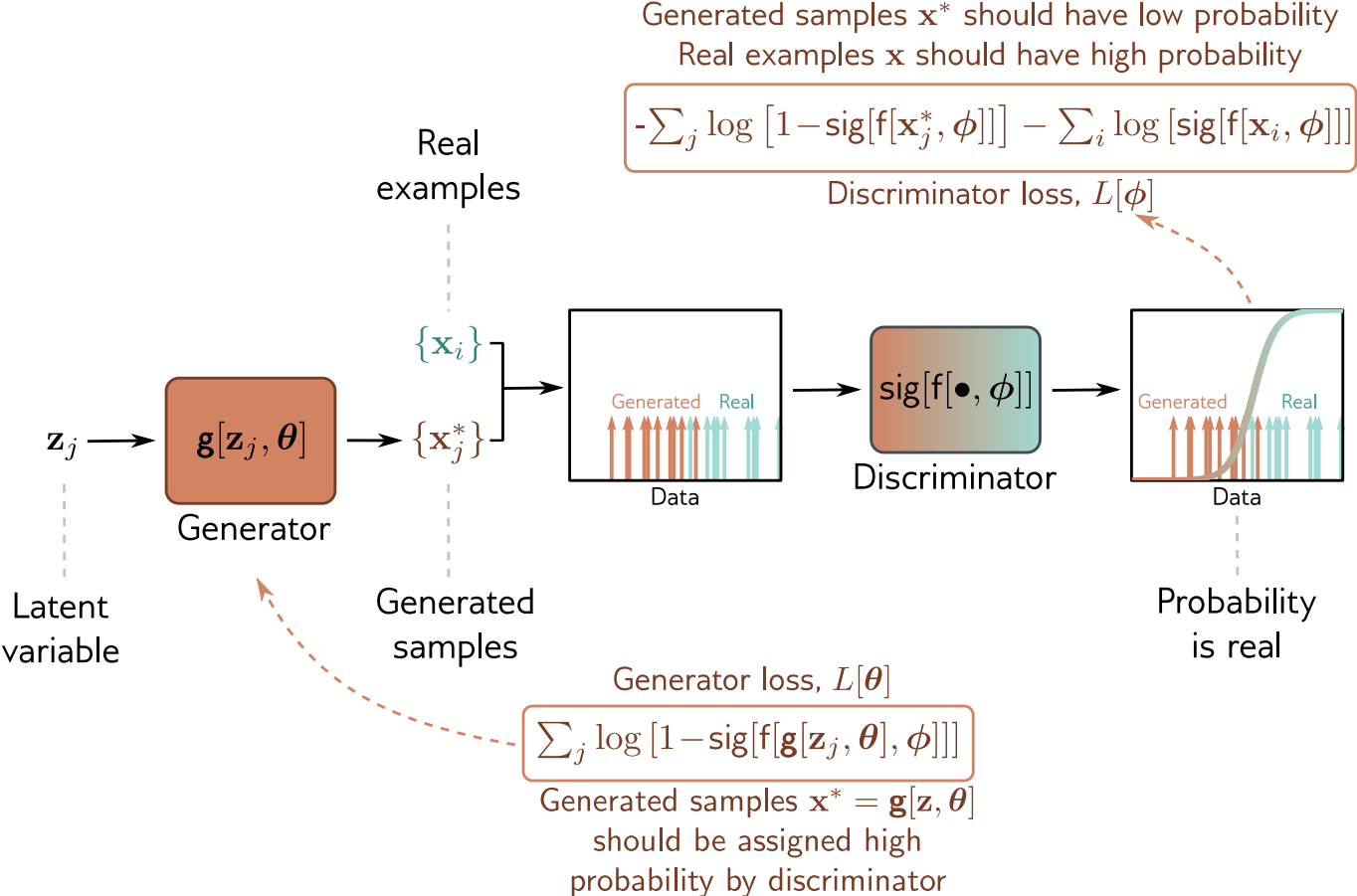
# GAN loss function



# GAN loss function



# GAN loss function





# GAN cost function

Discriminator uses standard cross entropy loss:

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

Discriminator: generated samples,  $y = 0$ , real examples,  $y = 1$ :

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

Generator loss: make generated samples more likely under discriminator (i.e. make discriminator loss larger)

$$\hat{\phi}, \hat{\theta} = \operatorname{argmax}_{\theta} \left[ \operatorname{argmin}_{\phi} \left[ \sum_j -\log [1 - \operatorname{sig}[f[\underbrace{\mathbf{g}[\mathbf{z}_j, \theta]}_{\text{substituted the generator function for the generated sample}}, \phi]]] - \sum_i \log [\operatorname{sig}[f[\mathbf{x}_i, \phi]]] \right] \right]$$

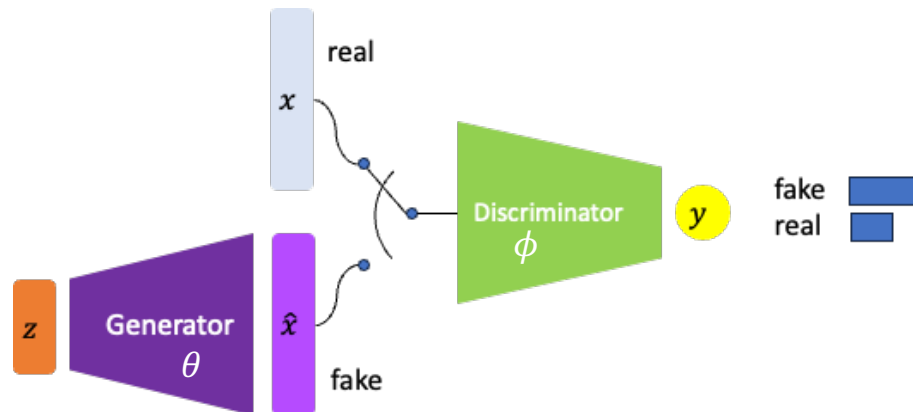
substituted the generator function  
for the generated sample

# GAN Cost function

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

The **discriminator** parameters,  $\phi$ , are manipulated to *minimize* the loss function

The **generator** parameters,  $\theta$ , are manipulated to *maximize* the loss function.



# GAN Cost function

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

The **discriminator** parameters,  $\phi$ , are manipulated to *minimize* the loss function

The **generator** parameters,  $\theta$ , are manipulated to *maximize* the loss function.

---

Can divide into two parts:

**discriminator** loss:  $L[\phi] = \sum_j -\log \left[ 1 - \operatorname{sig}[f[\mathbf{g}[\mathbf{z}_j, \theta], \phi]] \right] - \sum_i \log \left[ \operatorname{sig}[f[\mathbf{x}_i, \phi]] \right]$

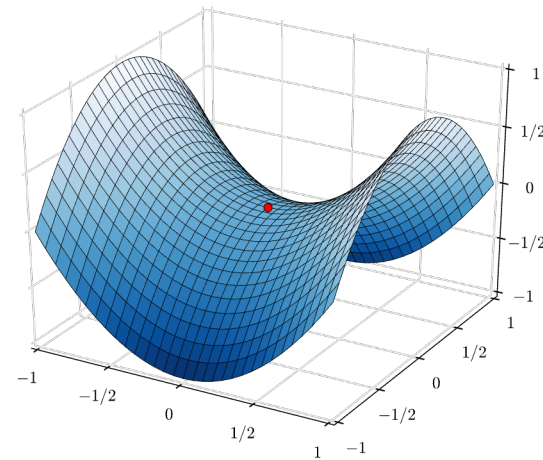
negated **generator** loss:  $L[\theta] = \sum_j \log \left[ 1 - \operatorname{sig}[f[\mathbf{g}[\mathbf{z}_j, \theta], \phi]] \right]$

The 2<sup>nd</sup> term is constant w.r.t.  $\theta$   
(gradient  $\partial \mathcal{L} / \partial \theta = 0$ ) so we can drop it)

# GAN Solution

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

- The solution is the *Nash equilibrium*
- It lays at a saddle point
- Is inherently unstable



## Nash equilibrium

In game theory, the Nash equilibrium, named after the mathematician John Nash, is the most common way to define the solution of a non-cooperative game involving two or more players.

...each player is assumed to know the equilibrium strategies of the other players, and no one has anything to gain by changing only one's own strategy. [Wikipedia](#)



# GAN Training Flow Pseudo Python

```
for c_gan_iter in range(n_gan_iters): # GAN Iterations

    # Run generator to produce synthesized data
    x_syn = generator(z, theta)

    # Update/train the discriminator
    phi = update_discriminator(x_real, x_syn, n_iter_discrim, phi)

    # Update/train the generator
    theta = update_generator(z, theta, n_iter_gen, phi)
```

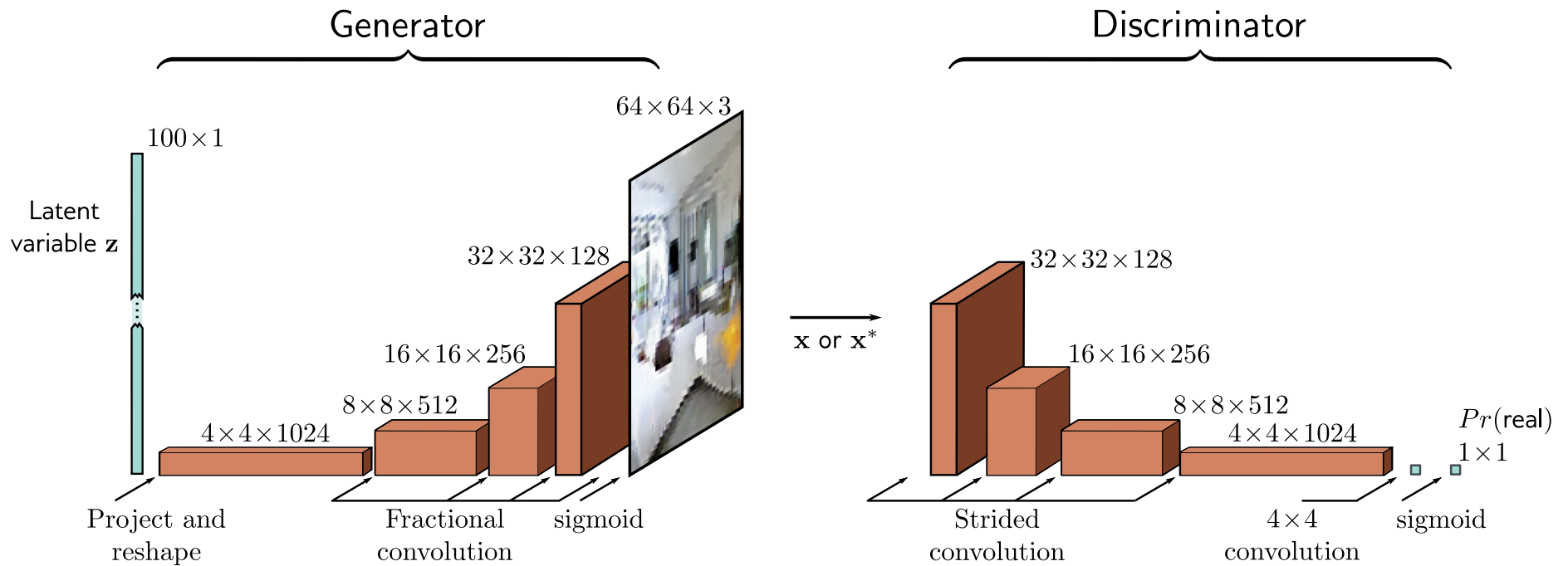
See Jupyter Notebook assignment (to be released shortly)

# GANs

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

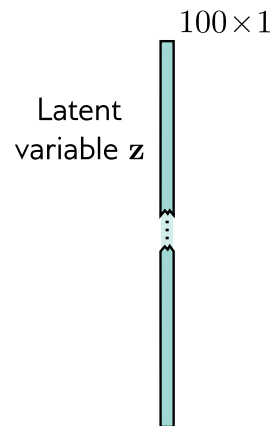
# Deep Convolutional (DC) GAN

- Early GAN specialized in image generation



# DCGAN -- Generator

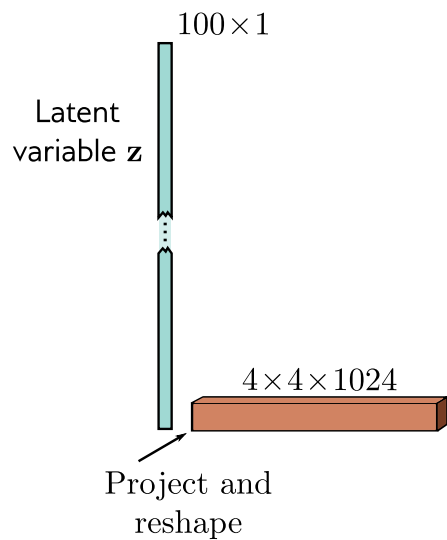
- Input is 100D latent variable,  $z$ , drawn from a uniform distribution



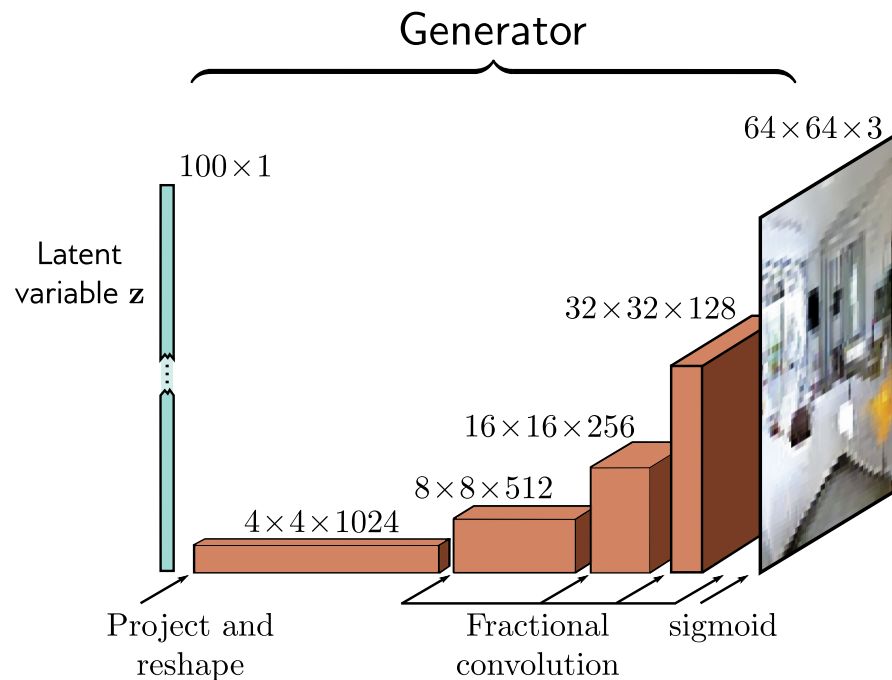


# DCGAN -- Generator

- Input is 100D latent variable,  $z$ , drawn from a uniform distribution
- Maps to  $4 \times 4 \times 1024$  via a linear transformation



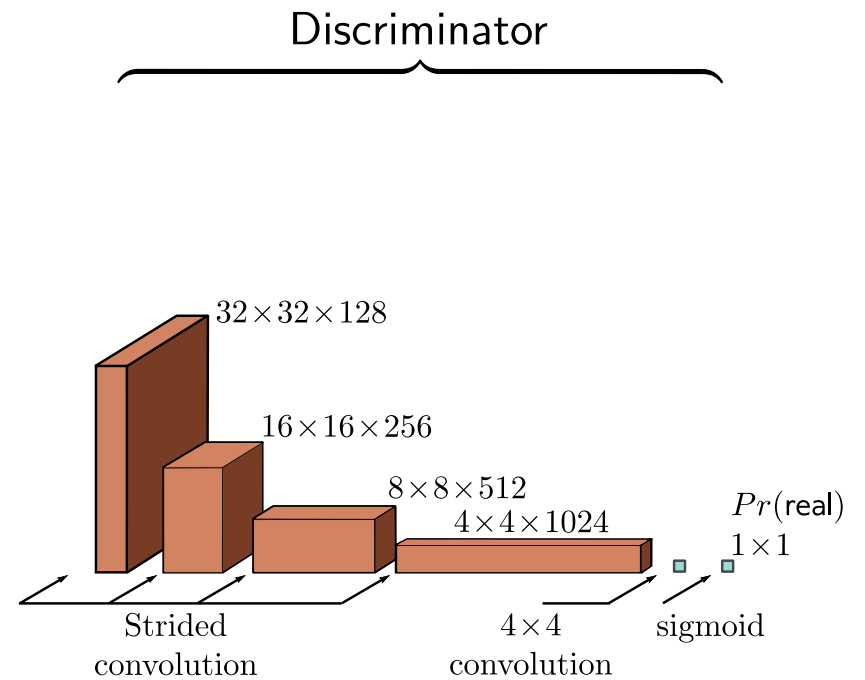
# DCGAN -- Generator



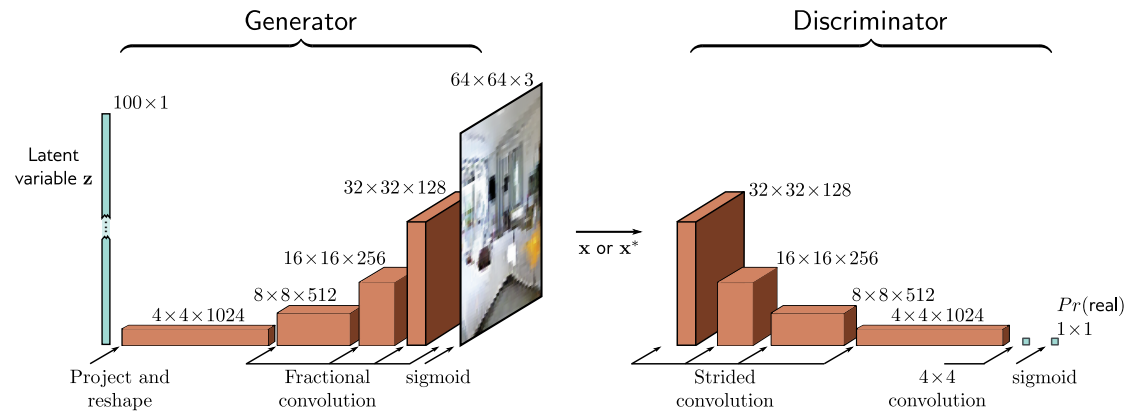
- Input is 100D latent variable,  $z$ , drawn from a uniform distribution
- Maps to  $4 \times 4 \times 1024$  via a linear transformation
- Fractionally strided (stride = 0.5) convolutions to double resolution in each dimension
- Final tanh to limit to  $[-1,1]$
- Rescaled to  $[0,255]$

# DCGAN -- Discriminator

- Standard convolution network
- Reduces to 1x1
- Final sigmoid to create output probability



# Deep Convolutional (DC) GAN



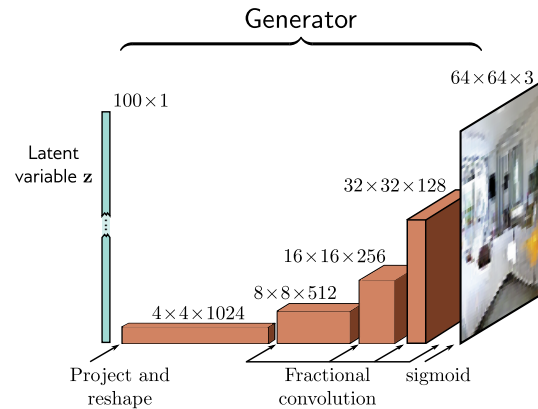
Trained as in the earlier example.

```
for c_gan_iter in range(5): # GAN Iterations
    # Run generator to produce synthesized data
    x_syn = generator(z, theta)

    # Update/train the discriminator
    phi = update_discriminator(x_real, x_syn, n_iter_discrim, phi)

    # Update/train the generator
    theta = update_generator(z, theta, n_iter_gen, phi)
```

# Deep Convolutional (DC) GAN



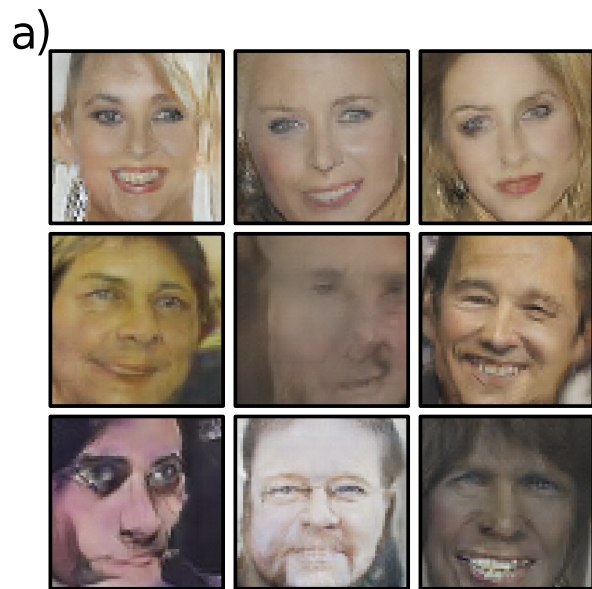
When training is complete

Discard discriminator

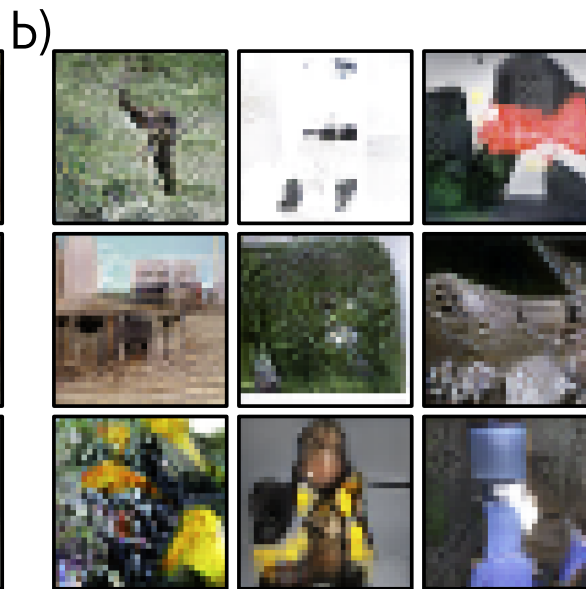
Draw new latent variable

Pass through generator

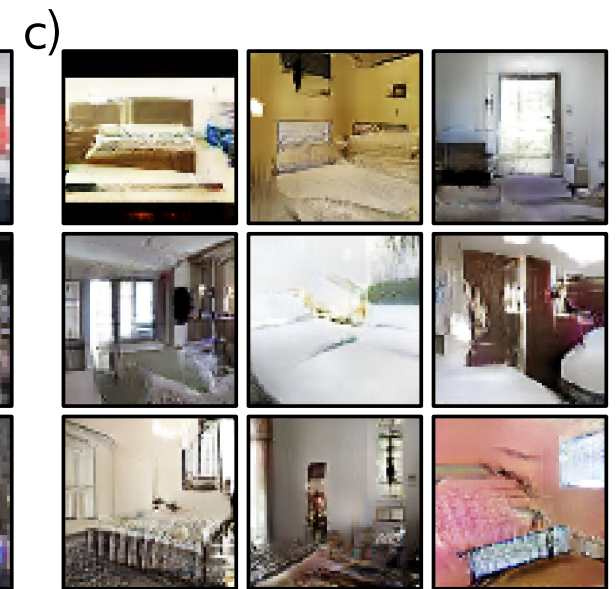
# DC GAN Results



Trained on a faces dataset.



Trained on ImageNet dataset.



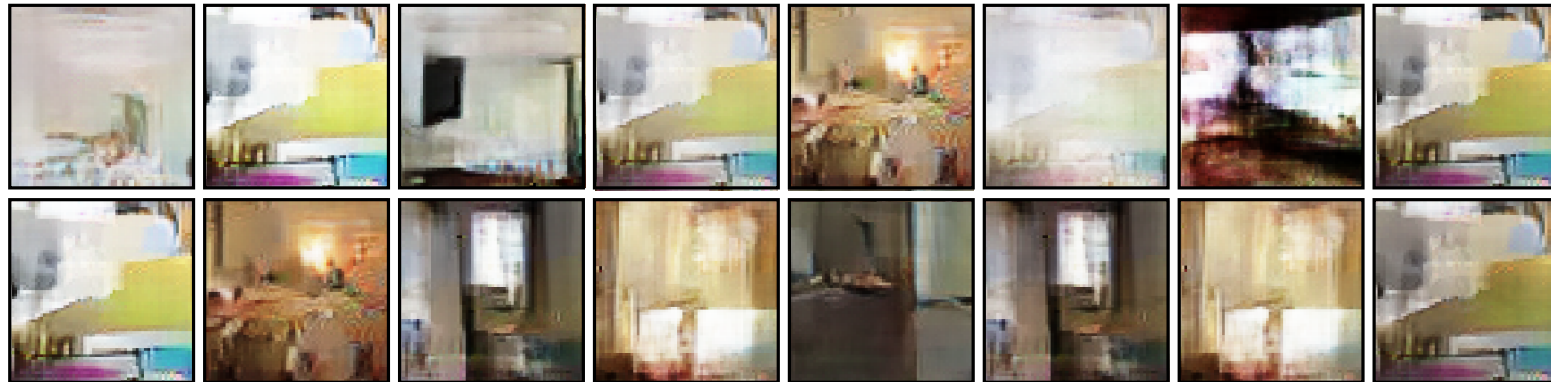
Trained on LSUN dataset.

The LSUN classification dataset contains 10 scene categories, such as dining room, bedroom, chicken, outdoor church, and so on. <sup>50</sup>

# Common Failures with GANs

**Mode Dropping:** Only represent a subset of the training distribution.

**Mode Collapse:** Extreme case where the generator mostly ignores the latent variable and collapses all samples to a few points



# GAN Performance and Distribution Distance

$$D_{JS}[Pr(\mathbf{x}^*) \parallel Pr(\mathbf{x})] = \underbrace{\frac{1}{2} D_{KL} \left[ Pr(\mathbf{x}^*) \parallel \frac{Pr(\mathbf{x}^*) + Pr(\mathbf{x})}{2} \right]}_{\text{quality}} + \underbrace{\frac{1}{2} D_{KL} \left[ Pr(\mathbf{x}) \parallel \frac{Pr(\mathbf{x}^*) + Pr(\mathbf{x})}{2} \right]}_{\text{coverage}}$$

Summary of lengthy analysis in §15.2.1 “Analysis of GAN loss function”

Can be rewritten in terms of dissimilarities between *generated* and *real* probability distributions.

Two important takeaways:

**Quality:** Generated samples need to occur where real samples are

**Coverage:** Where there is concentrations of real samples, there should be good representation from generated samples



We can conclude that:

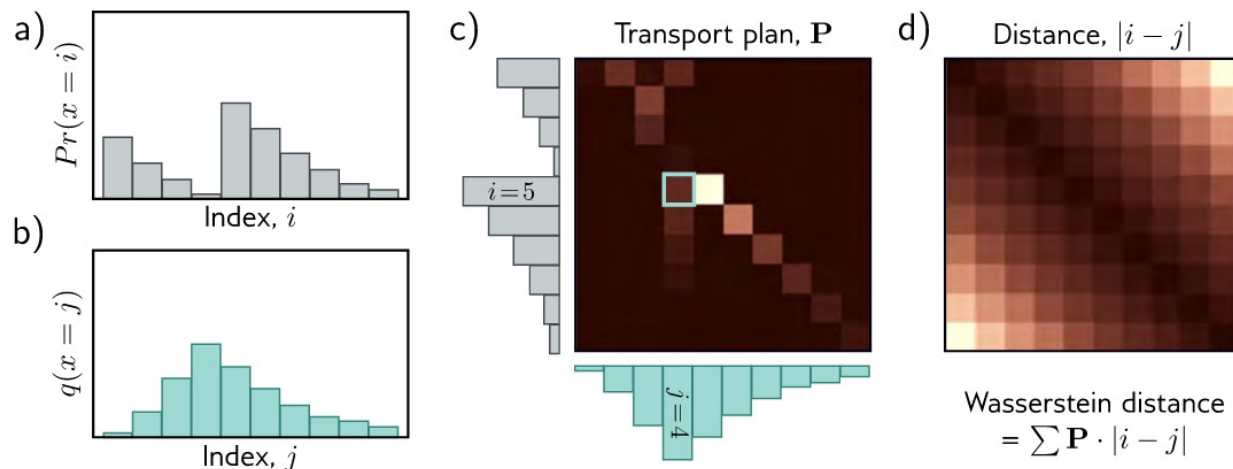
(i) the GAN loss can be interpreted in terms of distances between probability distributions and that

(ii) the gradient of this distance becomes zero when the generated samples are too easy to distinguish from the real examples.

We need a distance metric with better properties.

# Wassertein Distance (for continuous distributions) Earth Mover's Distance (for discrete probabilities)

- The quantity of work required to transport the probability mass from one distribution to create the other.
- Use linear programming to find an optimal “transport plan” that minimizes  $\sum \mathbf{P} \cdot |i - j|$

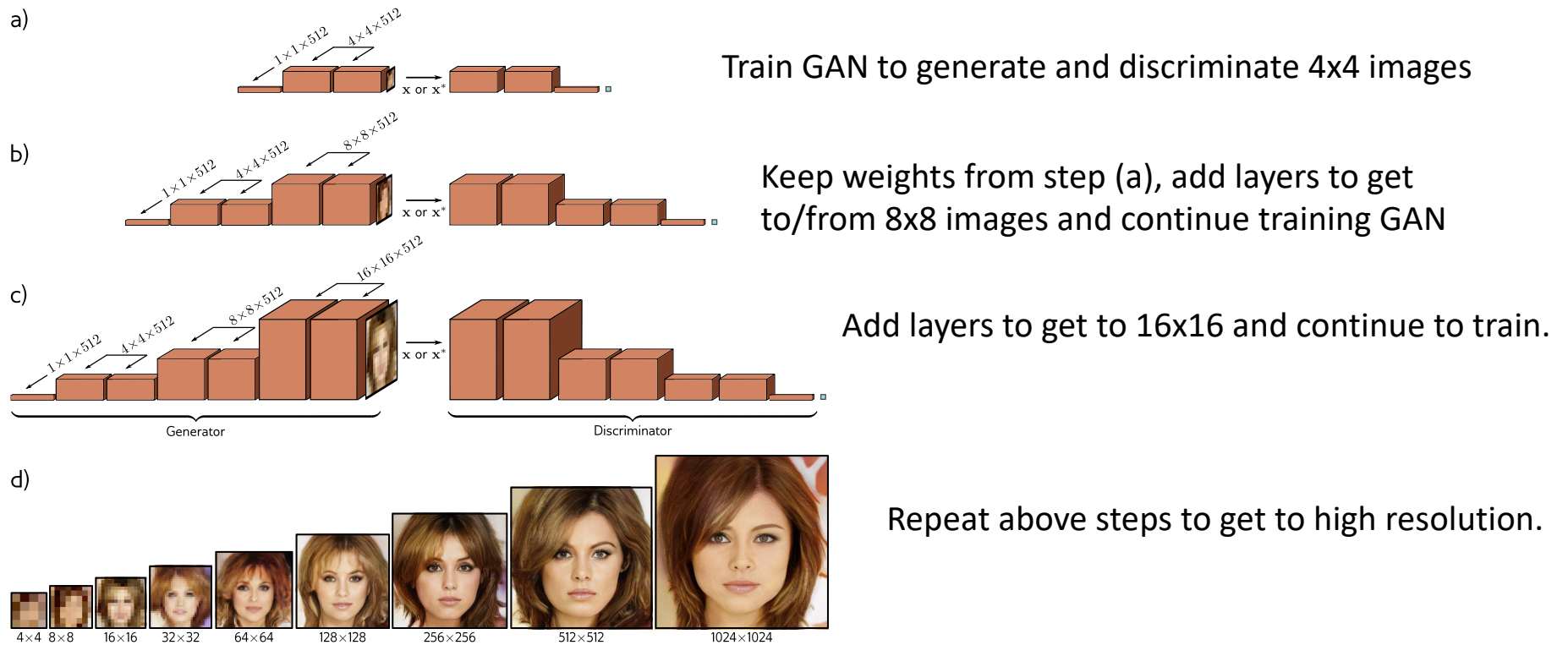


See 15.2.4 Wasserstein distance for discrete distributions

# GANs

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

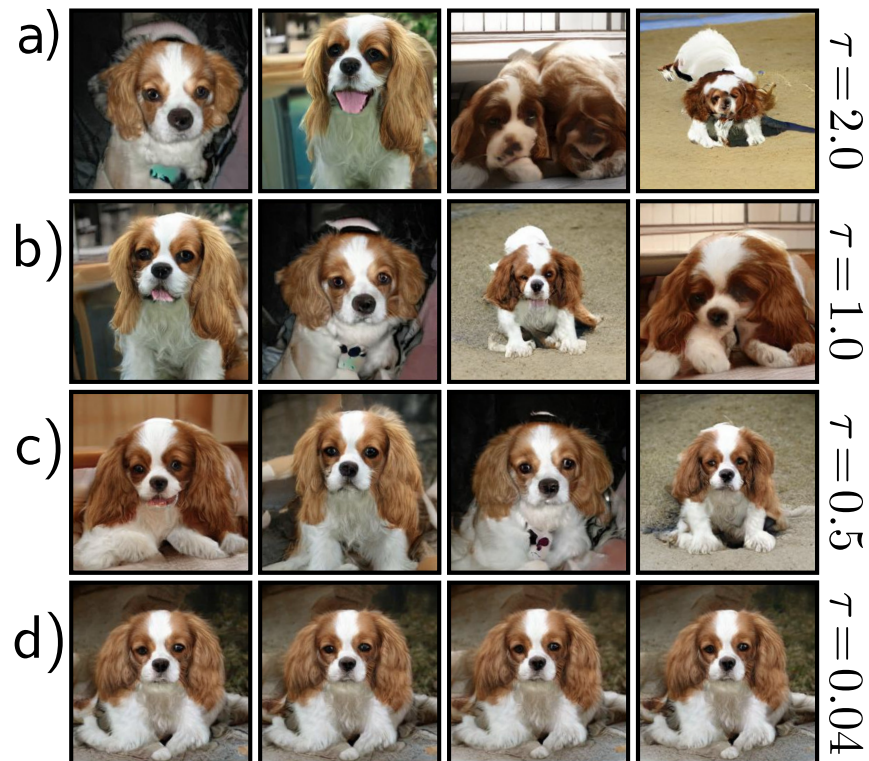
# Trick 1: Progressive growing



## Trick 2: Minibatch discrimination

- Add in statistics across minibatches of synthesized and real data
- Provided to the discriminator as an additional feature map
- Sends signal back to generator to try to better match real batch statistics

## Trick 3: Truncation



- Only choose random values of latent variables that are less than a threshold  $\tau$  distance from the mean of the latent variables.
- Reduces variation but improves quality

# Interpolation

**Well-behaved latent space:** Every latent variable  $z$  should correspond to a plausible data example  $x$  and smooth changes in  $z$  should correspond to smooth changes in  $x$ .



# Interpolation

**Well-behaved latent space:** Every latent variable  $z$  should correspond to a plausible data example  $x$  and smooth changes in  $z$  should correspond to smooth changes in  $x$ .





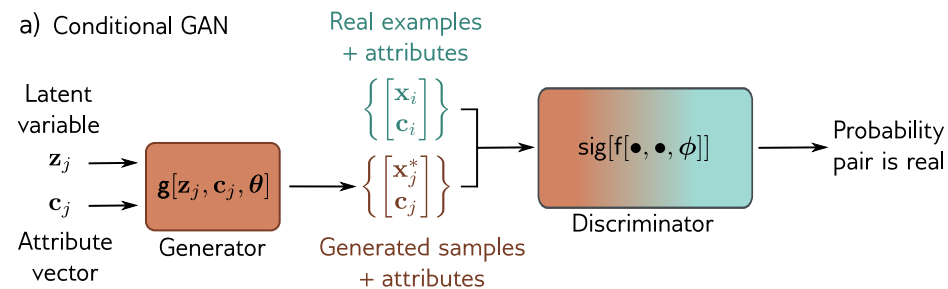
# GANs

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

## Lack of control

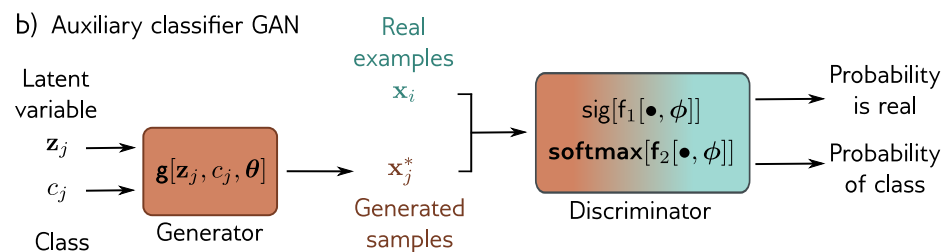
- Cannot specify attributes of generated images from vanilla GANs
- E.g. can't choose ethnicity, age, etc., for a GAN trained on faces.
- *Conditional generation* models provide this control

# Conditional GAN models



- Passes a vector  $\mathbf{c}$  of attributes to both the generator and discriminator
- Generator learns to generate sample with correct attribute
- Discriminator learns to distinguish between generated sample with target attribute and real sample with real attribute

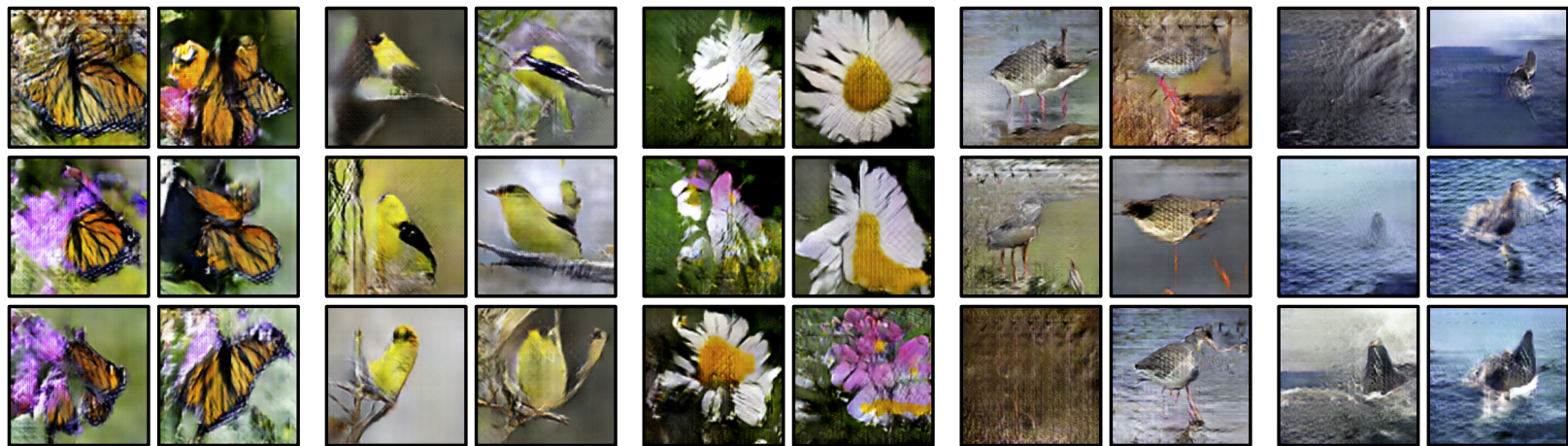
# Auxiliary classifier GAN



- Similar to Conditional GAN, but use class label instead of attribute vector
- Discriminator produces:
  - Binary real/fake classifier
  - Multi-class classifier

# Auxiliary Classifier GAN results

Trained on ImageNet images and classes.



monarch butterfly

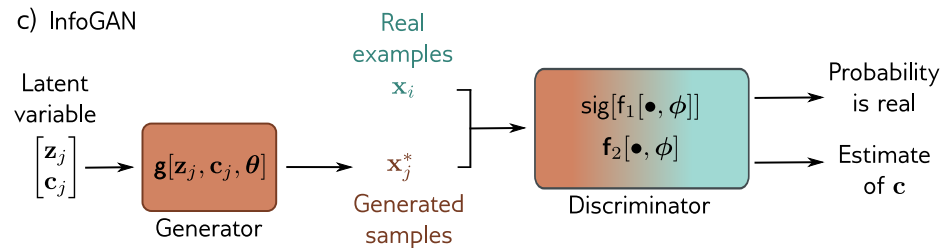
goldfinch

daisies

redshanks

gray whales

# InfoGAN

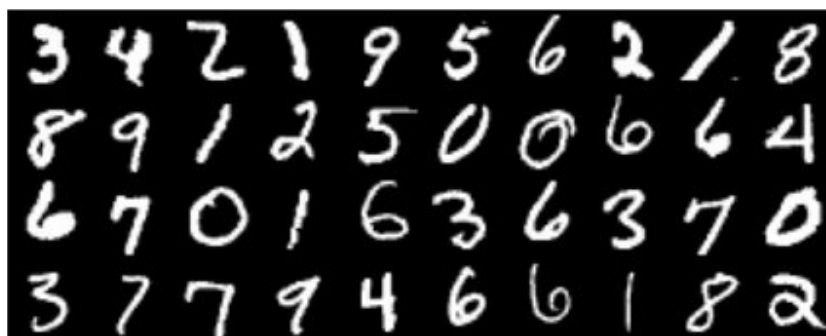


- Add random attribute variables  $\mathbf{c}$  to generator
- Discriminator learns to predict discrete and continuous values of the attributes

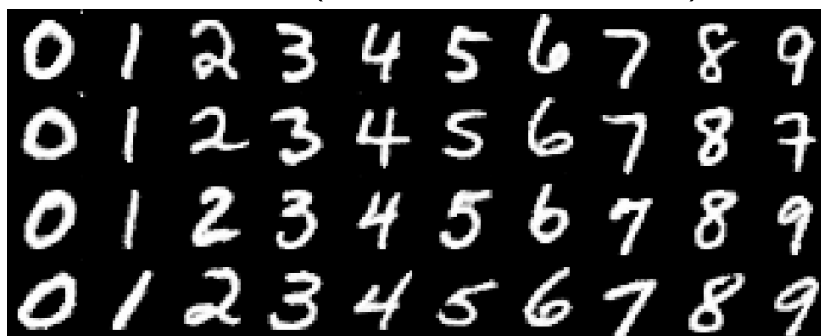
# InfoGAN results

Learns classes

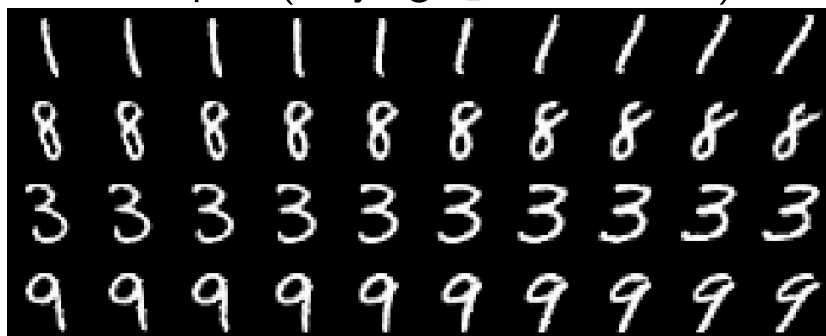
a) MNIST training data



b) Samples (varying  $c_1$ , discrete)

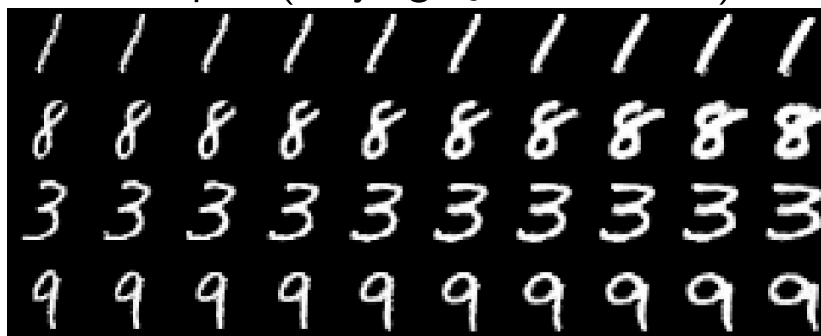


c) Samples (varying  $c_2$ , continuous)



Learns orientation

d) Samples (varying  $c_3$ , continuous)



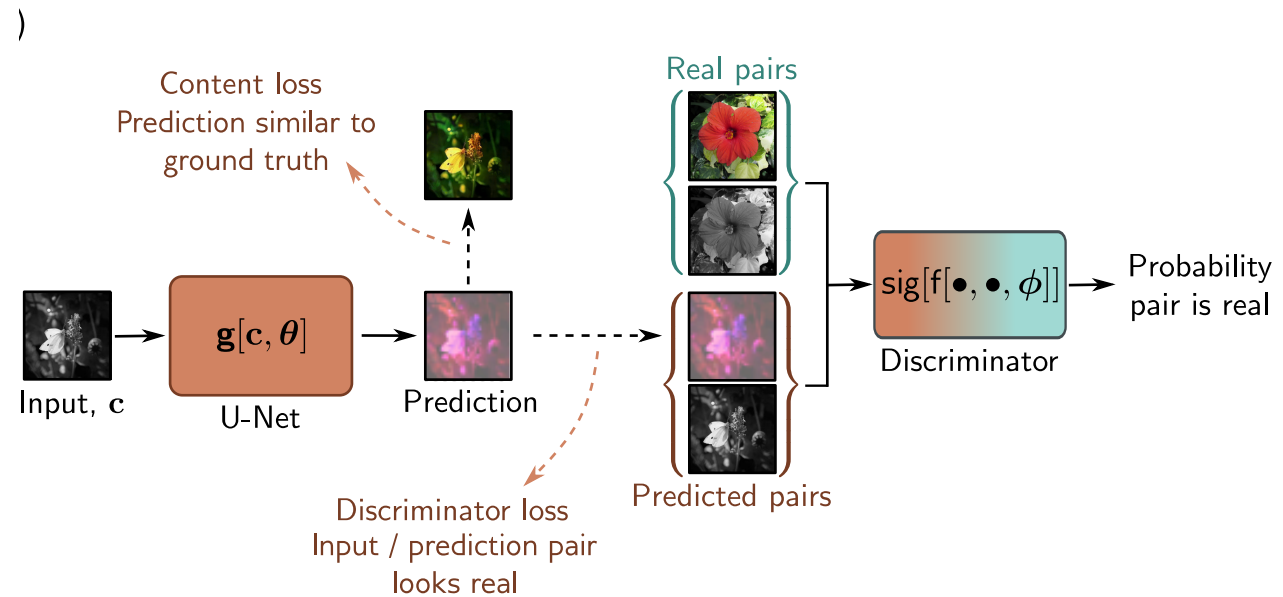
Learns stroke thickness

# GANs

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

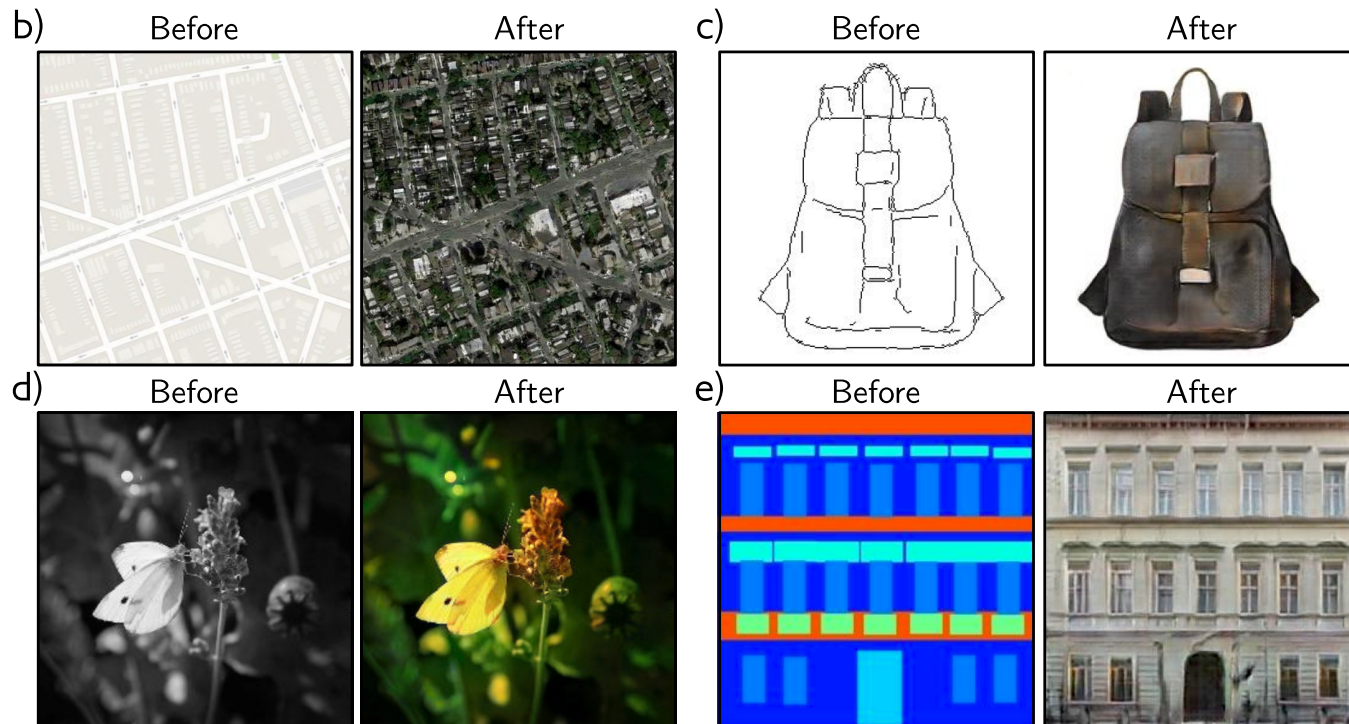


# Image translation: Pix2Pix



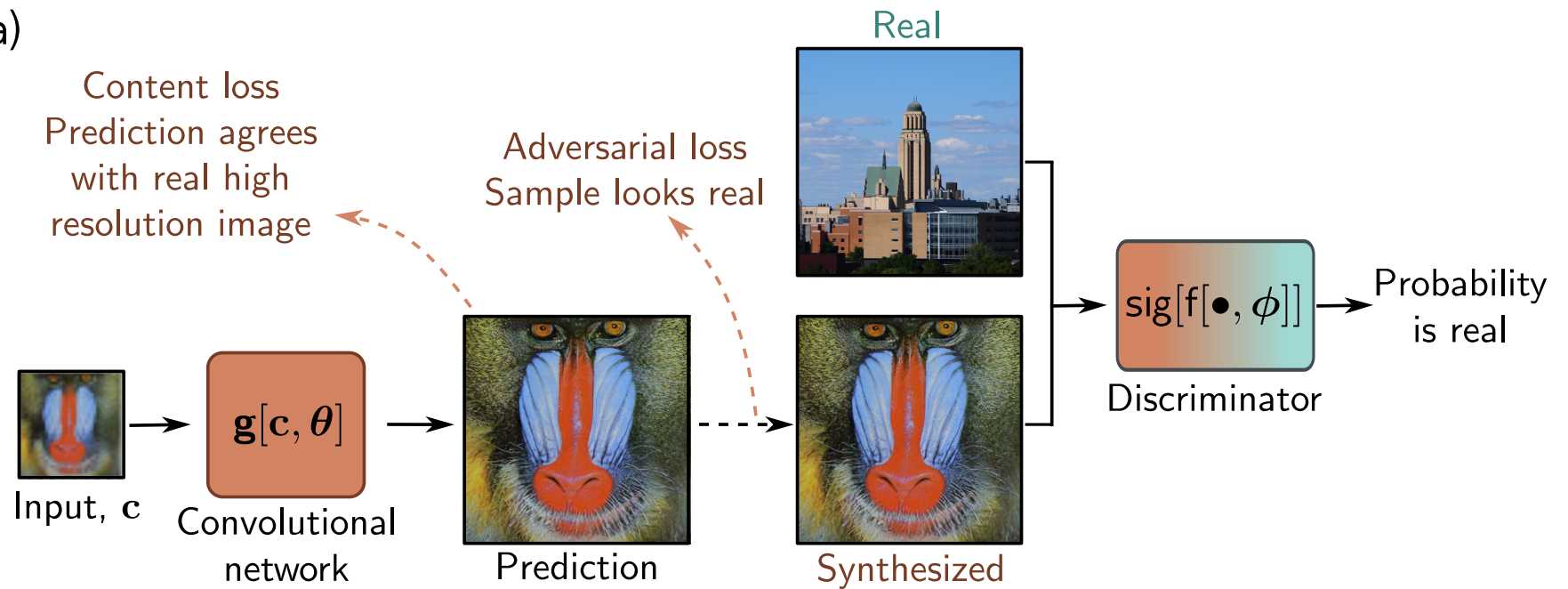
- Maps one image to a different style image using a U-Net type model
- Adds a content loss ( $\ell_1$  norm) to make the input similar to ground truth
- Discriminator fed input/prediction and real/modified pairs to predict real or fake

# Image translation: Pix2Pix



# Image translation: SRGAN

a)

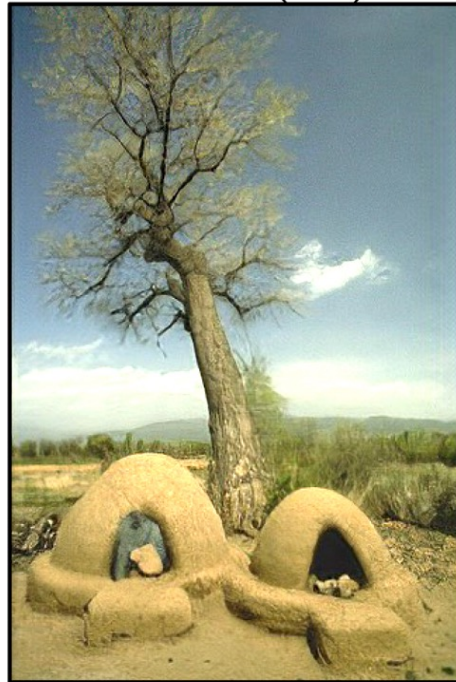


# Image translation: SRGAN

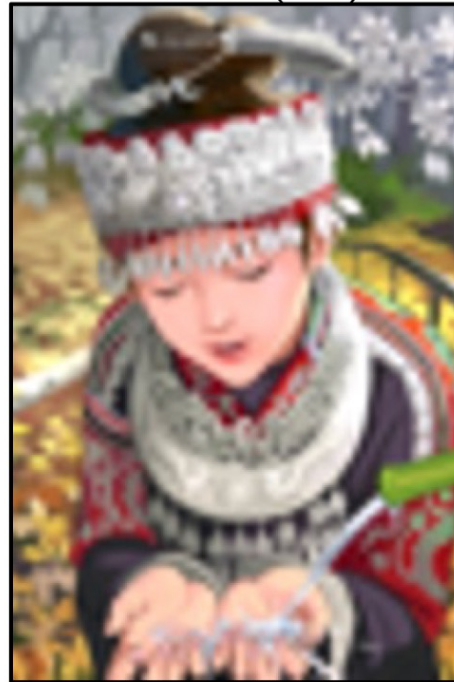
b) Bicubic (4×)



c) SRGAN (4×)



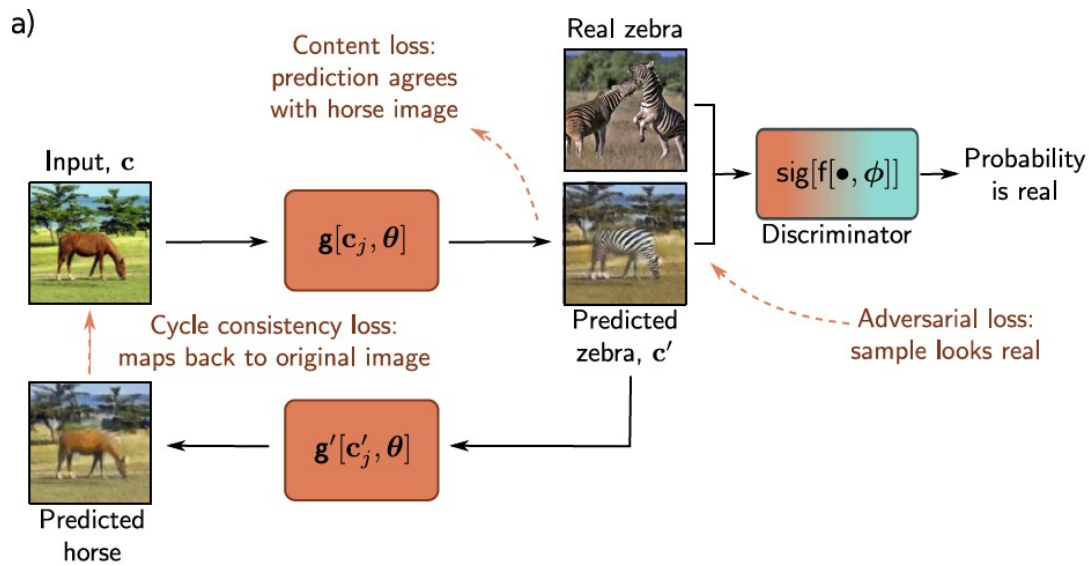
d) Bicubic (4×)



e) SRGAN (4×)



# Image translation: CycleGAN



2<sup>nd</sup> model is also trained.

Encourages the generator to be reversible

Doesn't need labeled or matched training pairs.

Have two sets of data with distinct styles but no matching pairs.

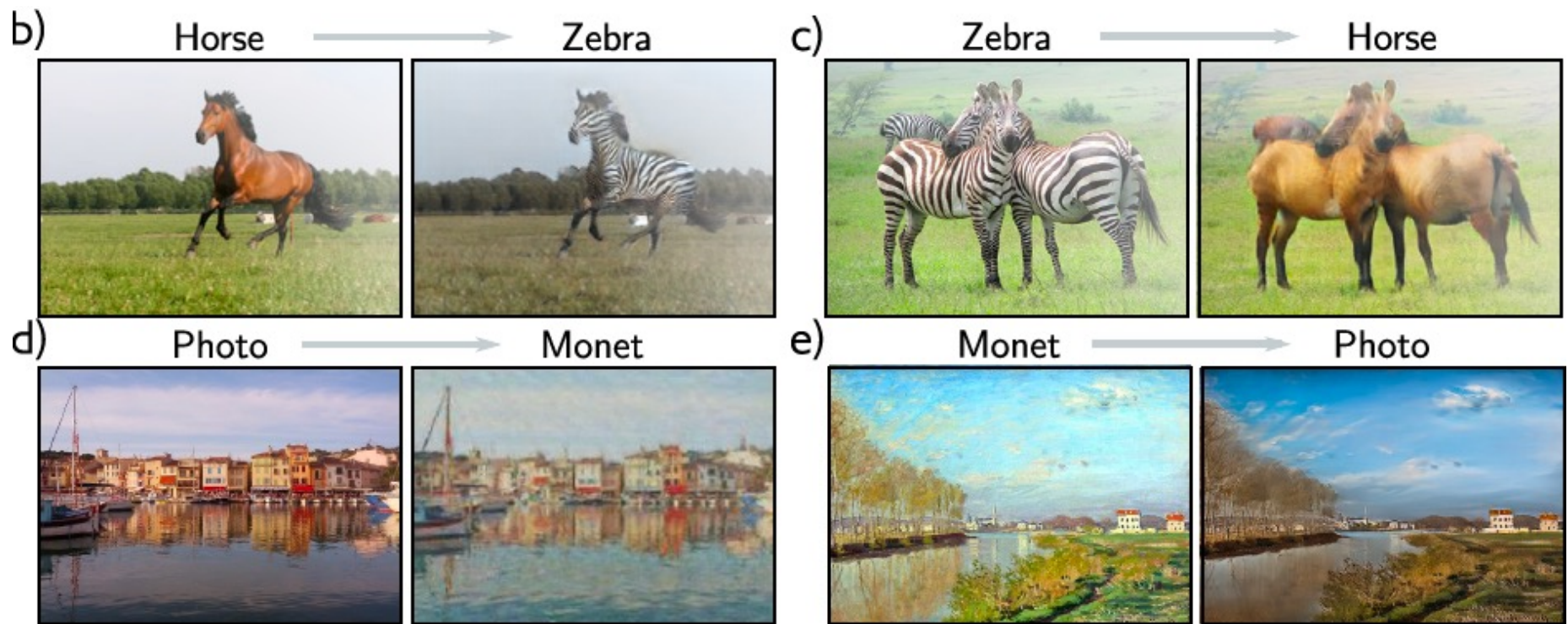
E.g. Horses and zebras, or photos and Monet paintings

Three losses

1. Content loss based on ( $\ell_1$  norm)
2. Discriminator loss (real vs fake)
3. Cycle-consistency loss



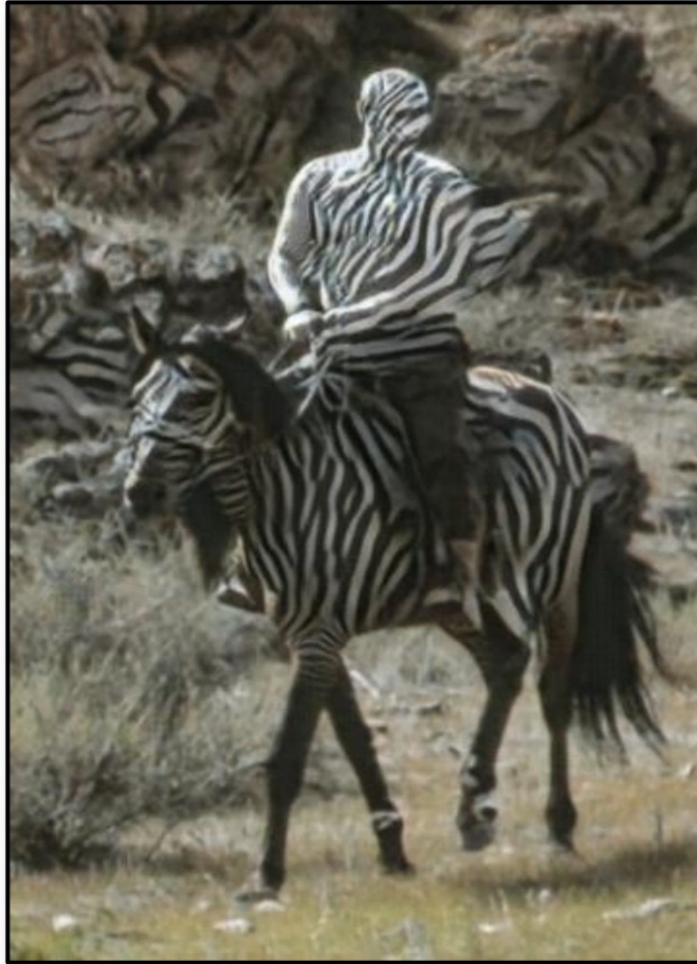
# Image translation: CycleGAN



Input



Output



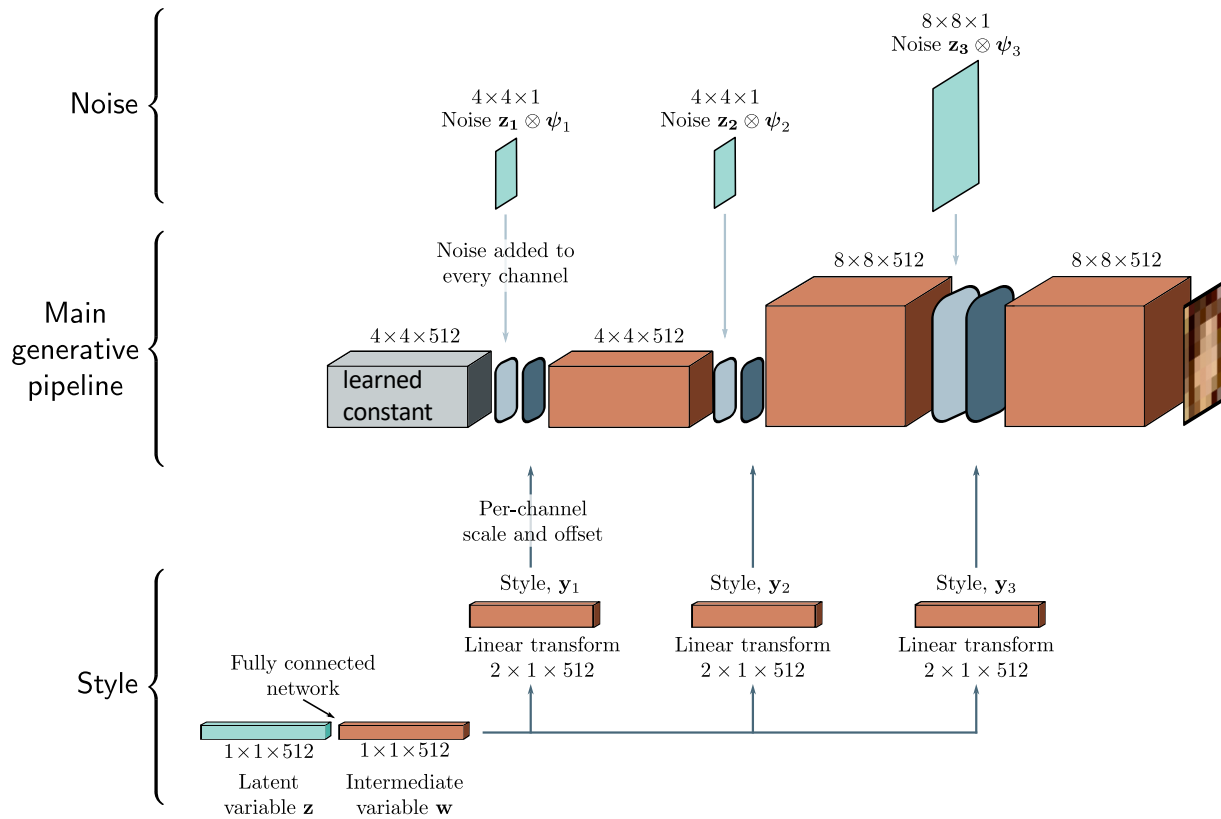
horse → zebra

# GANs

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models
- StyleGAN



# Style GAN



Separates style from noise at different scales

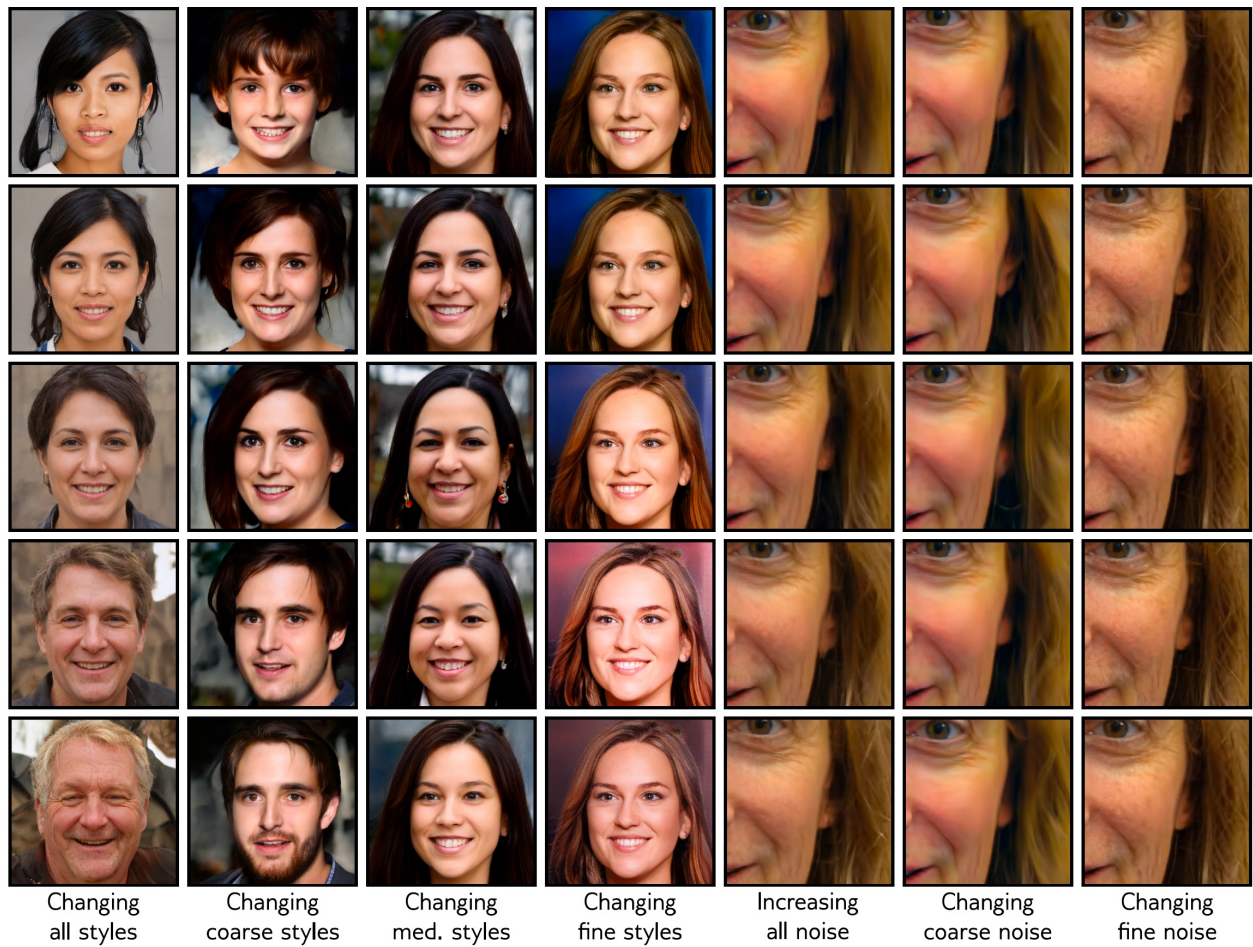
Face Examples

Large style changes: face shape, head pose

Medium-scale changes: facial features

Fine-scale: hair and skin color

Noise: hair placement, freckles

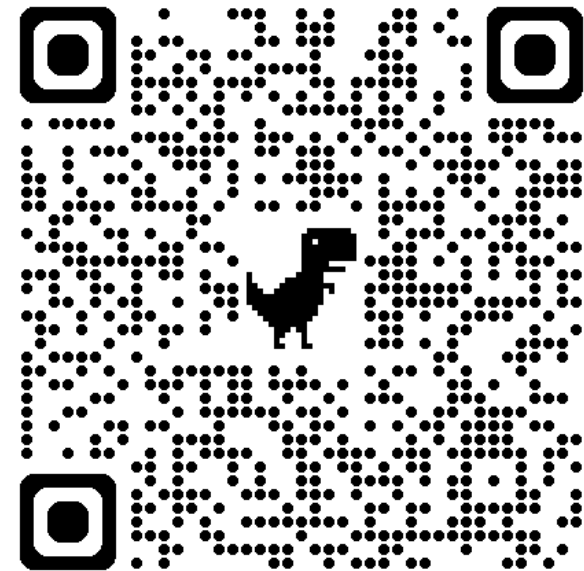


**Figure 15.20** StyleGAN results. First four columns show systematic changes in style at various scales. Fifth column shows the effect of increasing noise magnitude. Last two columns show different noise vectors at two different scales.

## Upcoming Topics

- VAEs
- Diffusion Models
- Graph Neural Networks
- Reinforcement Learning

## Feedback



[Link](#)