

Deep Learning for Data Science

DS 542

<https://dl4ds.github.io/fa2025/>

Adversarial Inputs and Generative Adversarial Networks



Re: Project 3

4 key elements –

1. Design model.
 - Stick with sample code until after benchmark and training.
2. Train model.
 - Stick with sample code until after benchmark.
3. Benchmark model.
 - Do this first!
4. Extract model for prediction script.
 - Submit early enough for feedback!

Plan for Today

- Intriguing properties of neural networks
- Adversarial features
- Generative adversarial networks (GANs)
- GAN loss functions
- GAN examples
- Controlling GANs

Intriguing Properties of Neural Networks

1. The output of a hidden layer is **better thought of as a vector space** than individually important vectors.

- No privileged basis.
- Next layer just takes linear combinations anyway, so **why expect one of the outputs to be special?**

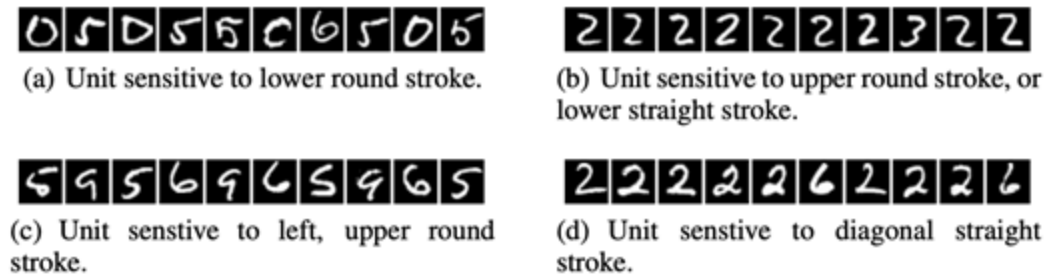


Figure 1: An MNIST experiment. The figure shows images that maximize the activation of various units (maximum stimulation in the natural basis direction). Images within each row share semantic properties.

These are examples maximizing particular outputs.

Intriguing Properties of Neural Networks

1. The output of a hidden layer is **better thought of as a vector space** than individually important vectors.

- No privileged basis.
- Next layer just takes linear combinations anyway, so **why expect one of the outputs to be special?**

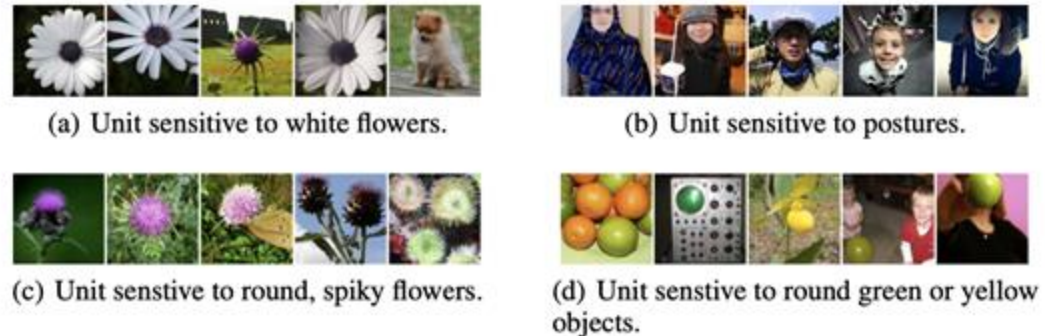


Figure 3: Experiment performed on ImageNet. Images stimulating single unit most (maximum stimulation in natural basis direction). Images within each row share many semantic properties.

These are examples maximizing particular outputs.

Intriguing Properties of Neural Networks

1. The output of a hidden layer is **better thought of as a vector space** than individually important vectors.

- No privileged basis.
- Next layer just takes linear combinations anyway, so **why expect one of the outputs to be special?**

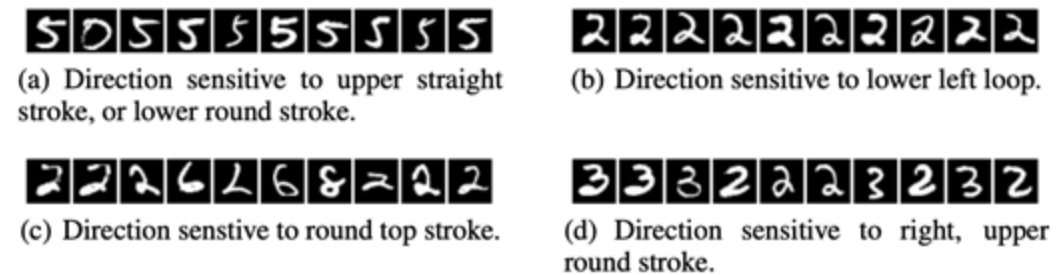


Figure 2: An MNIST experiment. The figure shows images that maximize the activations in a random direction (maximum stimulation in a random basis). Images within each row share semantic properties.

These are examples maximizing random directions.

Intriguing Properties of Neural Networks

1. The output of a hidden layer is **better thought of as a vector space** than individually important vectors.

- No privileged basis.
- Next layer just takes linear combinations anyway, so **why expect one of the outputs to be special?**



(a) Direction sensitive to white, spread flowers.



(b) Direction sensitive to white dogs.



(c) Direction sensitive to spread shapes.



(d) Direction sensitive to dogs with brown heads.

Figure 4: Experiment performed on ImageNet. Images giving rise to maximum activations in a random direction (maximum stimulation in a random basis). Images within each row share many semantic properties.

These are examples maximizing random directions.

Intriguing Properties of Neural Networks

2. It is easy to fool neural networks with small changes to the inputs.

- It was previously “obvious” that you could tweak inputs to change the outputs.
- It was a huge surprise that the tweaks could be so small!

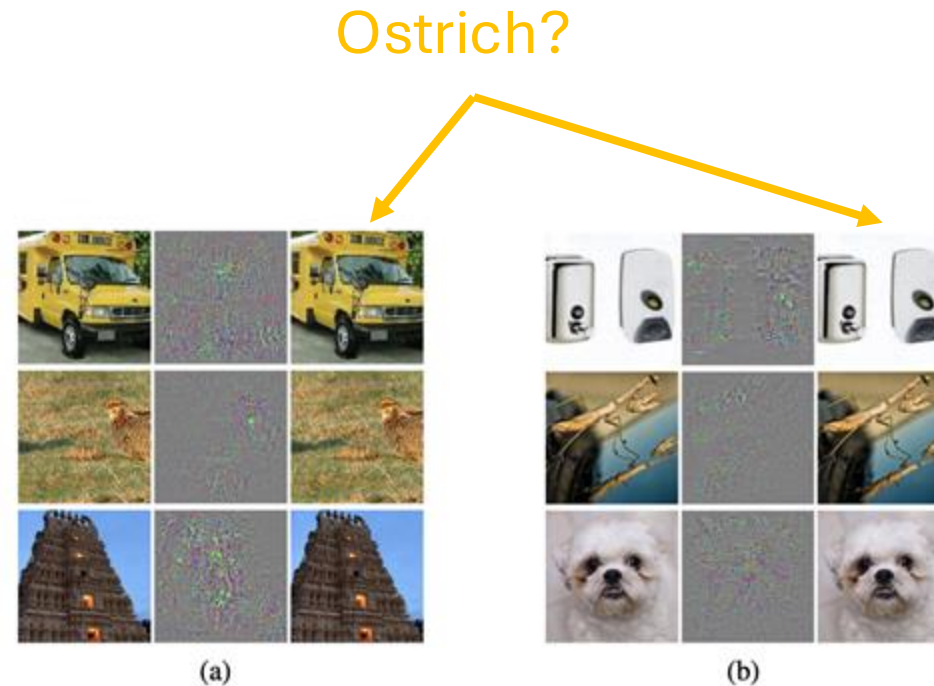


Figure 5: Adversarial examples generated for AlexNet [9].(Left) is a correctly predicted sample, (center) difference between correct image, and image predicted incorrectly magnified by 10x (values shifted by 128 and clamped), (right) adversarial example. All images in the right column are predicted to be an “ostrich, *Struthio camelus*”. Average distortion based on 64 examples is 0.006508. Please refer to <http://goo.gl/huaGPb> for full resolution images. The examples are strictly randomly chosen. There is not any postselection involved.

Intriguing properties of neural networks (2014)

Intriguing Properties of Neural Networks

2. It is easy to fool neural networks with small changes to the inputs.

- It was previously “obvious” that you could tweak inputs to change the outputs.
- It was a huge surprise that the tweaks could be so small!



Figure 6: Adversarial examples for QuocNet [10]. A binary car classifier was trained on top of the last layer features without fine-tuning. The randomly chosen examples on the left are recognized correctly as cars, while the images in the middle are not recognized. The rightmost column is the magnified absolute value of the difference between the two images.

Intriguing properties of neural networks (2014)

Intriguing Properties of Neural Networks

2. It is easy to fool neural networks with small changes to the inputs.

- It was previously “obvious” that you could tweak inputs to change the outputs.
- It was a huge surprise that the tweaks could be so small!

- “In addition, the specific nature of these perturbations is not a random artifact of learning: the same perturbation can cause a different network, that was trained on a different subset of the dataset, to misclassify the same input.”

Intriguing properties of neural networks (2014)

Takeaways

Neural network output space is **not standardized to match our intuitions**.

- “No privileged basis”
- Model ends up targeting vector spaces in the hidden layers?

Neural networks are **surprisingly easy to fool** with small input tweaks.

- These tweaks are usually imperceptible for images.
- They transfer across different data sets and models.
- Possibly related to common vector space representations?

Any Questions?



Moving on

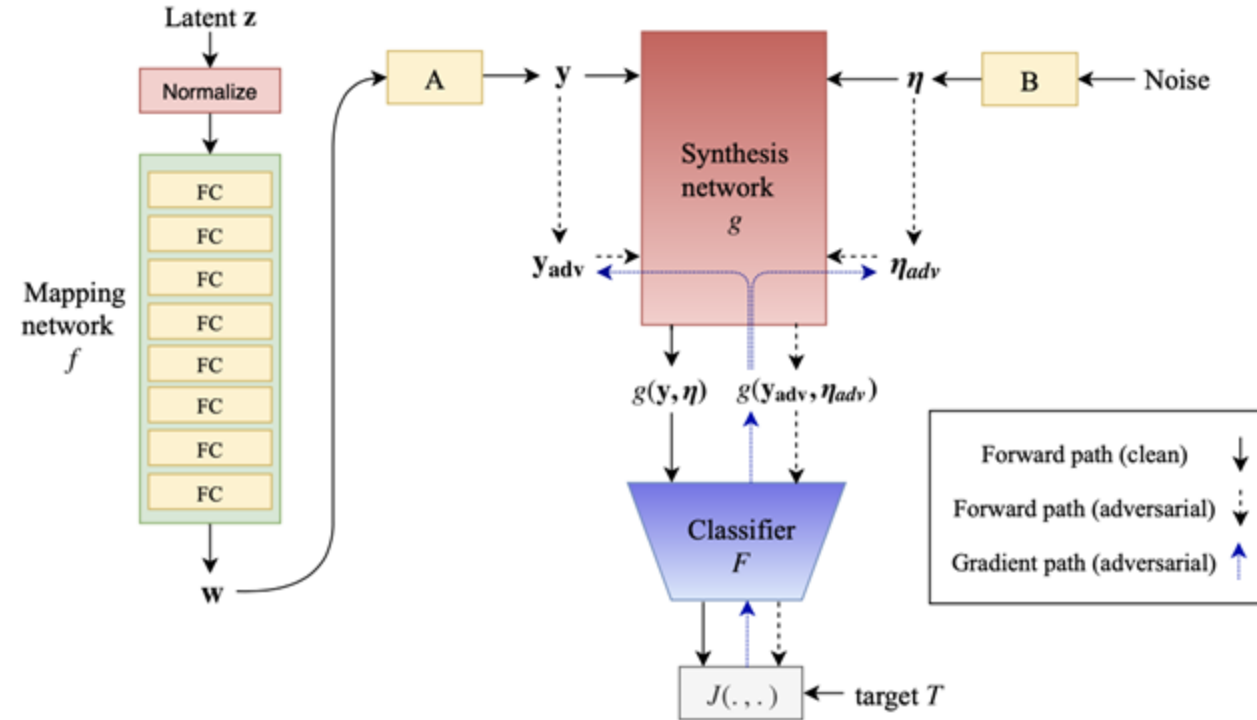
- Intriguing properties of neural networks
- **Adversarial features**
- Generative adversarial networks (GANs)
- GAN loss functions
- GAN examples
- Controlling GANs

Adversarial Training

Idea: use adversarial examples to train more robust classifiers.

Works in practice for the kinds of adversarial examples tested.

- But **not necessarily** for other types.
- Will see soon this **often comes at the cost of accuracy for clean examples**.



What makes a feature useful?

- Features can be useful if they are correlated with a target.
- If the correlation is negative, flip the sign.

ρ -useful features: For a given distribution \mathcal{D} , we call a feature f ρ -useful ($\rho > 0$) if it is correlated with the true label in expectation, that is if

$$\mathbb{E}_{(x,y) \sim \mathcal{D}}[y \cdot f(x)] \geq \rho. \quad (1)$$

← feature utility means minimum correlation

Adversarial Examples Are Not Bugs, They Are Features (2019)

What makes a feature robust?

- A feature is robust if it stays useful under adversarial permutations.

γ -robustly useful features: Suppose we have a ρ -useful feature f ($\rho_{\mathcal{D}}(f) > 0$). We refer to f as a *robust feature* (formally a γ -robustly useful feature for $\gamma > 0$) if, under adversarial perturbation (for some specified set of valid perturbations Δ), f remains γ -useful. Formally, if we have that

robust against specific kinds of changes

$$\mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\inf_{\delta \in \Delta(x)} y \cdot f(x + \delta) \right] \geq \gamma. \quad (2)$$

Adversarial Examples Are Not Bugs, They Are Features (2019)

What about useful but non-robust features?

- Training will learn useful features even if they are not robust.

Useful, non-robust features: A *useful, non-robust feature* is a feature which is ρ -useful for some ρ bounded away from zero, but is not a γ -robust feature for any $\gamma \geq 0$. These features help with classification in the standard setting, but may hinder accuracy in the adversarial setting, as the correlation with the label can be flipped.

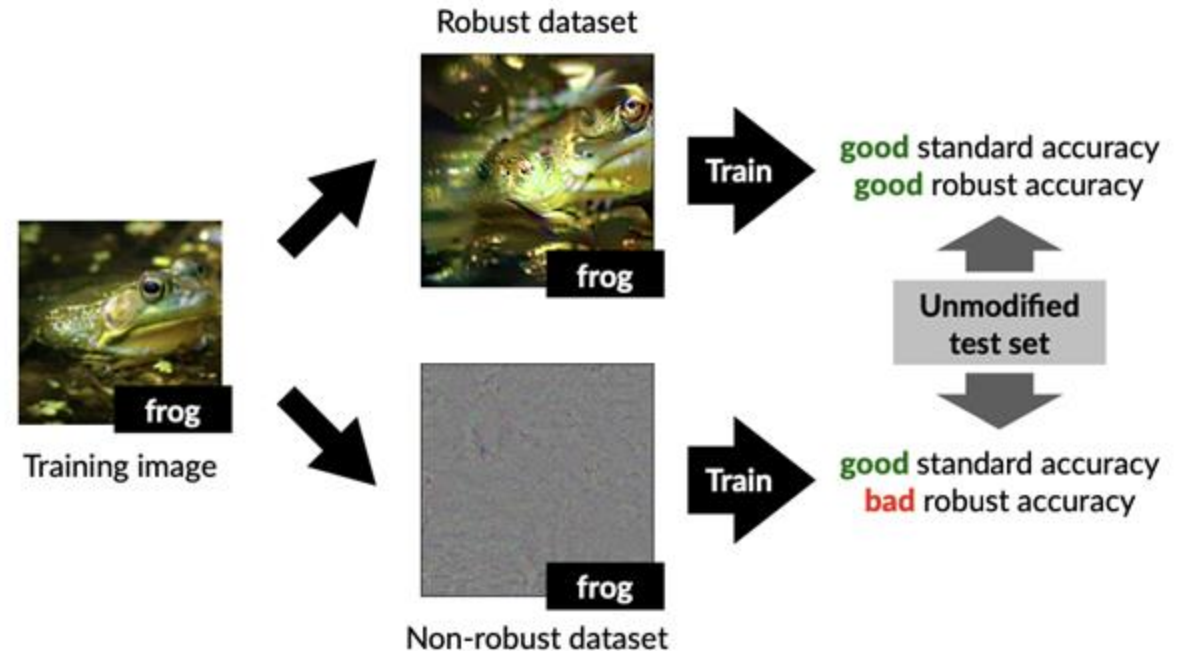
have minimum correlation,
but not after adversarial perturbation

Adversarial Examples Are Not Bugs, They Are Features (2019)

Create 1000 features that ~~are~~ 1% correlated w/target.
Averaging them gives a better predictor.

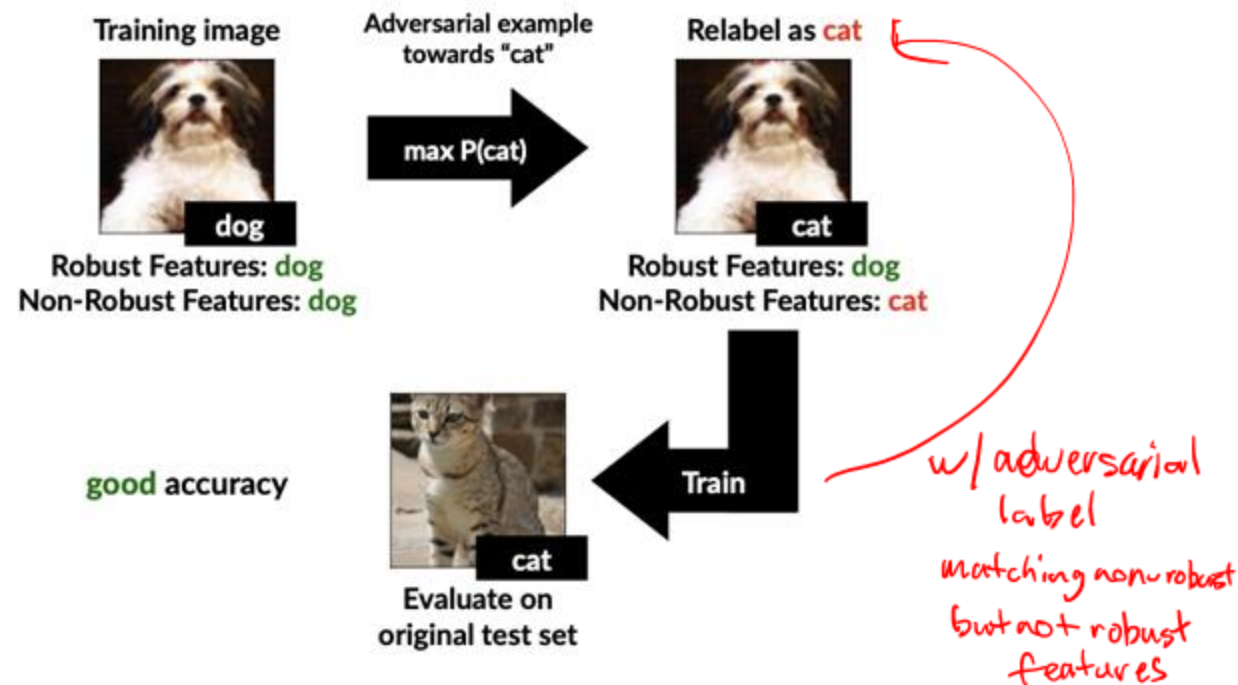
Training Data can be Tweaked to be More or Less Robust

Adversarial Examples Are Not Bugs, They Are Features (2019)



Silly Training Tricks

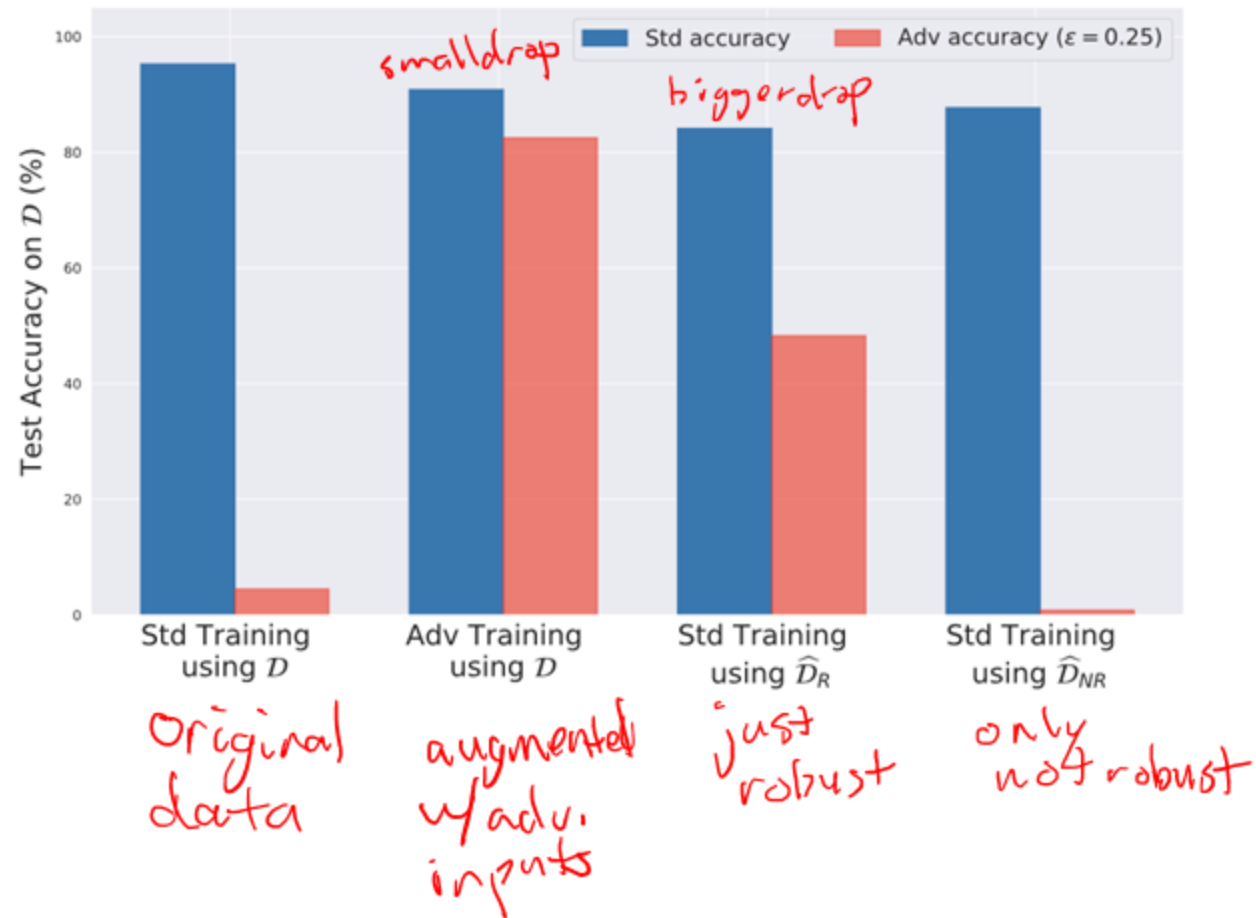
Adversarial Examples Are Not Bugs, They Are Features (2019)



Takeaways

- Adversarial inputs take advantage of real features in the training data.
- Can try to remove these “non-robust” features, but may cost performance.

Adversarial Examples Are Not Bugs, They Are Features
(2019)



Any Questions?



Moving on

- Intriguing properties of neural networks
- Adversarial features
- Generative adversarial networks (GANs)
- GAN loss functions
- GAN examples
- Controlling GANs

Generative Adversarial Networks

TLDR: use adversarial examples idea to build a really good image generator.

Appears to be an evolutionary dead end, but still interesting because

- Clever training procedure.
- First really good image generation technique.
- Helped identify a lot of possible pitfalls and sometimes techniques to fix them.

This Person Does Not Exist

<https://thispersondoesnotexist.com>



Imagine...

A function f that maps samples from normally distributed noise to images.

image generator.

like a seed, but really latent code

Can we train a function g that distinguishes the output of f from a real image?

Imagine...

A function f that maps samples from normally distributed noise to images.

Can we train a function g that distinguishes the output of f from a real image?

- If f is randomly initialized but untrained, then g should be easy to train.

Imagine...

A function f that maps samples from normally distributed noise to images.

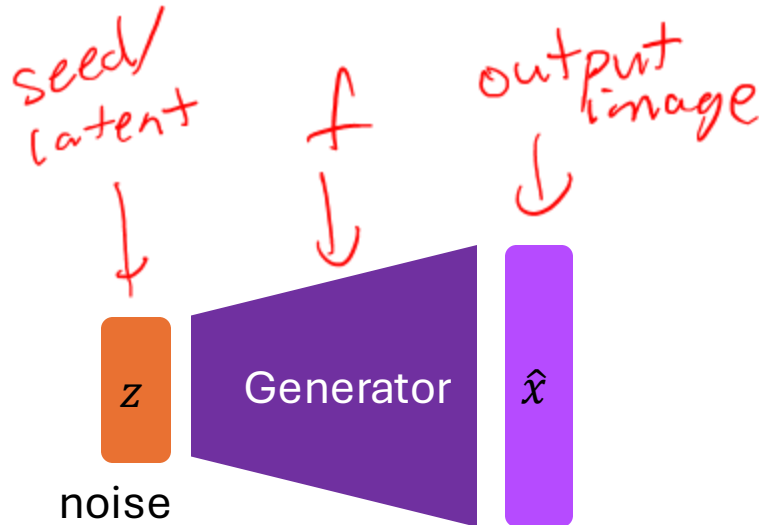
Can we train a function g that distinguishes the output of f from a real image?

- If f is randomly initialized but untrained, then g should be easy to train.
- What if we then train f to fool g ?

training objective

$g(f(\mathbf{z})) \Rightarrow \text{True}$
↑
looks legit!
maximize g 's prob(real)

General Idea of GANs

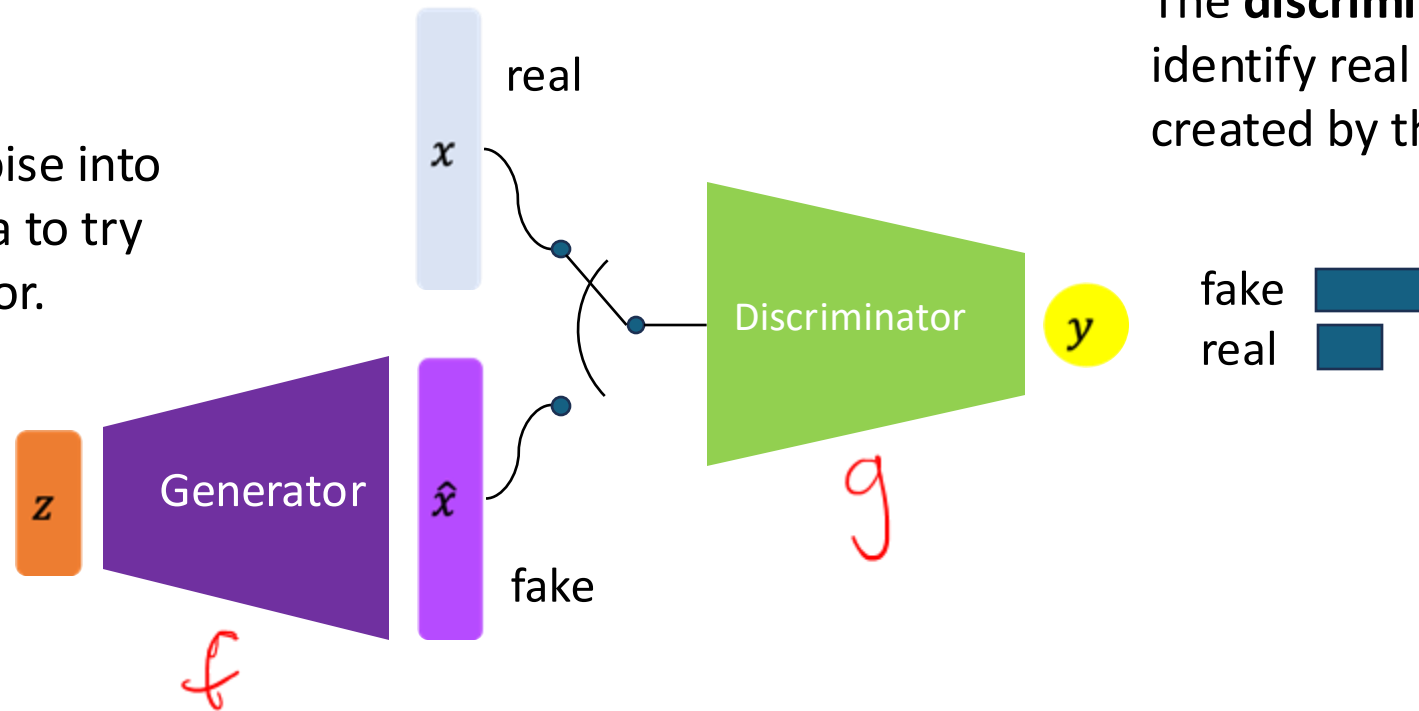


- Don't try to build a probability model directly *use arbitrary z*
- Learn a transformation from a sample of noise to look similar to training data distribution

Generative Adversarial Networks

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

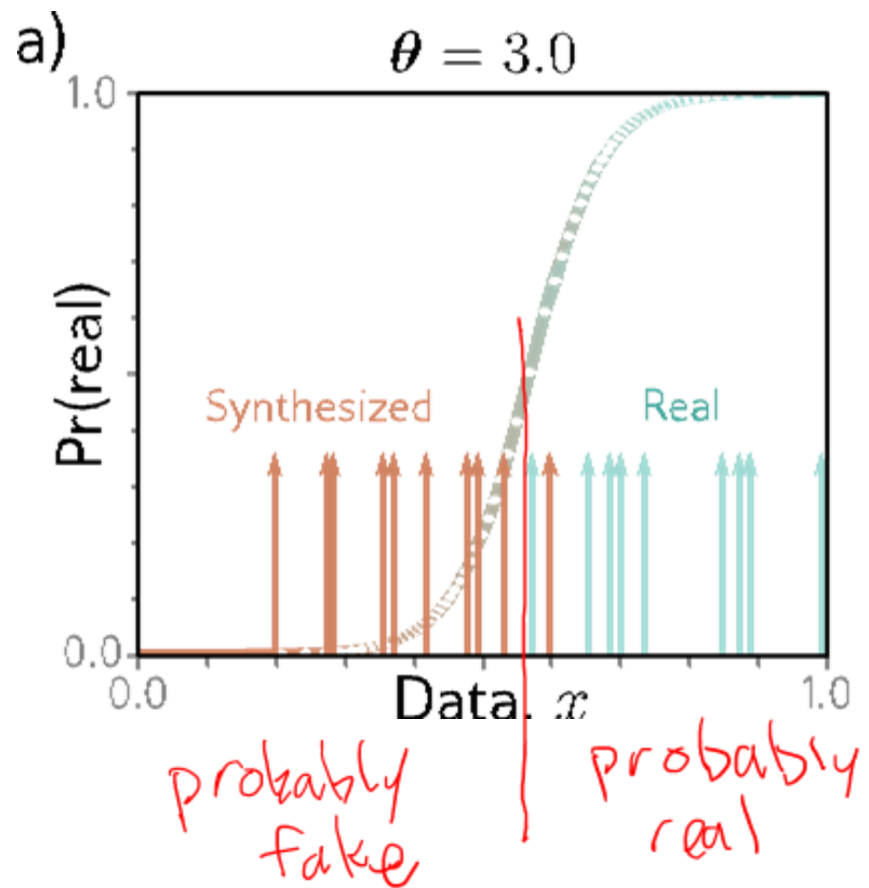
The **generator** turns noise into an imitation of the data to try to trick the discriminator.



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

GAN example

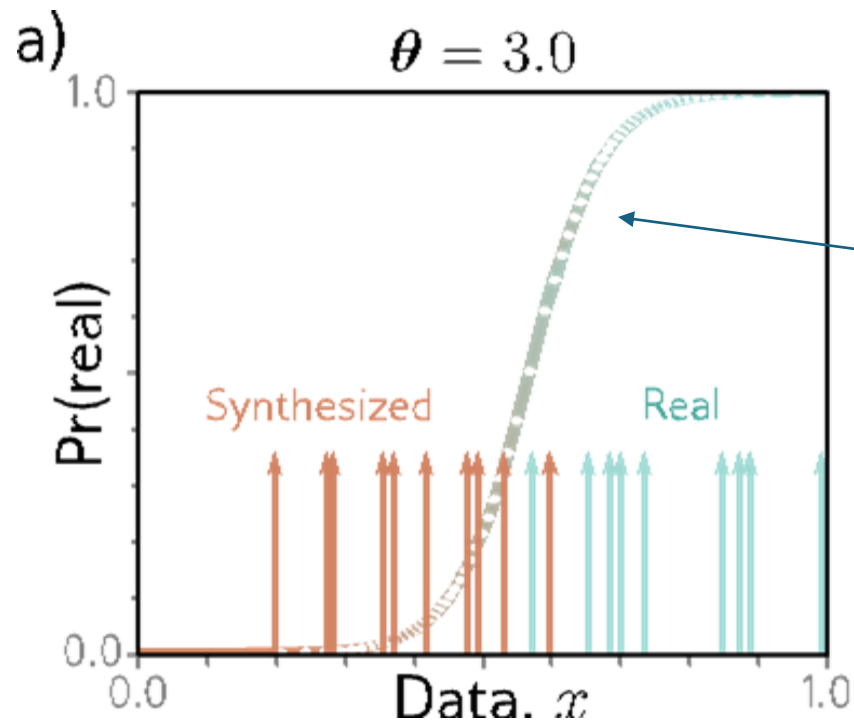
$$x_j^* = g[z_j, \theta] = z_j + \theta$$



- 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 θ .
- Train a classifier on the data

GAN example

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

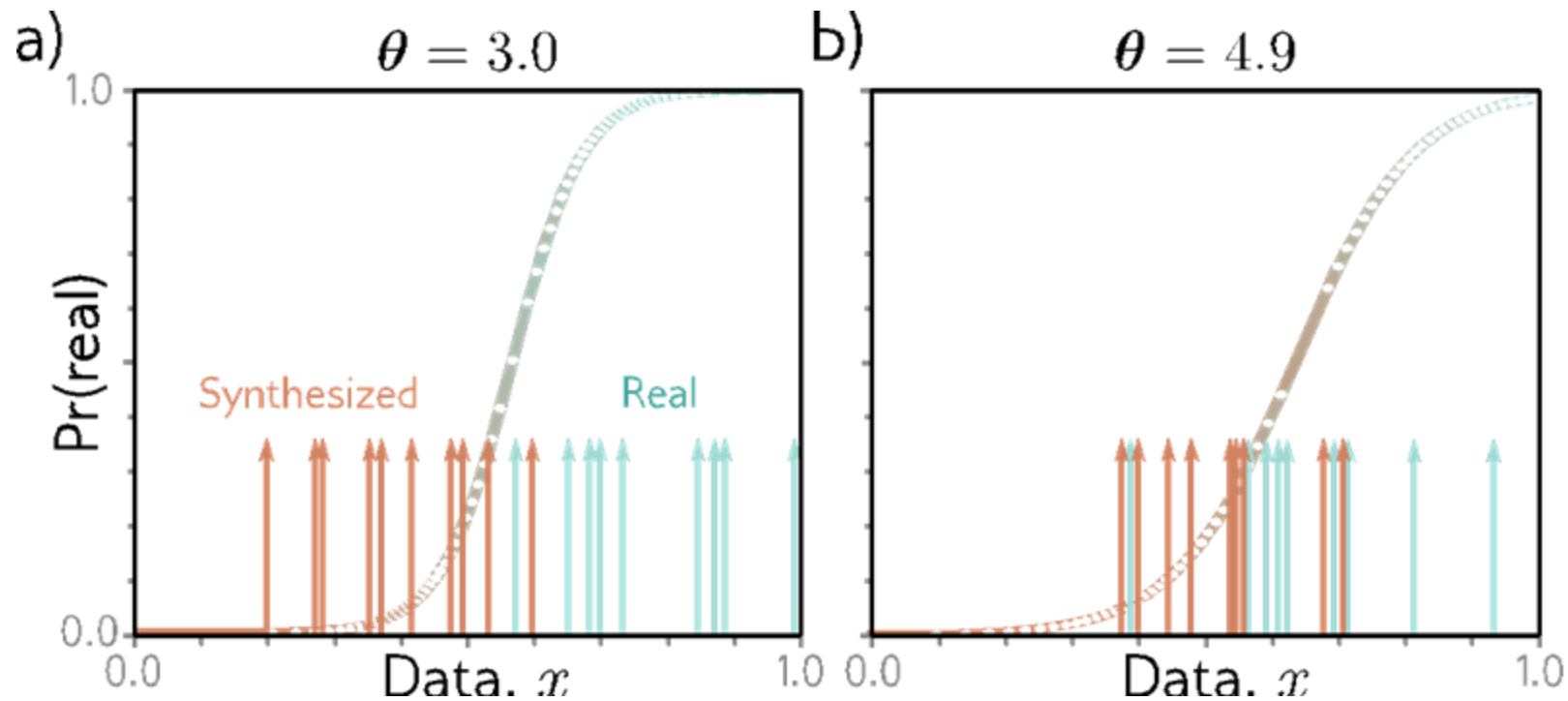


$f[\bullet, \phi]$
Discriminator

- Train the **discriminator**
- using logistic regression parameterized by ϕ
- 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

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

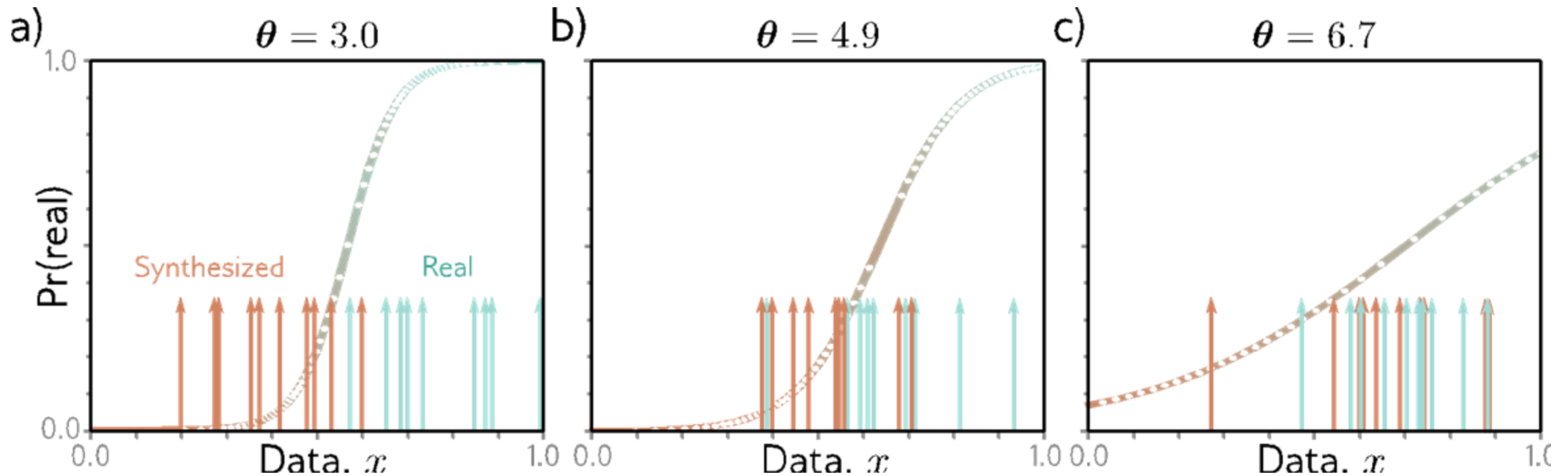


- Train the **generator** to update θ in order to *increase* the loss on the discriminator
- Then train the **discriminator** to *decrease* the loss

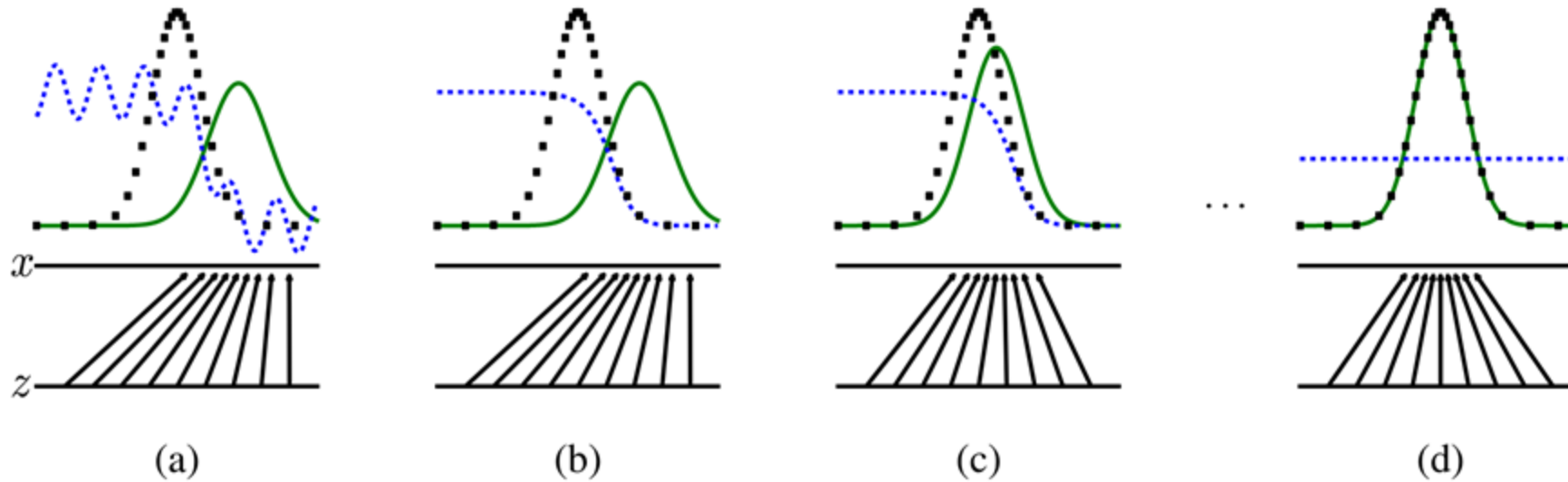
GAN example

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

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



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

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

Any Questions?



Moving on

- Intriguing properties of neural networks
- Adversarial features
- Generative adversarial networks (GANs)
- **GAN loss functions**
- GAN examples
- Controlling GANs

GAN cost function

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

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

real
or
fake

GAN cost function

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

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

$y_i = 1$ / real case

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

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

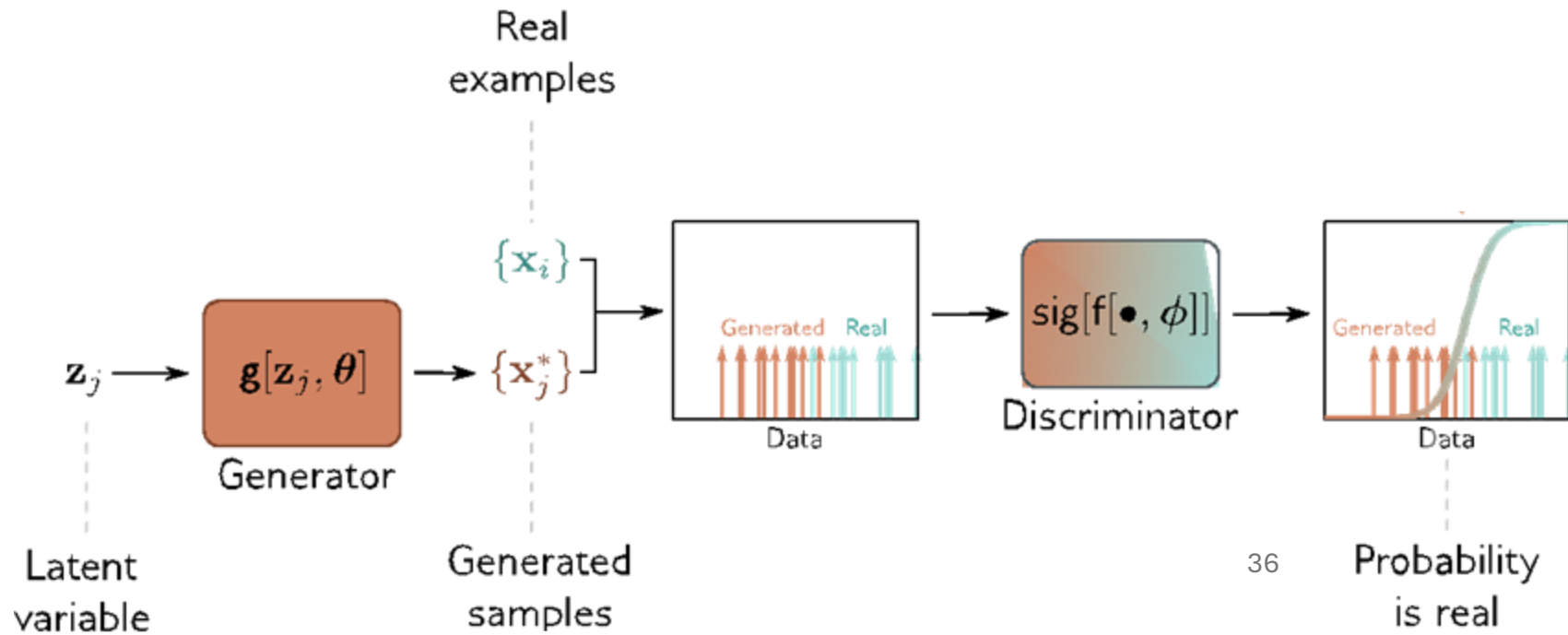
35

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

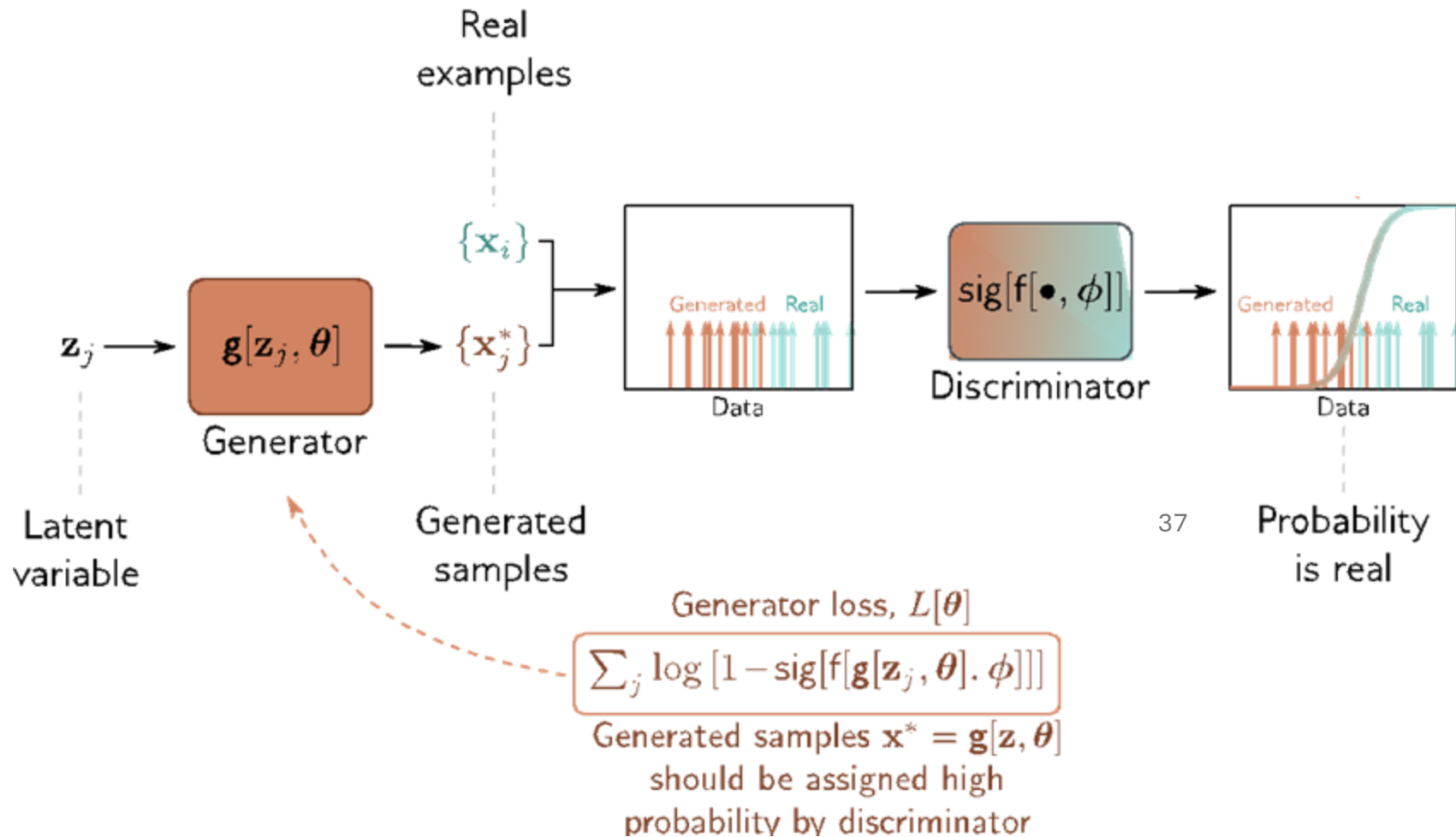
*These are **real** samples so $y_i = 1$*

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

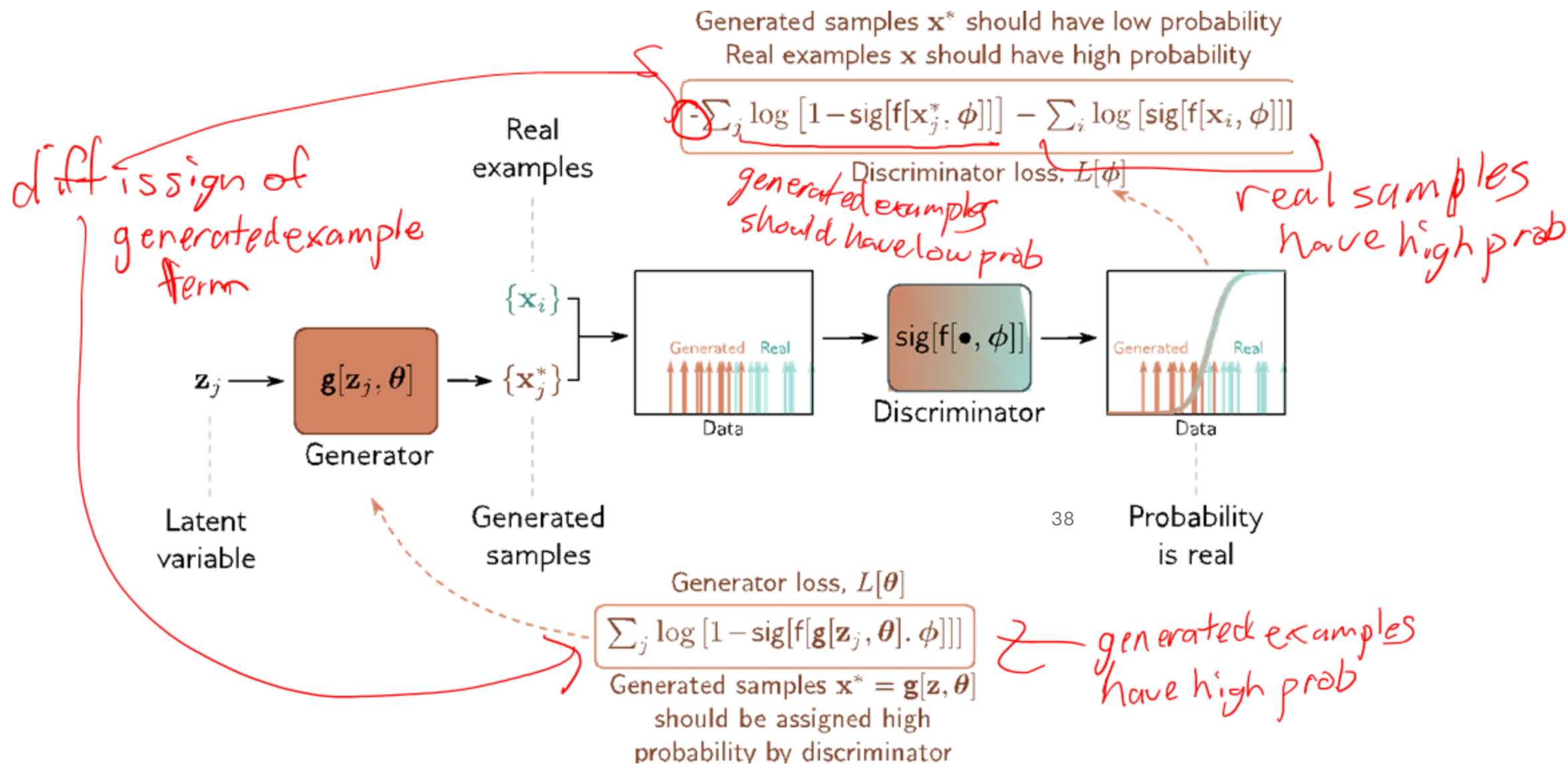
GAN loss function



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, $y = 1$:

Previous

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

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

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

generator params

disc params

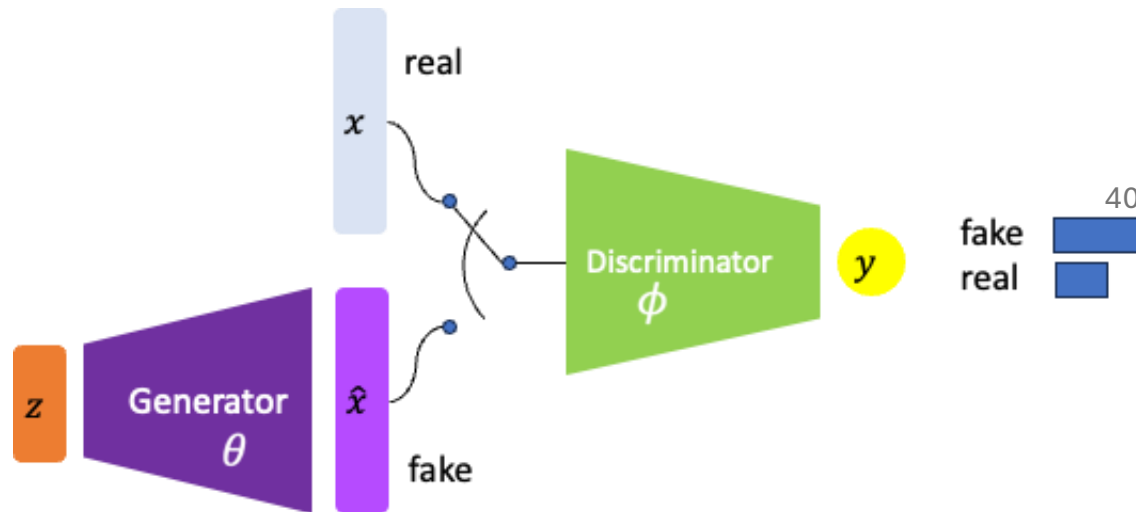
constant for θ opt.

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, ϕ , are manipulated to *minimize* the loss function

The **generator** parameters, θ , are manipulated to *maximize* the loss function.



GAN Cost function

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

The **discriminator** parameters, ϕ , are manipulated to *minimize* the loss function

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

Generator
does not
need
access to
real
examples.

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

GAN Solution

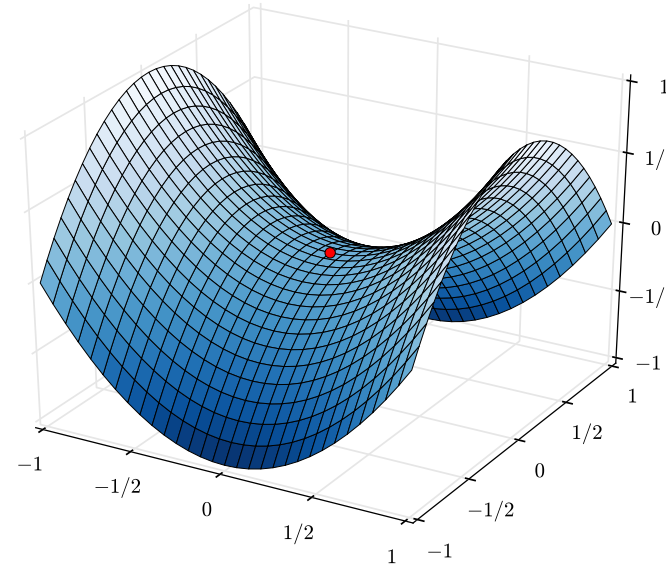
$$\hat{\phi}, \hat{\theta} = \operatorname{argmax}_{\theta} \left[\operatorname{argmin}_{\phi} \left[\sum_j -\log \left[1 - \operatorname{sig}[f[\mathbf{g}[\mathbf{z}_j, \theta], \phi]] \right] - \sum_i \log \left[\operatorname{sig}[f[\mathbf{x}_i, \phi]] \right] \right] \right]$$

- The 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](#)



must optimize
both together



Convergence Proof Theory

4.2 Convergence of Algorithm 1

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G , and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\mathbf{x} \sim p_{data}} [\log D_G^*(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g} [\log(1 - D_G^*(\mathbf{x}))]$$

then p_g converges to p_{data} real distribution

generated distribution

Generative Adversarial Nets (2014)

Any Questions?

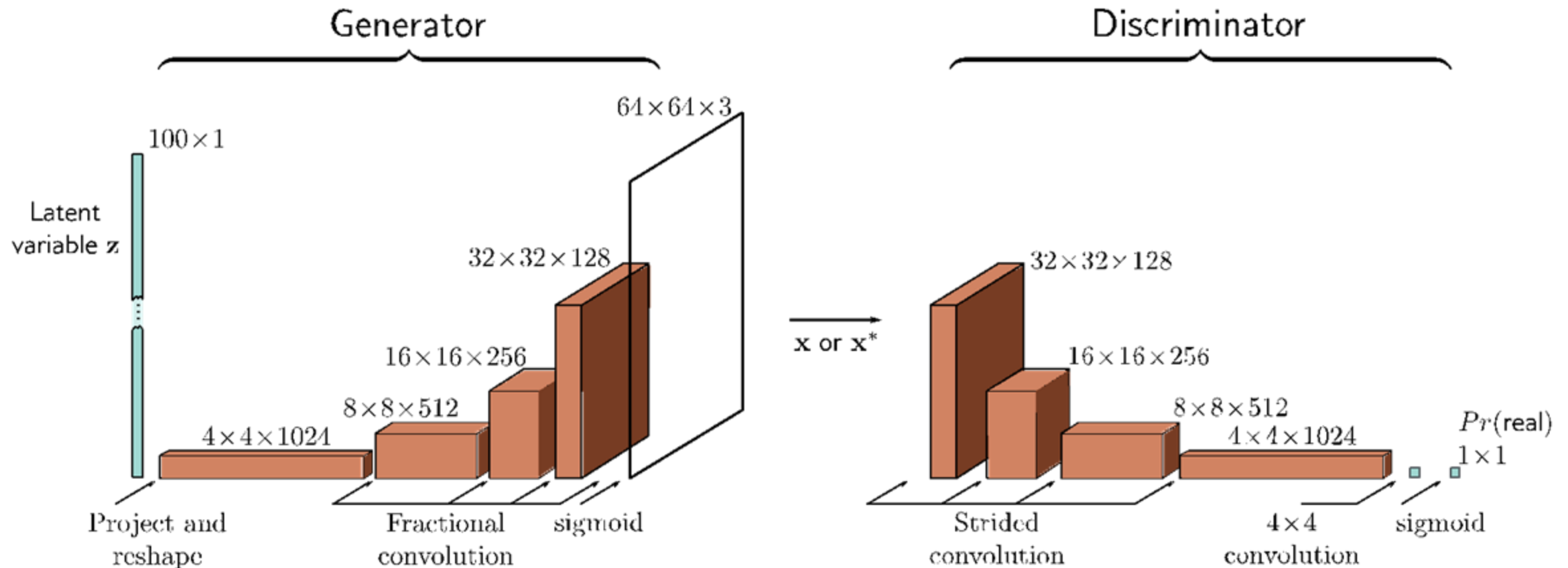


Moving on

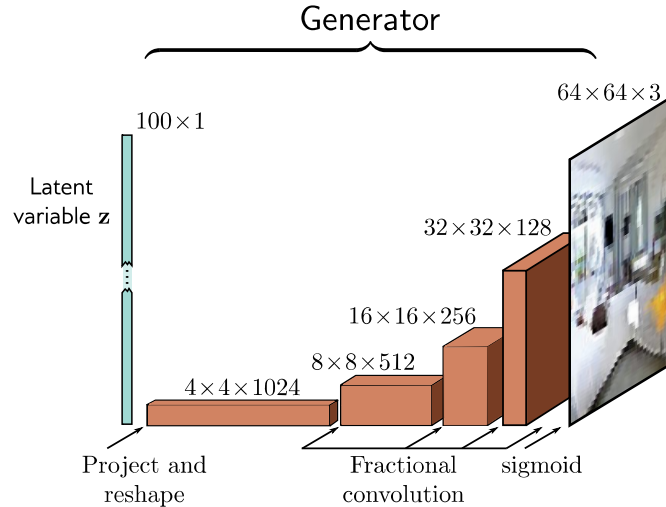
- Intriguing properties of neural networks
- Adversarial features
- Generative adversarial networks (GANs)
- GAN loss functions
- **GAN examples**
- Controlling GANs

Deep Convolutional (DC) GAN

- Early GAN specialized in image generation



Deep Convolutional (DC) GAN



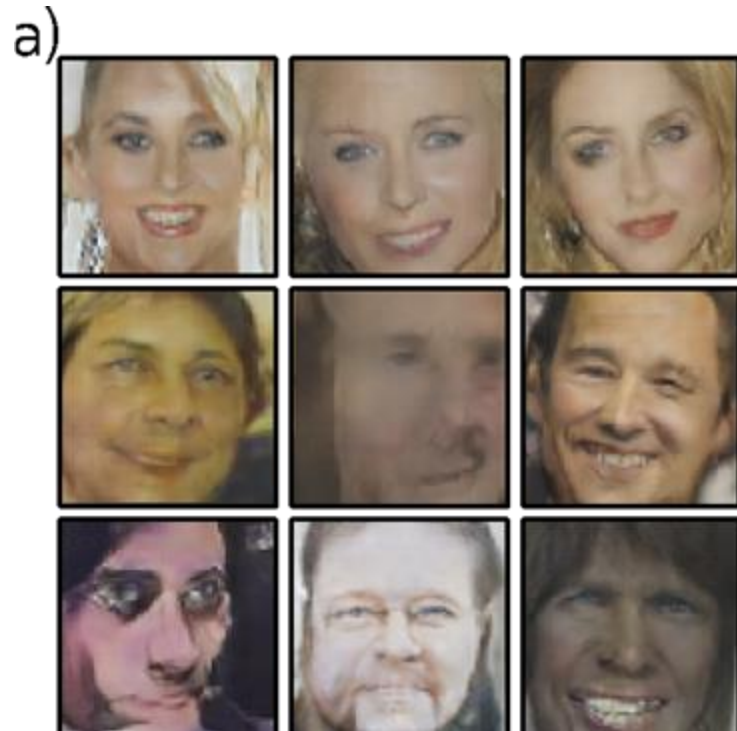
When training is complete

Discard discriminator

Draw new latent variable

Pass through generator

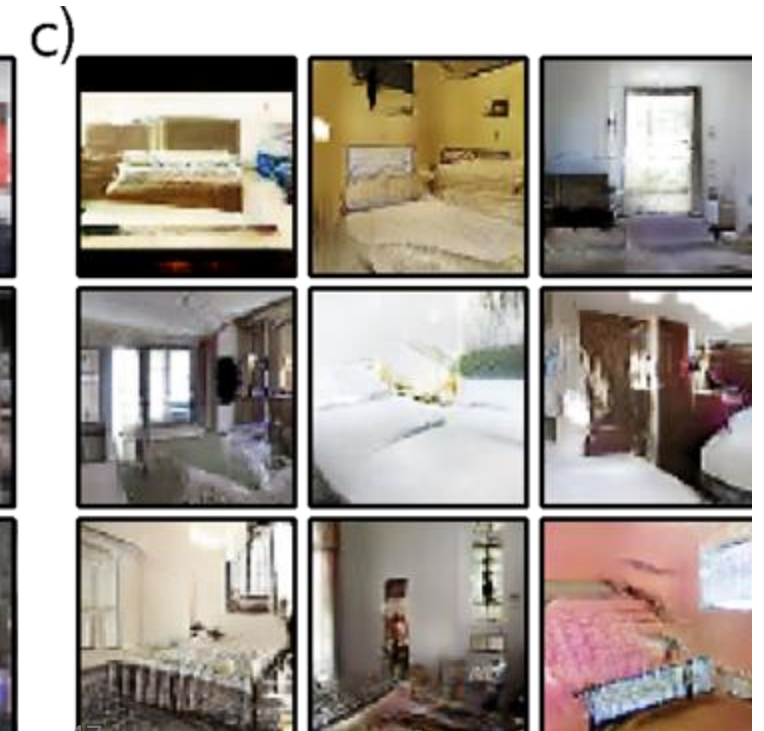
DC GAN Results



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.

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

$$D_{KL}(P||Q) = \sum_x P(x) \log \frac{Q(x)}{P(x)}$$

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

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:

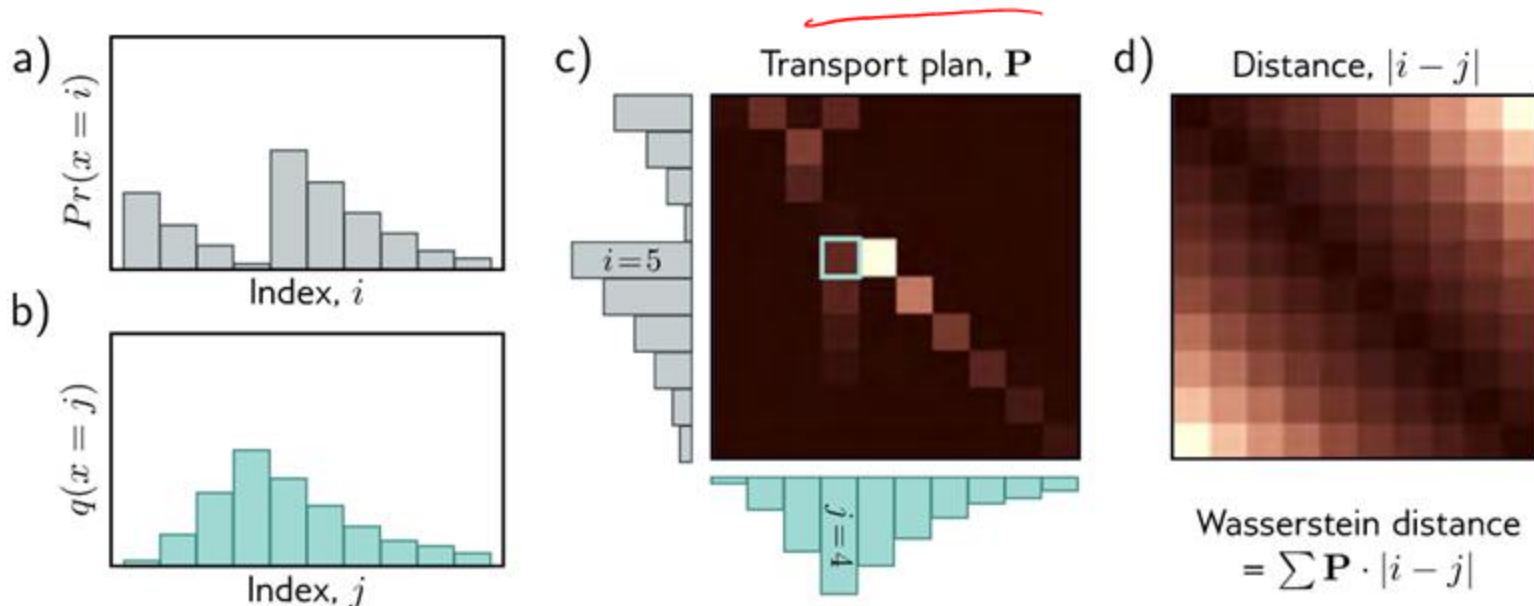
(i) the GAN loss can be interpreted in terms of distances between probability distributions and that

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

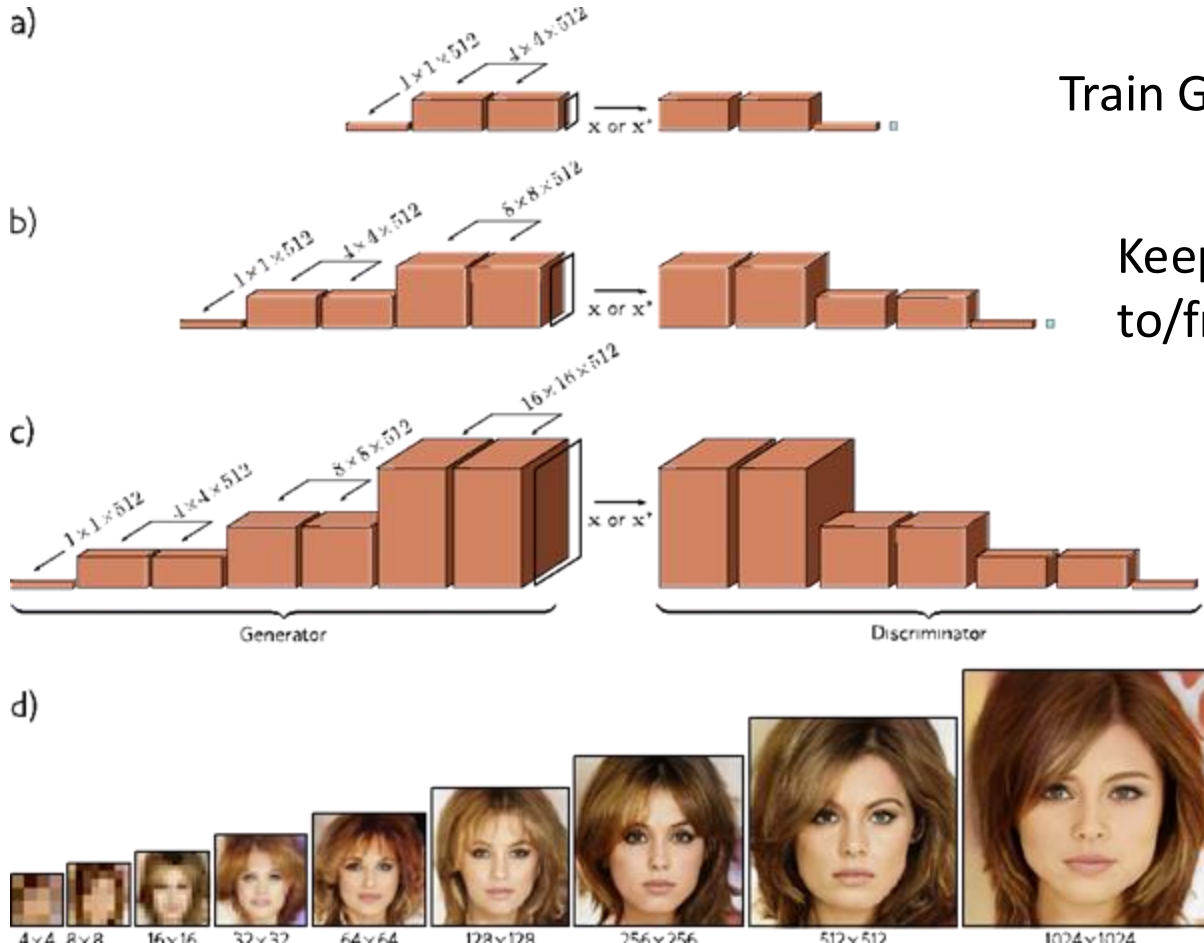
We need a distance metric with better properties. *→ more tractable prob. distance*

Wasserstein 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 $\sum \mathbf{P} \cdot |i - j|$



Trick 1: Progressive growing



Train GAN to generate and discriminate 4x4 images

Keep weights from step (a), add layers to get to/from 8x8 images and continue training GAN

Add layers to get to 16x16 and continue to train.

Repeat above steps to get to high resolution.

Trick 2: Minibatch discrimination

(Remember
Batch Norm?)

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

↓
things to fix

Trick 3: Truncation



- Only choose random values of latent variables that are less than a threshold τ 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 .



StyleGAN (2019-2021)

GAN development focused on quality.

- Integrated styles into generation process.
- Analysis using more visually oriented evaluation (Fréchet inception distance)
- Later models systematically worked on known artifacts in the generation process.

[A Style-Based Generator Architecture for Generative Adversarial Networks \(2019\)](#)

[Analyzing and Improving the Image Quality of StyleGAN \(2019\)](#)

[Alias-Free Generative Adversarial Networks \(2021\)](#)



StyleGAN

Considered a very specific notion of style...

- Coarse vs medium vs fine styles
- Able to separate and remix them separately

A Style-Based Generator
Architecture for Generative
Adversarial Networks (2019)

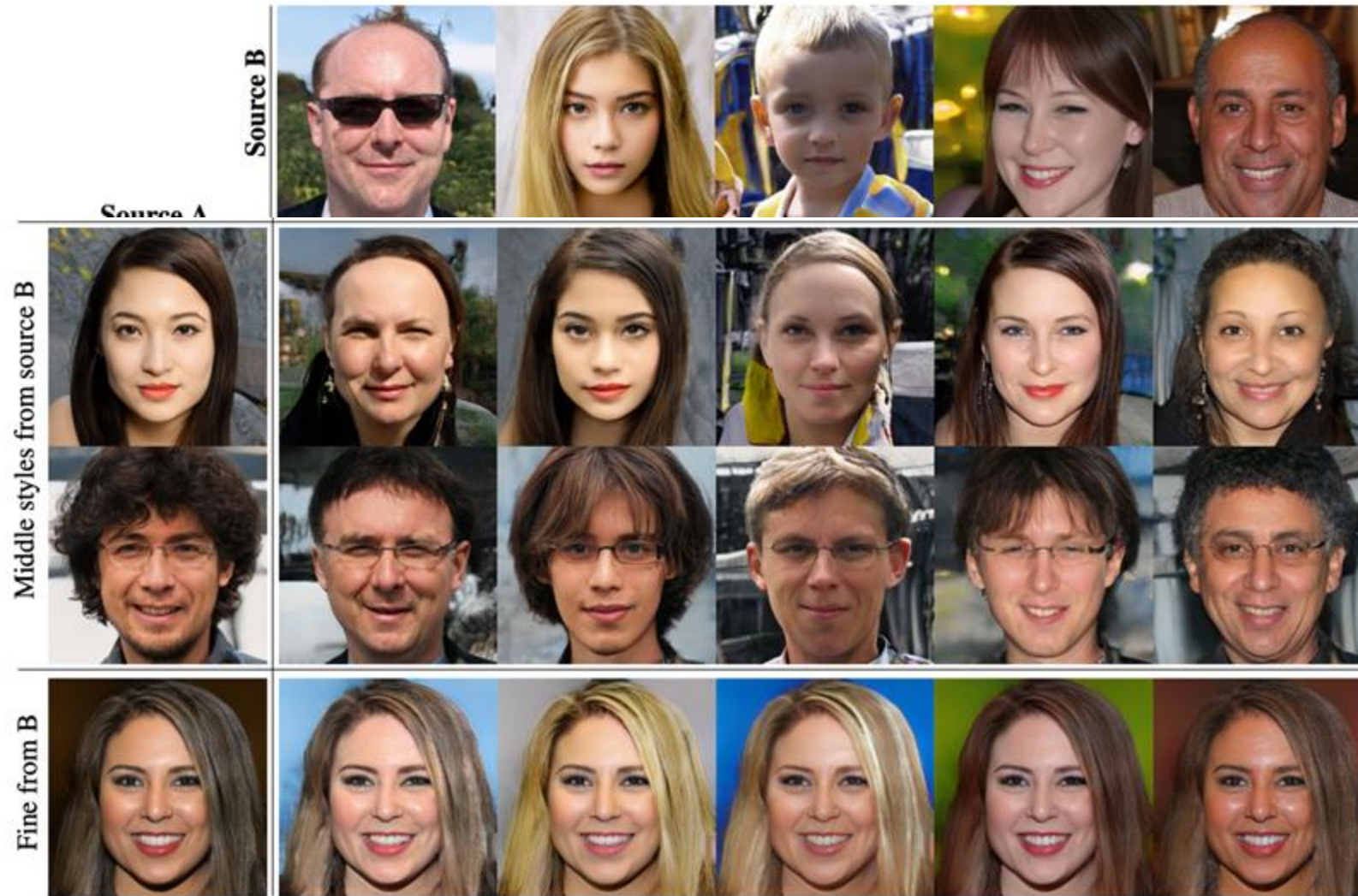


StyleGAN

Considered a very specific notion of style...

- Coarse vs medium vs fine styles
- Able to separate and remix them separately

[A Style-Based Generator Architecture for Generative Adversarial Networks \(2019\)](#)



This Person Does Not
Exist

Images made with
StyleGAN (still v1?)

<https://thispersondoesnotexist.com/>



Any Questions?



Moving on

- Intriguing properties of neural networks
- Adversarial features
- Generative adversarial networks (GANs)
- GAN loss functions
- GAN examples
- Controlling GANs

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

Auxiliary Classifier GAN results

Trained on ImageNet images and classes.



monarch
butterfly

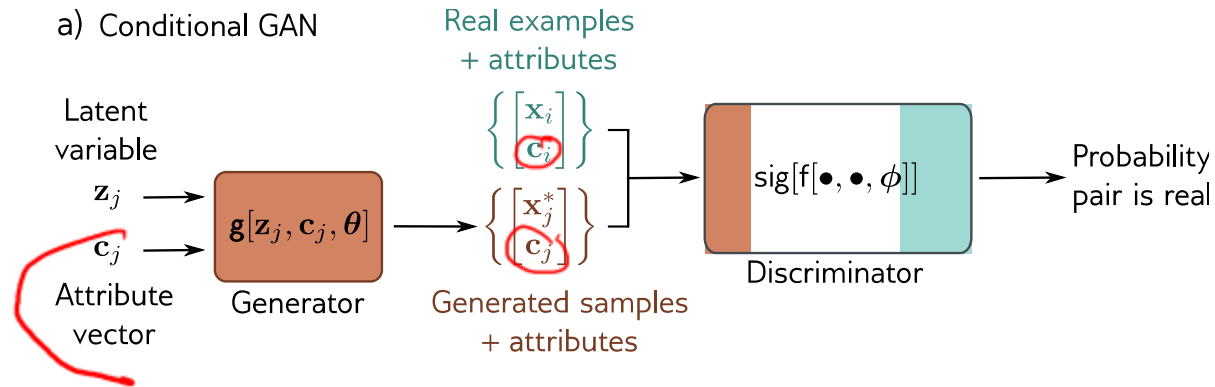
goldfinch

daisies

redshanks

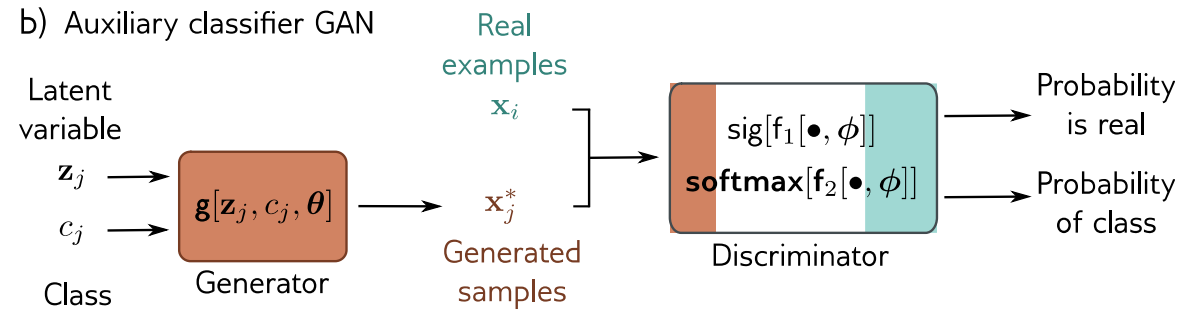
gray whales

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

Image translation: Pix2Pix

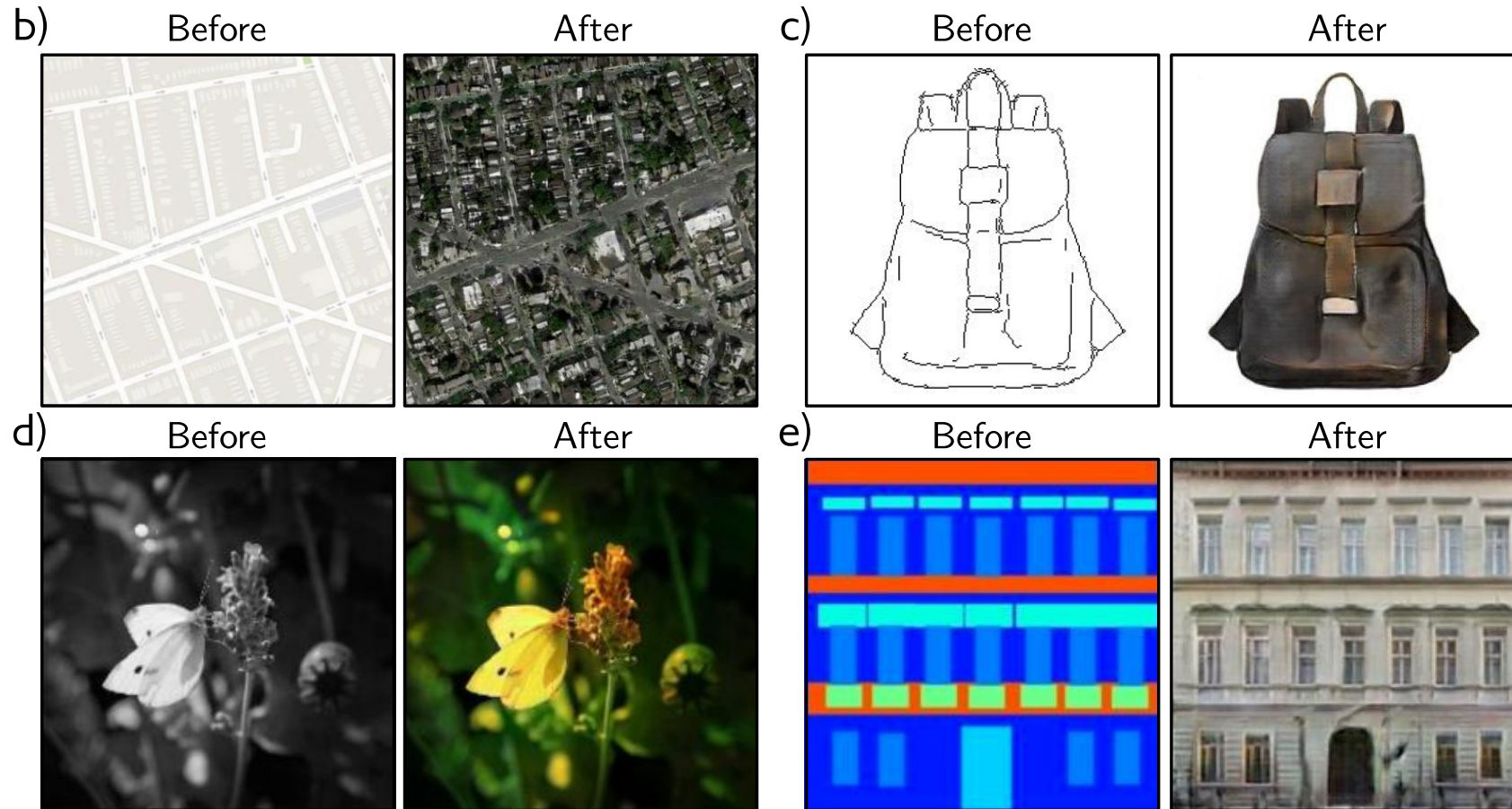
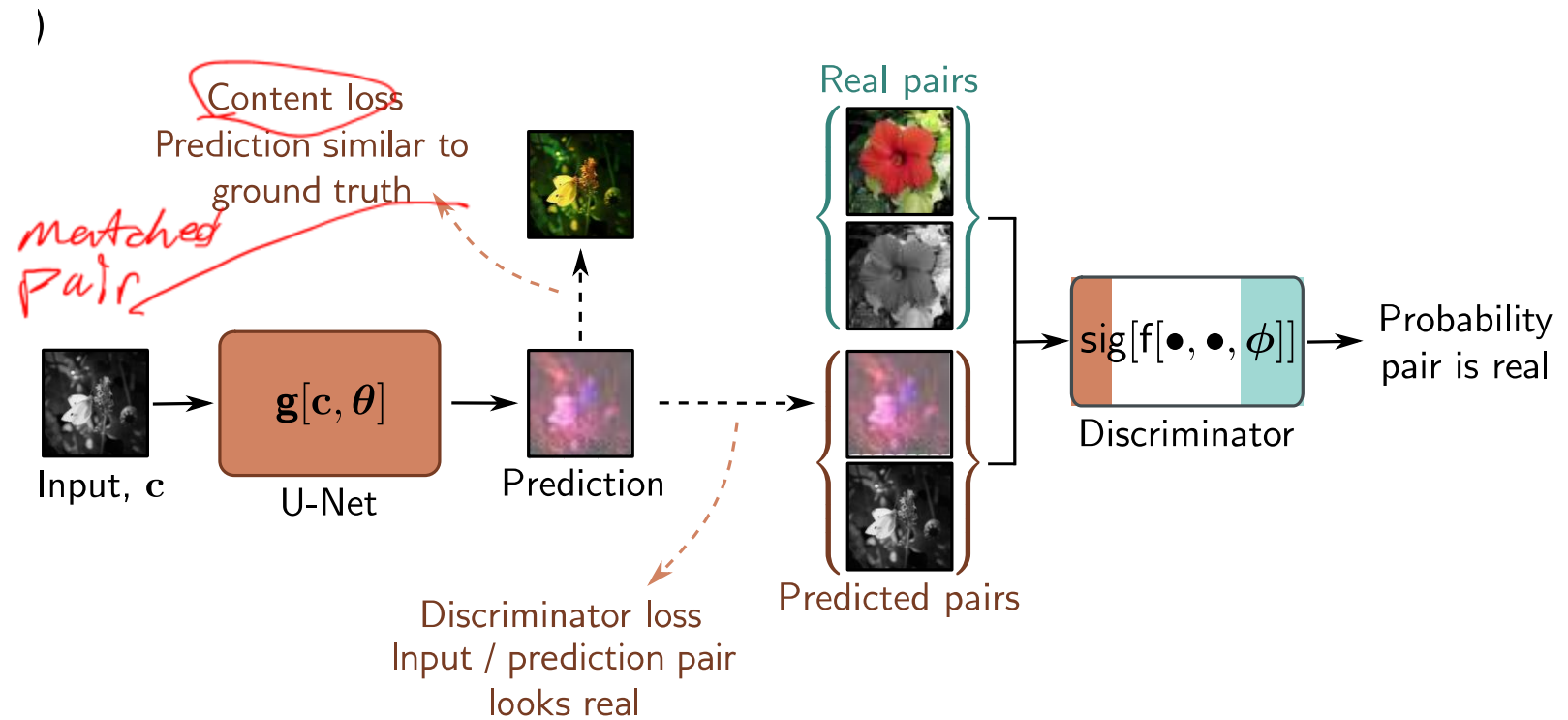


Image translation: Pix2Pix



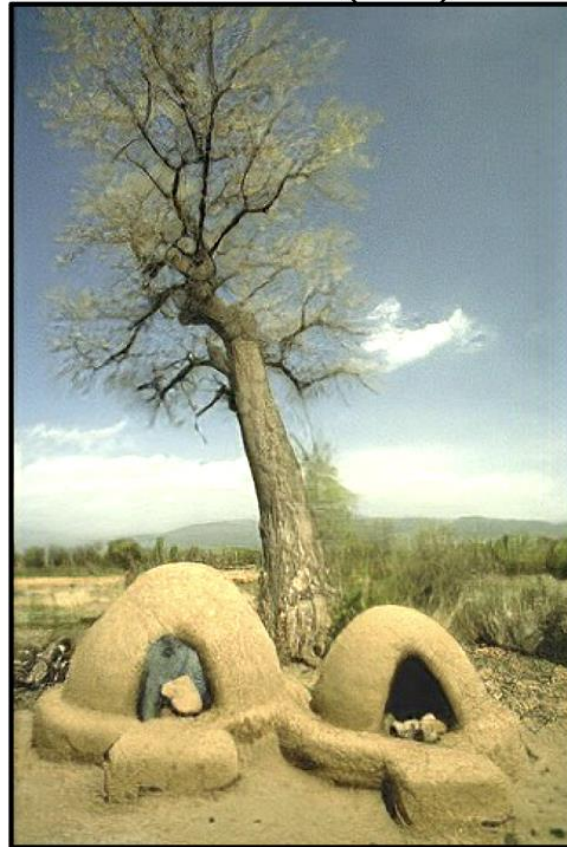
- Maps one image to a different style image using a U-Net type model
- Adds a content loss (ℓ_1 norm) to make the input similar to ground truth
- Discriminator fed input/prediction and real/modified pairs to predict real or fake

Image translation: SRGAN

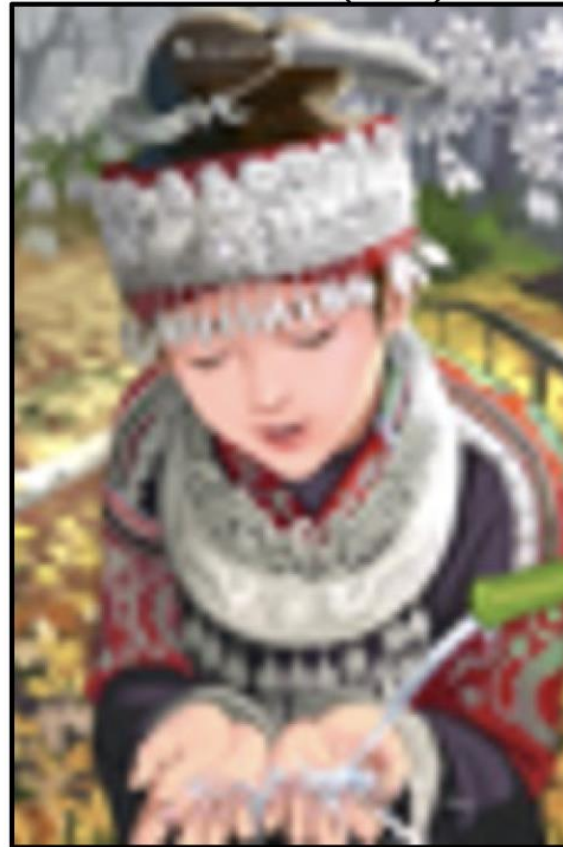
b) Bicubic (4×)



c) SRGAN (4×)



d) Bicubic (4×)



e) SRGAN (4×)



Image translation: SRGAN

a)

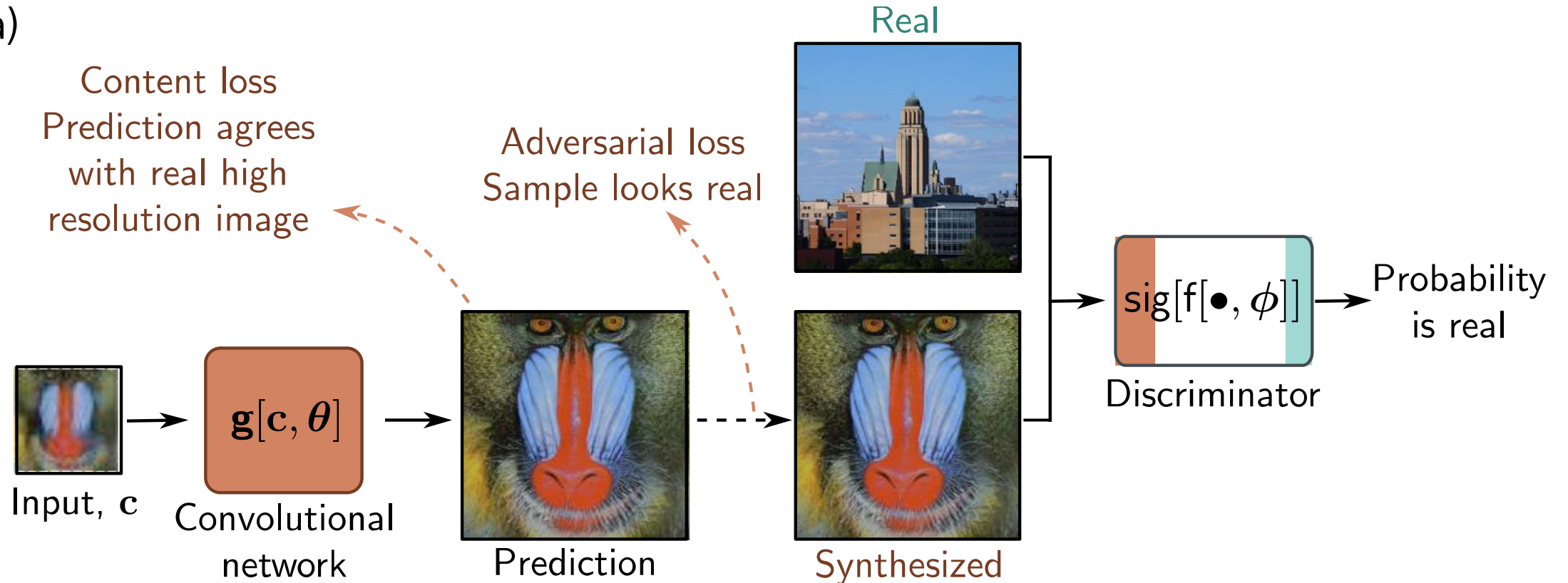
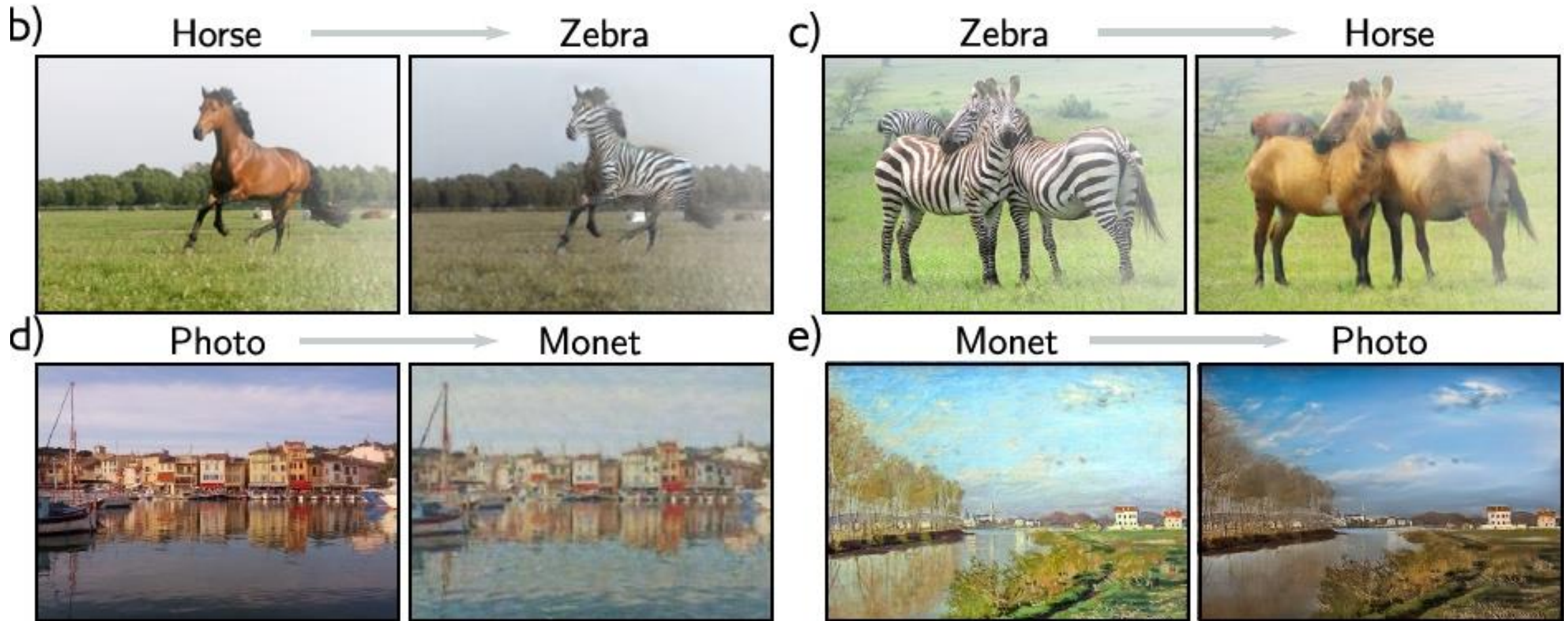


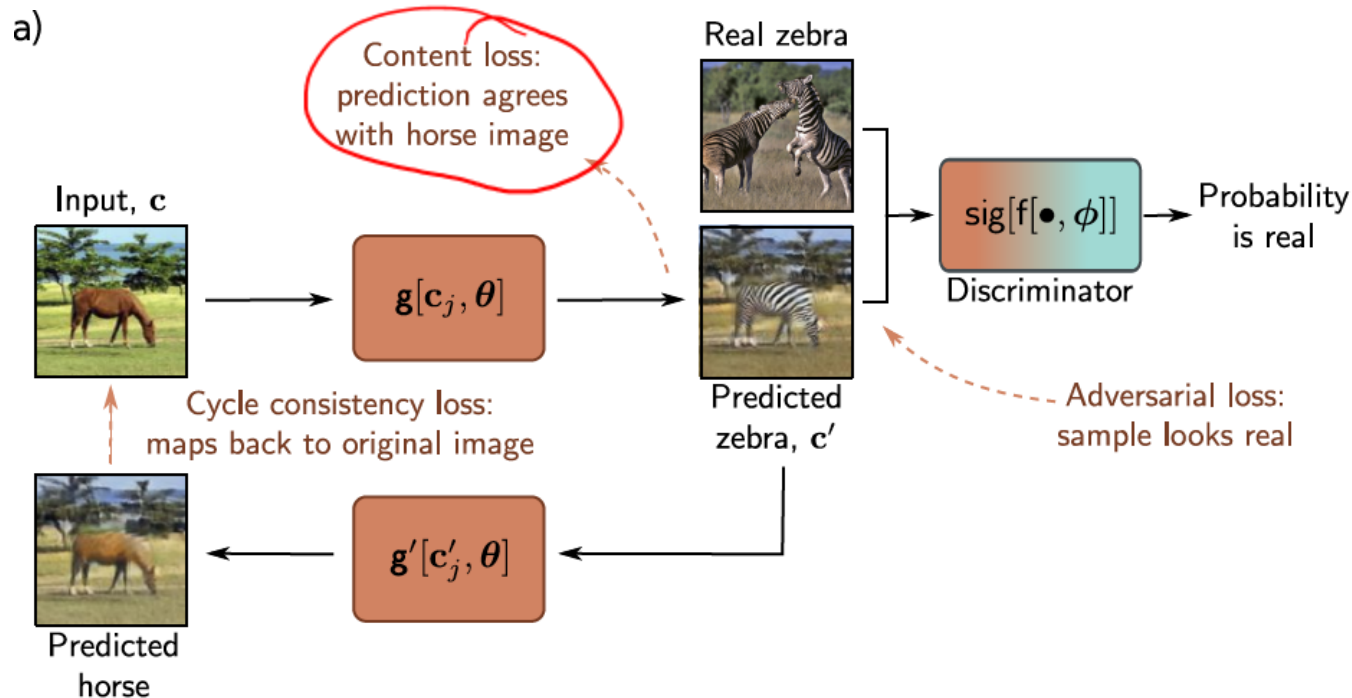
Image translation: CycleGAN



Challenge: very few style transfer pairs for training. Also very few Monet samples.

Image translation: CycleGAN

$$F(G(x)) \approx x \quad G(F(y)) \approx y$$



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 (ℓ_1 norm)
2. Discriminator loss (real vs fake)
3. Cycle-consistency loss

2nd model is also trained.

Encourages the generator to be reversible, and **to actually use its input.**

[Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks \(2017\)](#)

Input



Output



horse \rightarrow zebra

Inverting the Generator

Given the generator of a GAN and an image, can we find its latent?

- This is pretty straightforward with gradient descent.
 - More reliable than training inverse?
- Why?
 - Because the latents tend to be useful.
 - Similar latent ~ similar image ~ similar semantics.

[Inverting The Generator Of A Generative Adversarial Network \(2016\)](#)

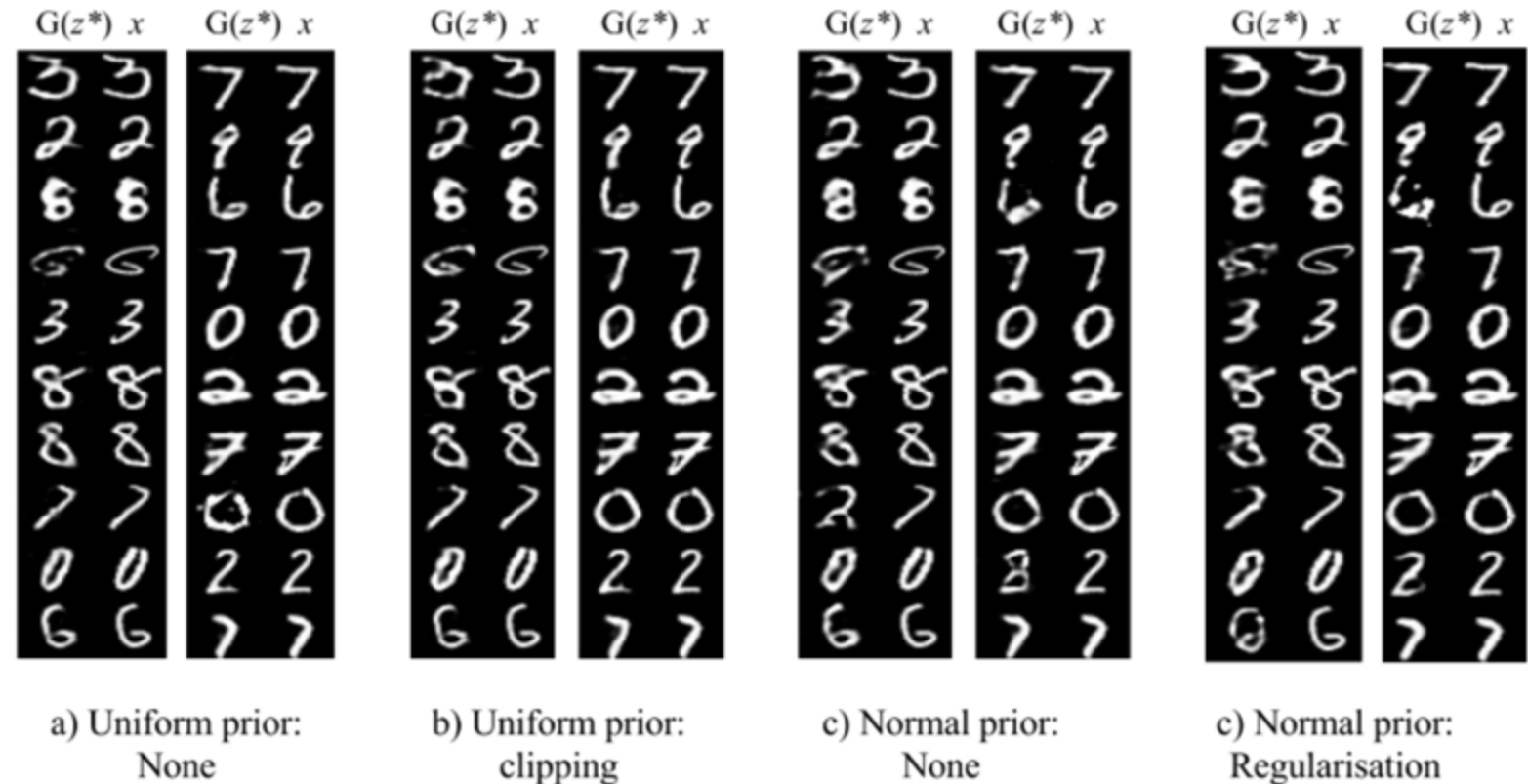


Figure 2: **Reconstructions for MNIST:** inverting a generator trained using a uniform prior (a-b) and a normal prior (c-d). The original image, x is on the right, while the inverted image is on the left $G(z^*)$.

Use this to check if an image came out of a particular generator?

Takeaways

- Generative adversarial networks are surprisingly effective at image generation.
 - If you can get them to converge.
 - If there is no mode collapse.
- This was one of the feet in the door to get generative image models working.
 - Identified many challenges for later kinds of models to avoid.
 - Still active research here.

Any Questions?



- Intriguing properties of neural networks
- Adversarial features
- Generative adversarial networks (GANs)
- GAN loss functions
- GAN examples
- Controlling GANs