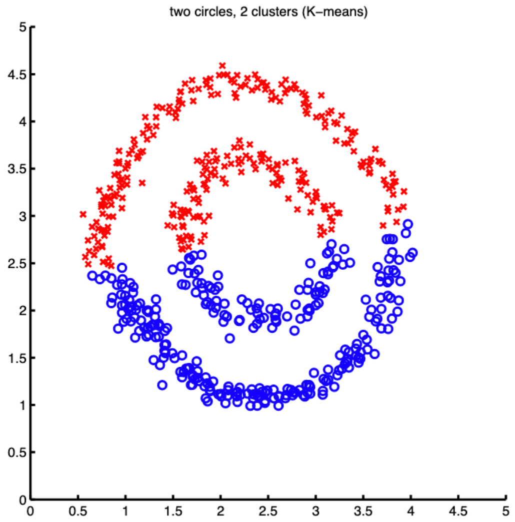
Lecture 25: Spectral clustering

- Unsupervised learning
  - Dimensionality reduction
    - PCA
    - Auto-encoder
  - Clustering
    - k-means
    - **Spectral**,t-SNE,UMAP
  - Generative models
  - Density estimation

k-means and GMMs are inherently linear

- It tries to find linear boundaries between centers
- It fails completely on non-linearly clustered datasets such as

K-means

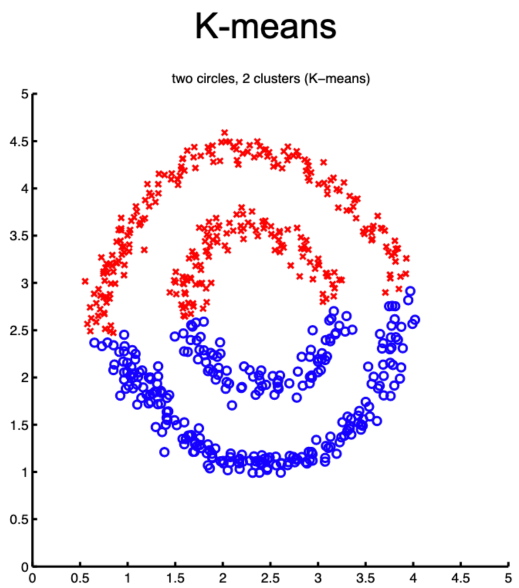


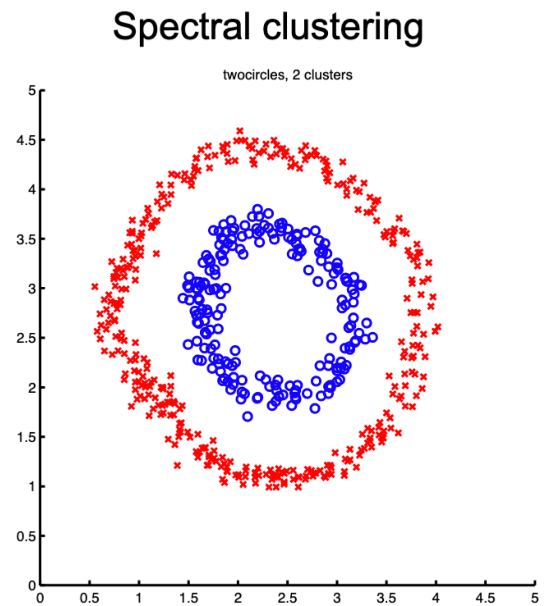
- Any suggestions?

[Shi,Malik,'00],[Ng,Jordan,Weiss,'01]

Spectral clustering

- Main idea:
  - Transform the dataset into a graph encoding similarities
  - Use eigenvalues (also called spectrum) and vectors of a graph to cluster

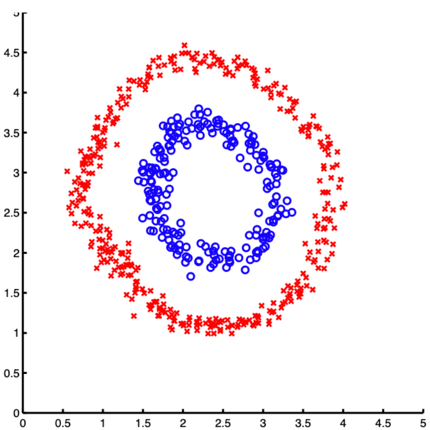


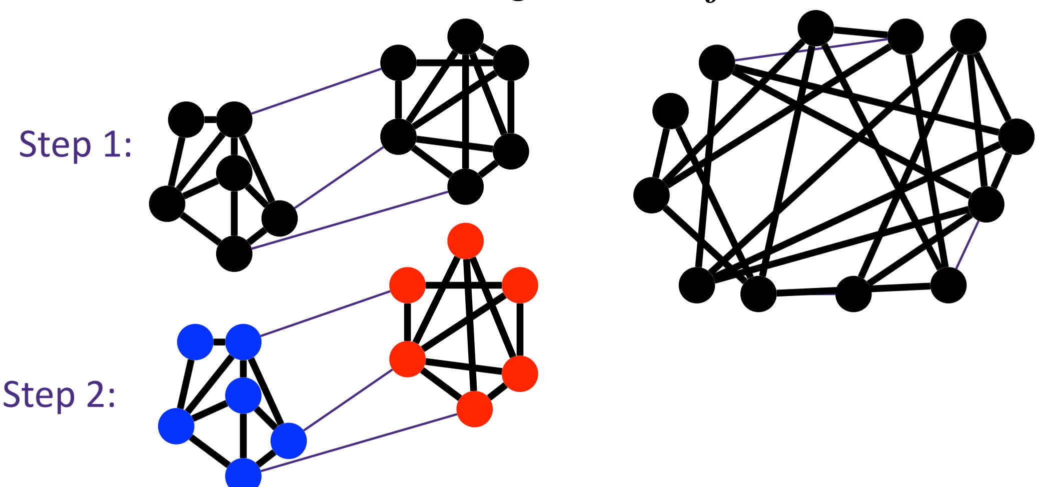


[Shi,Malik,'00],[Ng,Jordan,Weiss,'01]

Step 1. From dataset to a graph

- Given $\mathscr{D} = \{x_i \in \mathbb{R}^d\}_{i=1}^n$, create a graph with n nodes and weighted edges $\{w_{ij}\}$, where each node represents each sample and each edge measures the similarity between the two nodes
  - Example 1: Gaussian kernel
  $$w_{ij} = e^{-\frac{\|x_i - x_j\|_2^2}{\sigma^2}}$$
  - Example 2: k-nearest neighbor graph
  $w_{ij} = 1$ if j is one of k-nearest neighbors of i or
  i is one of k-nearest neighbors of j



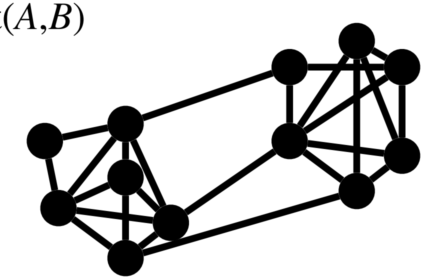


Step 2. Graph partitioning

- Once we have a similarity graph, how do we partition it?
- Can we use **minimum cut** for a graph $G(V, E)$?
  - Set of nodes $V = \{1, \ldots, n\}$
  - Set of edges $E = \{(i, j)\}$
  - If it is a weighted graph we have weights $\{w_{ij}\}_{(i,j) \in E}$
- **Minimum cut** of a graph is a partition $A \cup B = V$ and $A \cap B = \emptyset$ such that

$$\arg\min_{A,B} \underbrace{\sum_{i \in A} \sum_{j \in B} w_{i,j}}_{\text{cut}(A,B)}$$

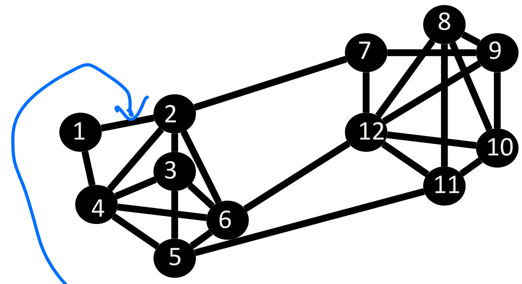
$$\arg\min_{A,B} \ \text{cut}(A,B) \times \left(\frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)}\right)$$



$$\text{Vol}(A) = \sum_{i,j \in A} w_{ij}$$

Step 2. Graph partition using Graph Laplacian

- Definitions (we will define it for unweighted graphs, but everything naturally generalizes to weighted graphs)
  - **Adjacency matrix** of a graph $A \in \mathbb{R}^{n \times n}$
    $A_{ij} = 1$ if $(i, j) \in E$
    0 otherwise
  - **Degree** of a node i, is $d_i = \sum_{j=1}^{n} A_{ij}$, which is number of edges connected to node i
  - Define $D \in \mathbb{R}^{n \times n}$ as a diagonal matrix with the degrees of each node in the diagonal
  - The **Graph Laplacian** of a graph is defined as
    $$L_G = D - A$$

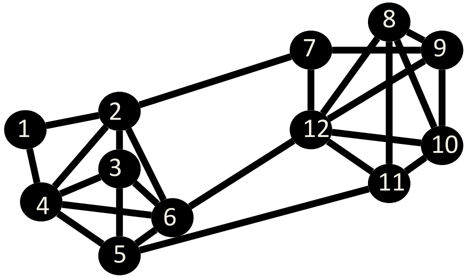


$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \end{bmatrix}$$

Step 2. Graph partition using Graph Laplacian

- Graph Laplacian $L_G = D - A$ can capture some structure of the graph
- Consider placing each node in 1-dim line at positions $x = [x_1, x_2, \ldots, x_n]$



quadratic form of L_G is useful in capturing the structure of the graph:

$$x^T L_G x = \sum_i d_i x_i^2 - \sum_{(i,j)\in E} 2x_i x_j$$

$$d_i = \sum_{j:(i,j)\in E} 1 \longrightarrow = \sum_i \sum_{j:(i,j)\in E} x_i^2 - \sum_{(i,j)\in E} 2x_i x_j$$

$$\sum_{i=1}^{n} \sum_{j:(i,j)\in E} 1 = \sum_{i=1}^{n} d_i = 2|E| = \sum_{(i,j)\in E} 2 \longrightarrow = \sum_{(i,j)\in E} 2x_i^2 - 2x_i x_j$$

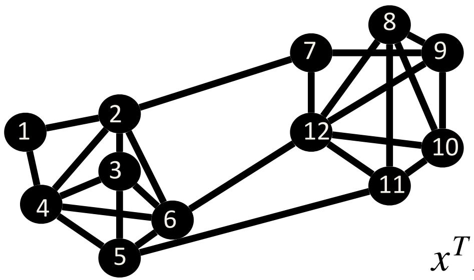
$$= \sum_{(i,j)\in E} x_i^2 + x_j^2 - 2x_i x_j$$

$$= \sum_{(i,j)\in E} (x_i - x_j)^2$$

$$\min_{x \in \mathbb{R}^n} x^T L_G x.$$

Step 2. Graph partition using Graph Laplacian

- Graph Laplacian $L_G = D - A$ can capture some structure of the graph
- Consider placing each node in 1-dim line at positions $x = [x_1, x_2, \ldots, x_2]$



$$x^T L_G x = \sum_{(i,j)\in E} (x_i - x_j)^2$$

- If we want a good graph partition, we want to place nodes such that the distance between connected nodes are smaller
- This naturally leads to the following problem:

$$\arg\min_{x\in\mathbb{R}^n} \; x^T L_G x = \sum_{(i,j)\in E} (x_i - x_j)^2$$

- There is a trivial solution to this problem: $x_i = 1$ for all i, which achieves the minimum value of zero, so we change it to

$$\arg\min_{x\in\mathbb{R}^n} \; x^T L_G x = \sum_{(i,j)\in E} (x_i - x_j)^2 \qquad \text{subject to } x^T\mathbf{1} = 0$$

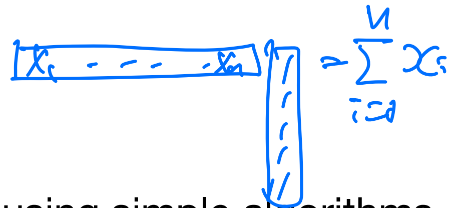
Step 2. Graph partition using Graph Laplacian

- To solve graph partitioning, we solve

$$\arg\min_{x \in \mathbb{R}^n} \; x^T L_G x = \sum_{(i,j)\in E} (x_i - x_j)^2$$

$$\text{subject to } x^T \mathbf{1} = 0 \quad = \sum_{i=1}^{n} x_i$$

$$\|x\|_2 = 1$$



  and place nodes as per x, and find a partition using simple algorithms like k-means

- It turns out that the above optimization has a efficient solver, because The optimal x turns out to be the second smallest eigen vector of the graph Laplacian L_G
- Since, eigen values of a matrix is also called a spectrum, this is called a spectral clustering algorithm

Spectral clustering

- Step 1. Define a similarity graph $G(V, E, W)$
- Step 2. Compute the Graph Laplacian

$$L_G = D - W$$

where $D$is a diagonal matrix with $D_{ii} = \sum_{j=1}^{n} w_{ij}$ $\rightarrow$ $x^T L_G x = \sum_{(i,j)\in E} w_{ij}(x_i - x_j)^2$

  - let x be the Eigen vector corresponding to the second smallest Eigen value
  - Place samples according to x and apply k-means clustering

- instead of using just the second smallest Eigen pair, you can use multiple smallest Eigen pairs

$\{x_i \in \mathbb{R}^d\}_{i=1}^n \rightarrow G \rightarrow L_G \rightarrow V \in \mathbb{R}^n \rightarrow \boxed{\{v_i\}_{i=1}^n}$

cluster k-means.

Questions?

https://www.whichfaceisreal.com/

Deep Generative Models

- Unsupervised learning
  - Dimensionality reduction
    - PCA
    - Auto-encoder
  - Clustering
    - k-means
    - Spectral,t-SNE,UMAP
  - **Generative models**
  - Density estimation

Deep generative model

- traditional parametric generative model
  - Gaussian:

$$f_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

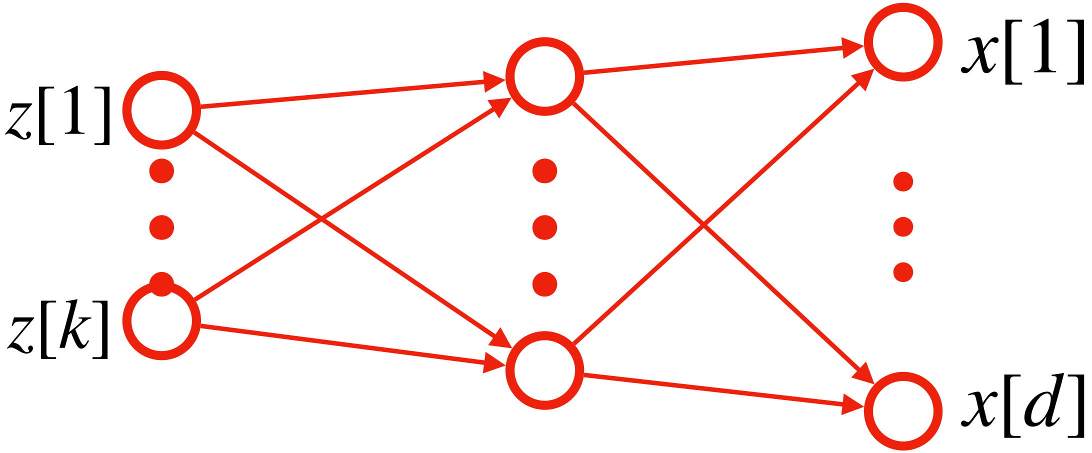
  - Gaussian Mixture Models (GMM)

$$f_{\{\mu_i\},\{\sigma_i\},\{\pi_i\}}(x) = \sum_{i=1}^{k} \pi_i \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$$

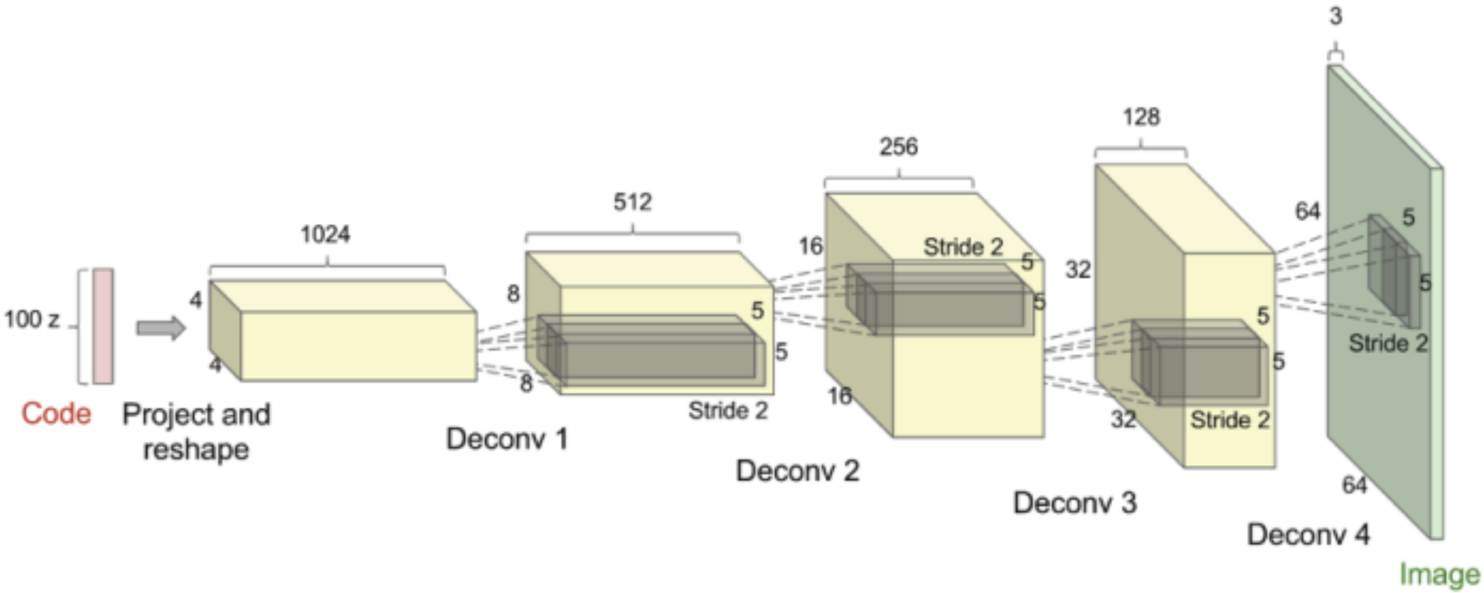
- deep generative model
  - easy to sample
  - high representation power
  - but no tractable evaluation of the density (i.e. p.d.f.)

Deep generative model

- sampling from a deep generative model, parametrized by w
    - first sample a **latent code** $z \in \mathbb{R}^k$ of small dimension $k \ll d$, from a simple distribution like standard Gaussian $N(0, \mathbf{I}_{k \times k})$
    - pass the code through a neural network of your choice, with parameter w
    - the output sample $x \in \mathbb{R}^d$ is the sample of this deep generative model



Deep generative model using deep deconvolutional layers



Generative model

- a task of importance in unsupervised learning is fitting a generative model
- classically, if we fit a parametric model like mixture of Gaussians, we write the likelihood function explicitly in terms of the model parameters, and maximize it using some algorithms
- $$\text{maximize}_w \sum_{i=1}^{n} \log \left(\underbrace{P_w(x_i)}_{\text{p.d.f.}} \right)$$
- deep generative models use neural networks, but the likelihood of deep generative models cannot be evaluated easily, so we use alternative methods

Goal

- Given examples $\{x_i\}_{i=1}^{n}$ coming i.i.d from an unknown distribution $P(x)$, train a generative model that can generate samples from a distribution close to $P(x)$

These are computer generated images from the "bigGAN".

Adversarial training

- Classification
  - Consider the example of SPAM detection
  - Each sample x_i is an email
  - Distribution of **true email** is $P(x)$
  - Suppose spammers generate **spams** with distribution $Q(x)$
  - Spam detection: Typical classification task
    - Generate samples from true emails and label them $y_i = 1$
    - Generate samples from spams and label them $y_i = 0$
    - Using these as training data, train a classifier that outputs

$$\mathbb{P}(y_i = 1 \,|\, x_i) \simeq \frac{1}{1 + e^{-f_\theta(x)}}$$

for some neural network $f_\theta(\,\cdot\,)$ with parameter θ
(this is the **logistic model** for binary classification)

Adversarial training

- Applying logistic regression, we want to solve

$$\max_{\theta} \sum_{i:y_i=1} \log\left(\frac{1}{1+e^{-f_\theta(x_i)}}\right) + \sum_{i:y_i=0} \log\left(1-\frac{1}{1+e^{-f_\theta(x_i)}}\right)$$

- in **adversarial training**, it is customary to write

$$D_\theta(x) = \frac{1}{1+e^{-f_\theta(x)}}$$

which is called a **discriminator**

- and find the "best" discriminator by solving for

$$\max_{\theta} \mathscr{L}(\theta) = \sum_{x_i \sim P(\cdot)} \log D_\theta(x_i) + \sum_{x_i \sim Q(\cdot)} \log(1 - D_\theta(x_i))$$

as 1 labelled examples come from real distribution $P(\cdot)$
and 0 labelled examples come from spam distribution $Q(\cdot)$

Adversarial training

- Suppose now that the **spam detector (i.e. the discriminator)** is fixed, then the spammer's job is to generate spams that can fool the detector by making the likelihood of the spams being classified as spams **small**:

$$\min_{Q(\cdot)} \mathscr{L}(\theta) = \underbrace{\sum_{x_i \sim P(\cdot)} \log D_\theta(x_i)}_{\text{does not depend on } Q(\cdot)} + \sum_{x_i \sim Q(\cdot)} \log(1 - D_\theta(x_i))$$

- where 0 labelled examples are coming from the distribution $Q(\cdot)$, which is modeled by a **deep neural network generative model,** i.e. $x_i = G_w(z_i)$ where $z_i \sim N(0, \mathbf{I}_{k \times k})$.
- The minimization can be solved by finding. The "best" generative model that can fool the discriminator

$$\min_{w} \mathscr{L}(w, \theta) = \underbrace{\sum_{x_i \sim P(\cdot)} \log D_\theta(x_i)}_{\text{does not depend on } Q(\cdot)} + \sum_{x_i \sim Q(\cdot)} \log\Big(1 - D_\theta\big(G_w(z_i)\big)\Big)$$

Adversarial training

- Now we have a game between the spammer and the spam detector:

$$\min_{w} \max_{\theta} \sum_{x_i \sim P(\cdot)} \log D_\theta(x_i) + \sum_{z_i \sim N(0,\mathbf{I})} \log(1 - D_\theta(G_W(z_i)))$$

- Where $P(\cdot)$ is the distribution of real data (true emails), and $Q(\cdot)$ is the distribution of the generated data (spams) that we want to train with a **deep generative model**
- jointly training the discriminator and the generator is called **adversarial training**
- Alternating method is used to find the solution

Alternating gradient descent for adversarial training

- Gradient update for the **discriminator** (for fixed w)

$$\max_{\theta} \sum_{x_i \sim P(\cdot)} \log D_\theta(x_i) + \sum_{x_i \sim Q(\cdot)} \log(1 - D_\theta(x_i))$$

- First sample n examples from real data (in the training set) and the generator data $x_i \sim G_w(z_i)$
(for the current iterate of the generator weight w)
- compute the gradient for those $2n$ samples using back-propagation
- Update the discriminator weight θ by subtracting the gradient with a choice of a step size

Alternating gradient descent for adversarial training

- gradient update for the **generator** (for fixed θ)

$$\min_{w} \sum_{x_i \sim P(\cdot)} \log D_\theta(x_i) + \sum_{z_i \sim N(0,\mathbf{I})} \log(1 - D_\theta(G_w(z_i)))$$

- Consider the gradient update on a single sample

$$\min_{w} \mathscr{L}(w, z_i) \;=\; \log(1 - D_\theta(G_w(z_i)))$$

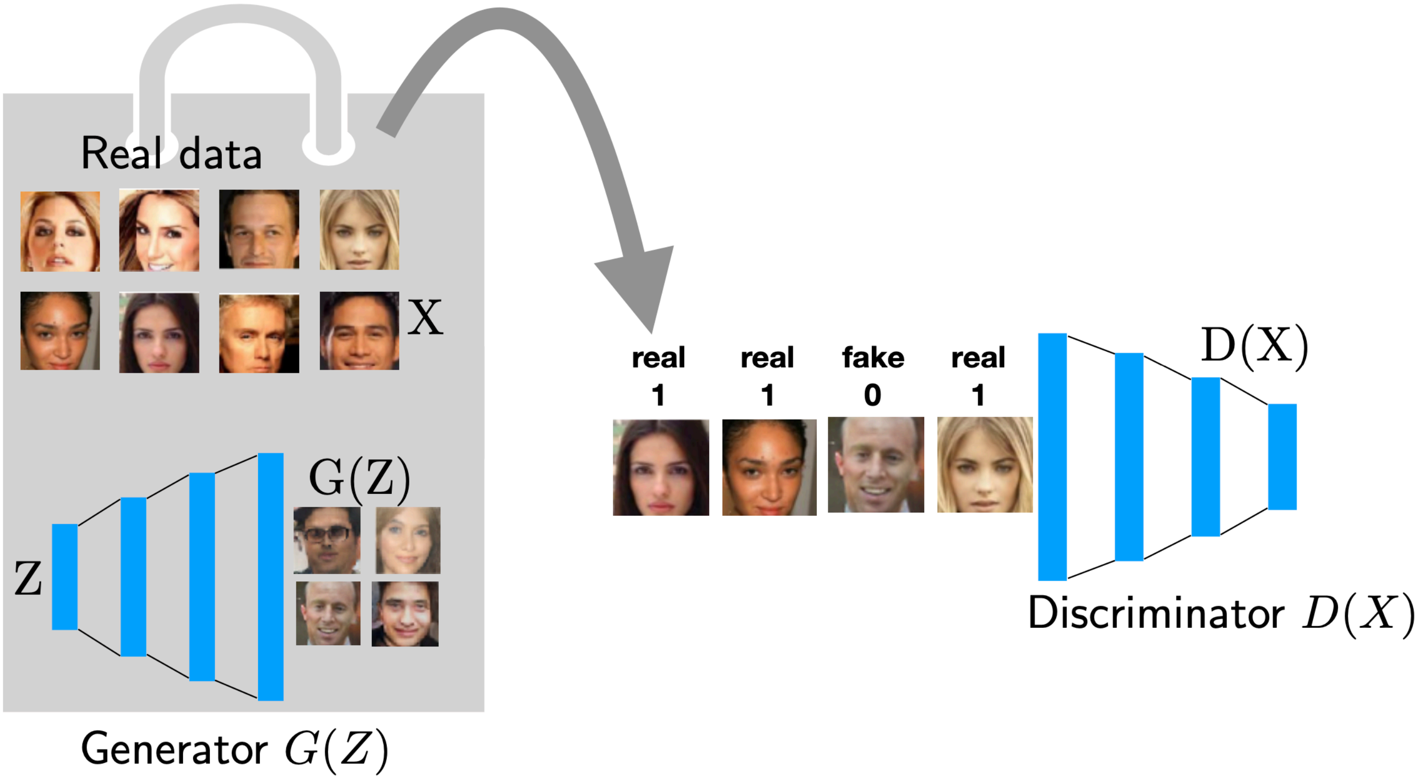
for a single $z_i \sim N(0,\mathbf{I})$ sampled from a Gaussian

- The gradient update is

$$\begin{aligned} w \;&=\; w - \eta \nabla_w \mathscr{L}(w, z_i) \\ &=\; w - \eta \; \nabla_w G_w(z_i) \; \nabla_x D_\theta(x) \, \frac{-1}{1 - D_\theta(x)} \end{aligned}$$

with $x = G_w(z_i)$

This gives a new way to train a deep generative model



$$\min_G \max_D V(G, D)$$

Not only is GAN amazing in generating realistic samples

http://whichfaceisreal.com

It opens new doors to exciting applications

- Cvcle-GAN

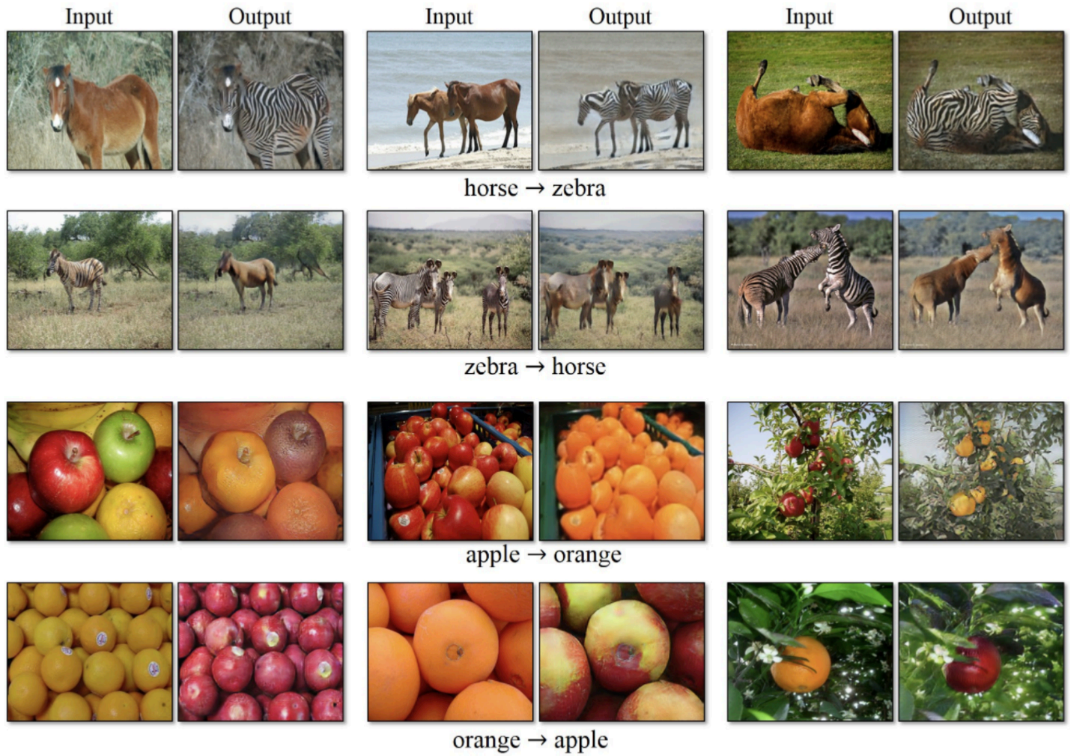


Figure 3: Street scene image translation results. For each pair, left is input and right is the translated image.

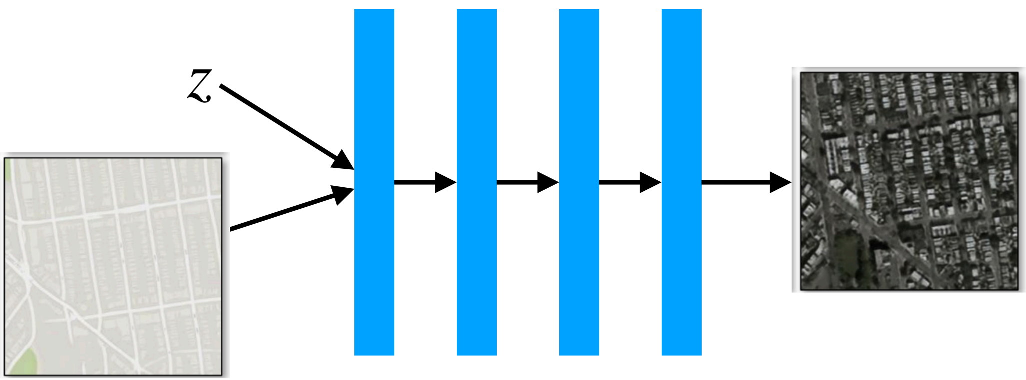
https://www.youtube.com/watch?v=PCBTZh41Ris

Style transfer with generative model

- If we have paired training data,

- And want to train a generative model G(x,z)=y,

- This can be posed as a regression problem

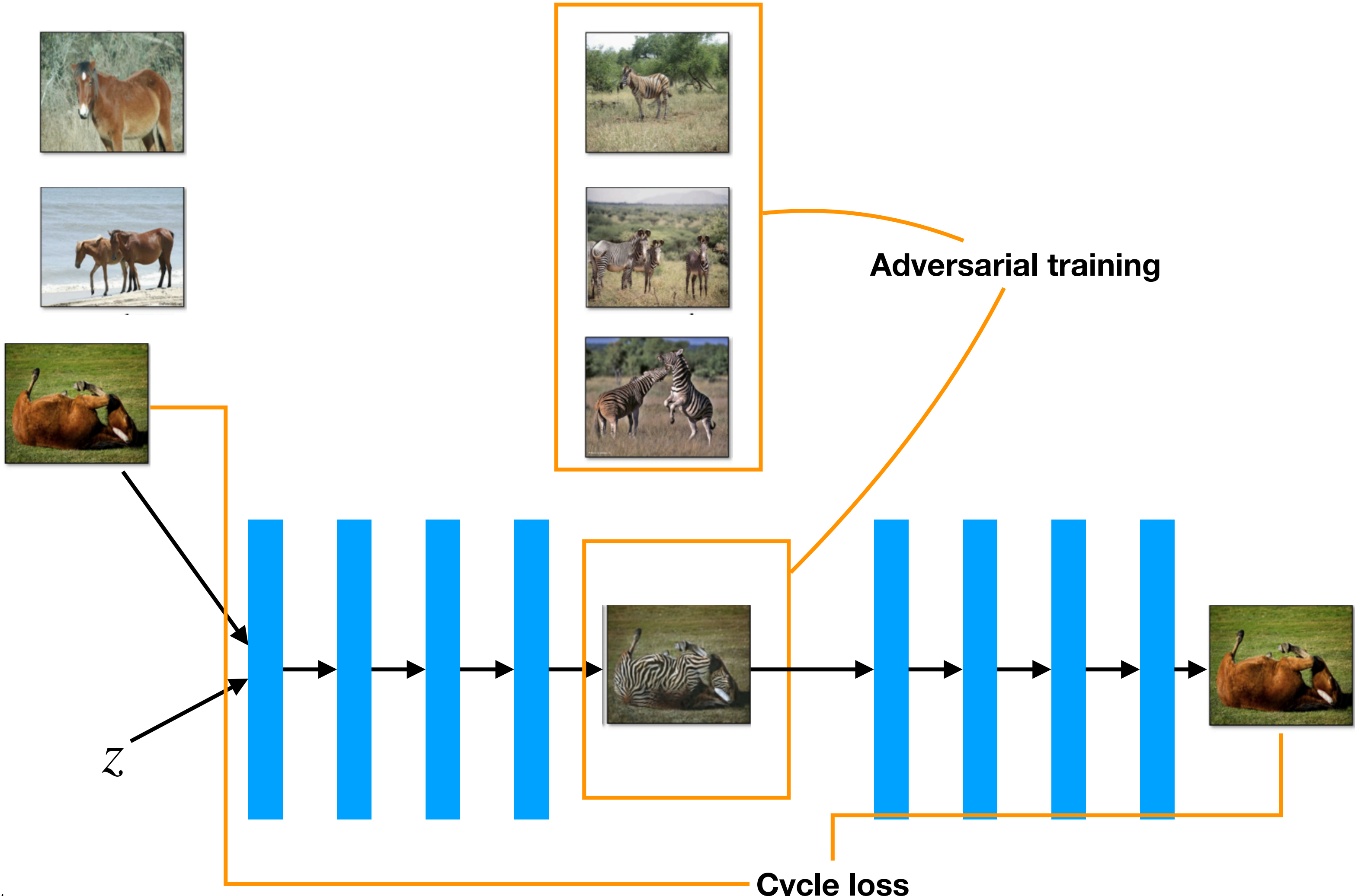


How do we do style transfer without paired data? Cycle-GAN

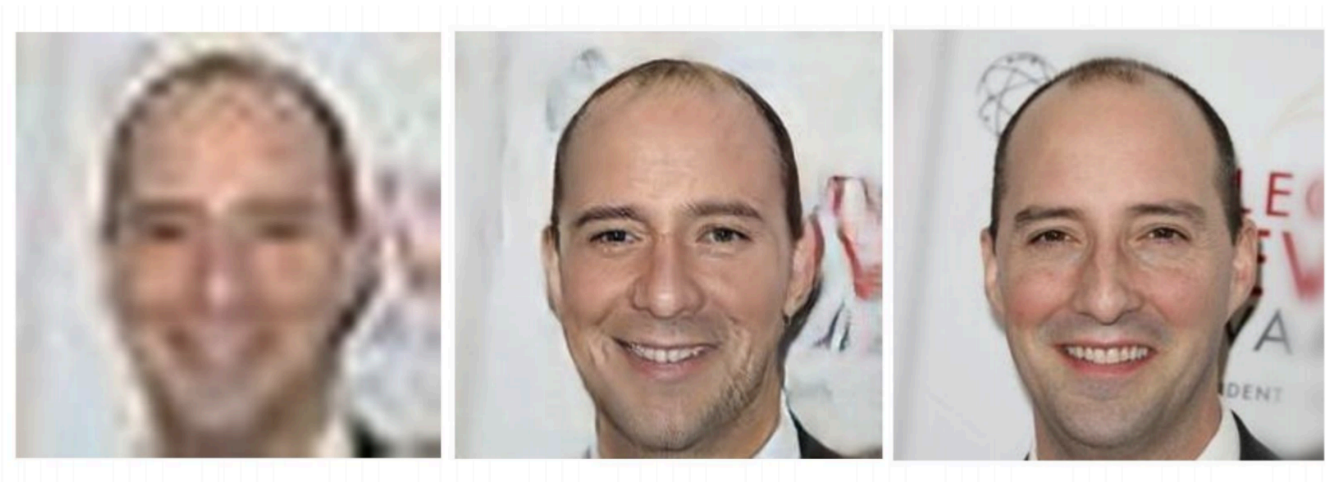




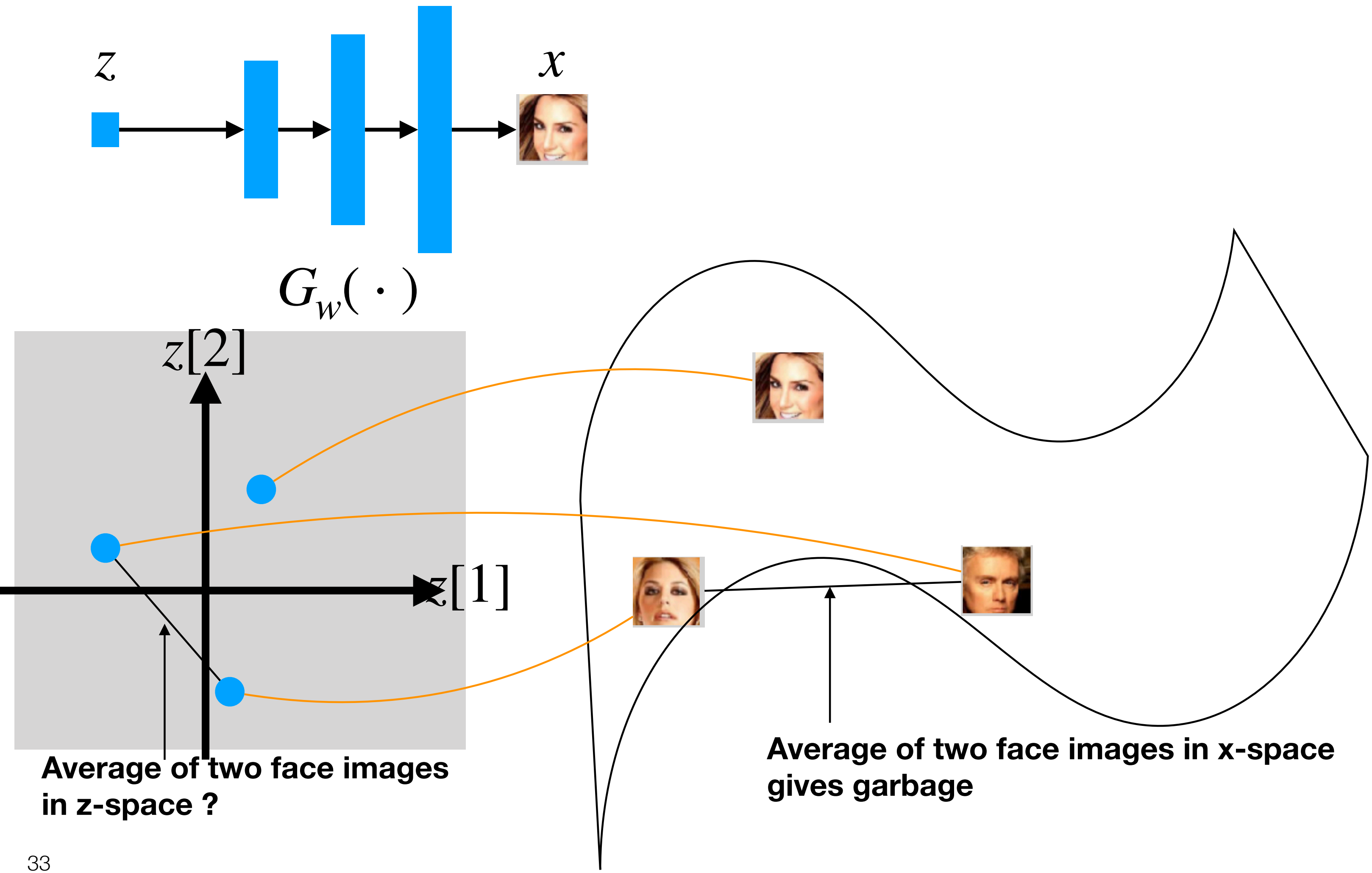

How do we do style transfer without paired data? Cycle-GAN



Super resolution



The learned latent space is important

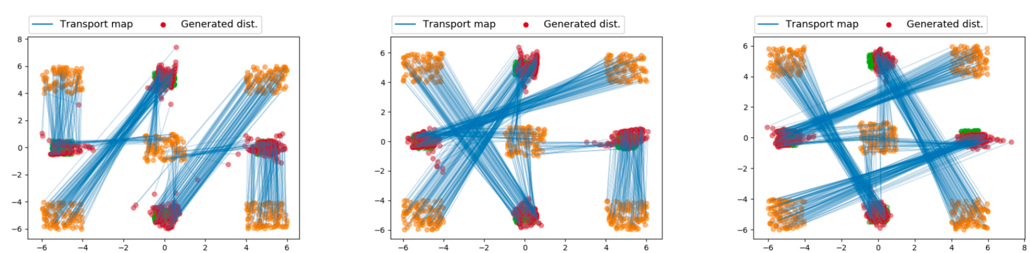


How do we check if we found the right manifold (of faces)?

- **latent traversal**

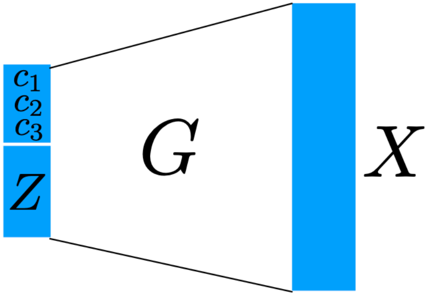
Can we make the relation between the latent space and the image space more meaningful?

- **Disentangling**
  - **GANs learn arbitrary mapping from z to x**
  - **As the loss only depends on the marginal distribution of x and not the conditional distribution of x given z (how z is mapped to x)**

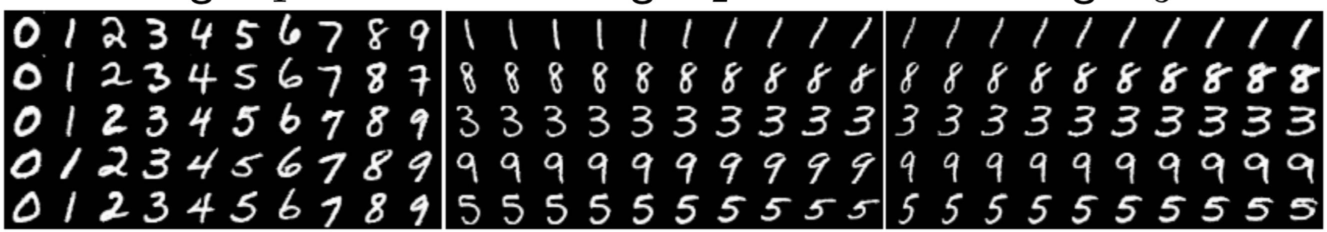


Disentangling seeks meaningful mapping from z to x

- there is no formal (mathematical) universally agreed upon definition of disentangling

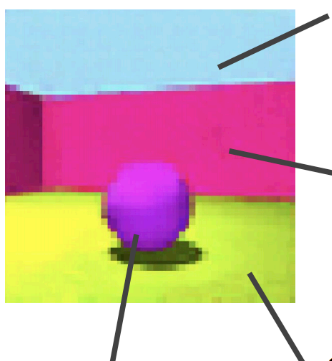


$\Rightarrow$ change c_1 $\Rightarrow$ change c_2 $\Rightarrow$ change c_3



- informally, we seek latent codes that
  - are "informative" or make "noticeable" changes
  - are "uncorrelated" or make "distinct" changes

Decompose data into a set of underlying **human-interpretable** factors of variation



Explainable models

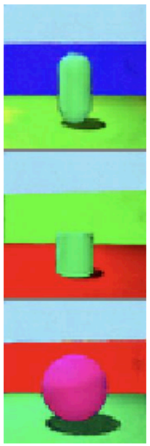
What is in the scene?

Controllable generation

Generate a red ball instead

Fully-supervised case

Strategy: Label everything



c_1 c_2 c_3

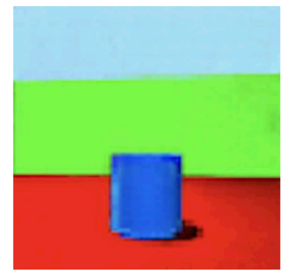
{dark blue wall, green floor, green oval}

{green wall, red floor, green cylinder}

{red wall, green floor, pink ball}

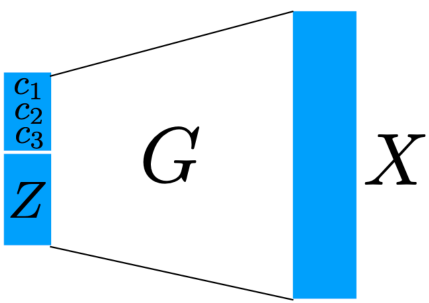
Controllable generation as **label-conditional generative modeling**

green wall, red floor, blue cylinder



$Q(x) \sim P(x)$

Fake Real

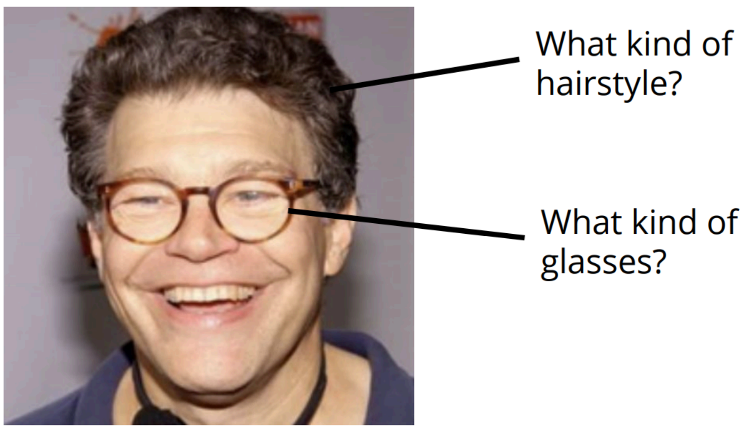


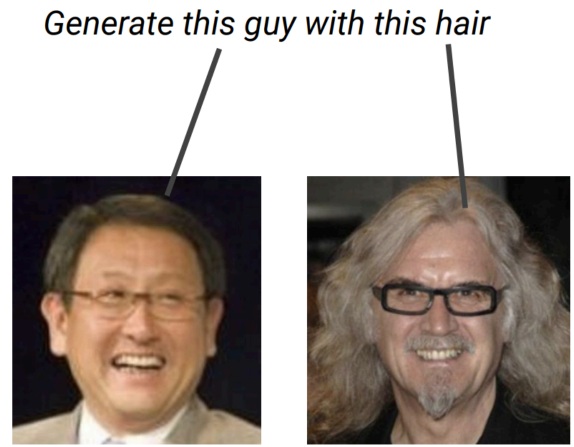
Train a **conditional GAN**, where

(c_1, c_2, c_3) is a numerical representation of the **labels** given in the training data, and Z is drawn from Gaussian

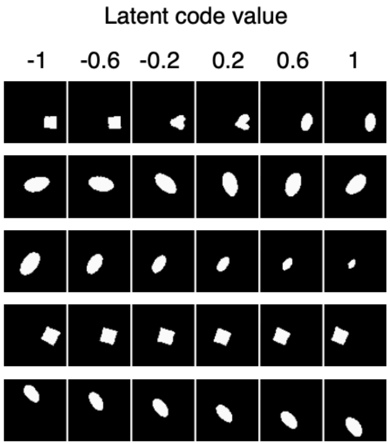
$Q(x|c) \sim P(x|c)$

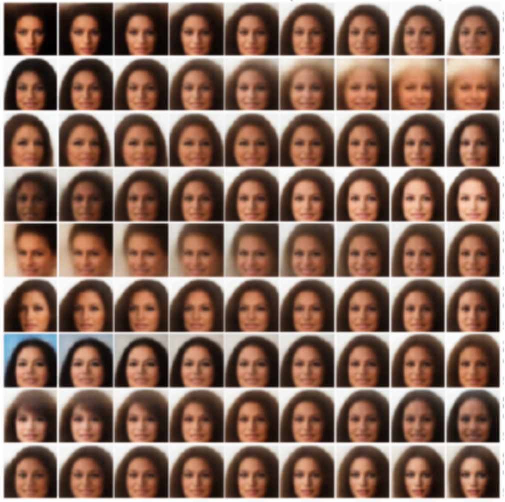
However, some properties are hard to represent numerically

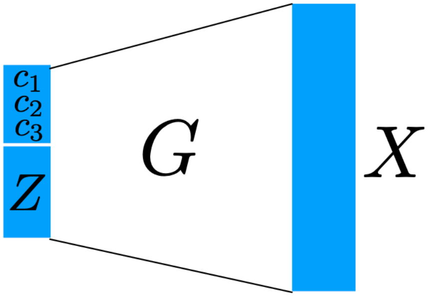




Unsupervised training of Disentangled GAN



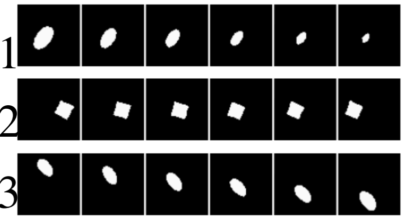




Disentangled GAN training: InfoGAN-CR, 2019

- 1. As in standard GAN training, we want $G_w(z)$ to look like training data (which is achieved by adversarial loss provided by a discriminator)

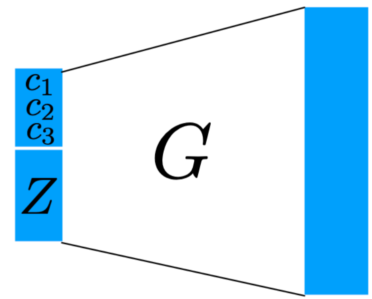
  D( **) = {real,fake}** $\Rightarrow Q(x) \sim P(x)$



- 2. We also want the controllable latent code c to be predictable from the image
  - add a NN regressor that predicts $\hat{c}(x)$, and train the generator that makes the prediction accuracy high (note that both this predictor and the generator works to make the prediction accurate, unlike adversarial loss).

NN prediction

minimize $\| \hat{c}($  $) - c\|^2$



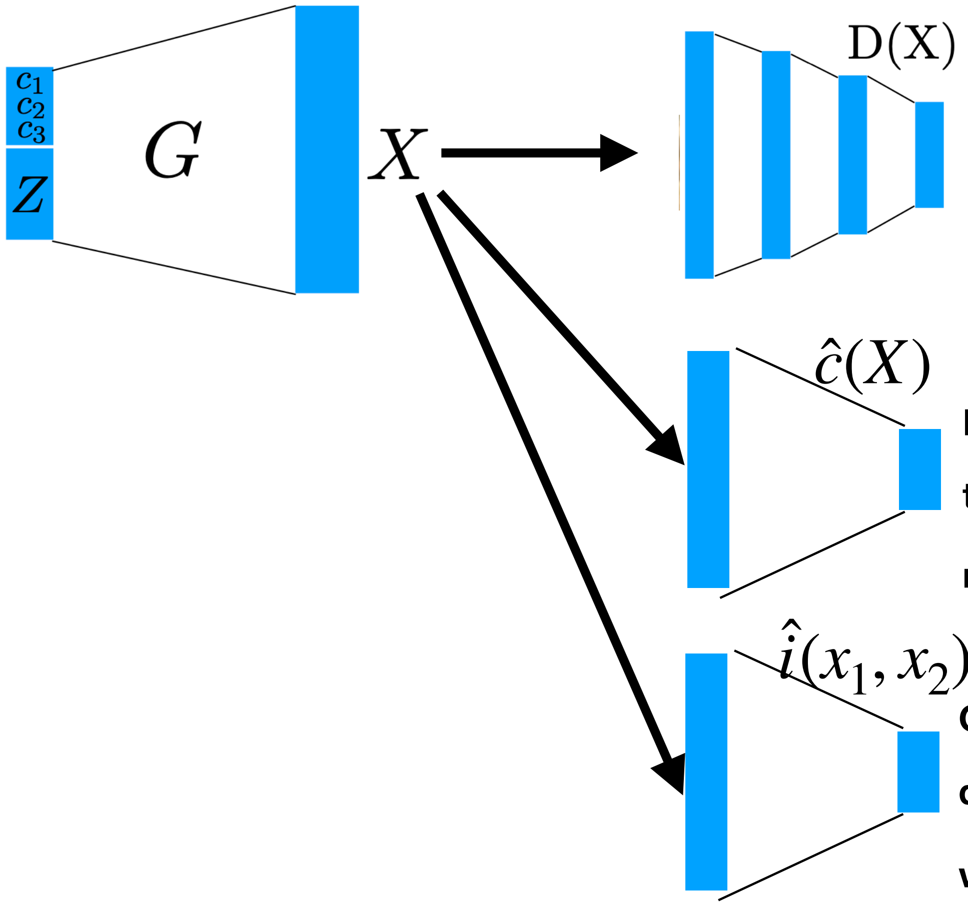
- 3. We also want each code to control distinct properties
  - add a NN that predicts which code was changed

NN prediction

$\hat{i}($  $) \simeq i$

Disentangling with contrastive regularizer

- To train a disentangled GAN, we use contrastive regularizer



Discriminator encourages

output X to be realistic

Predictor makes sure that

the changes in C make

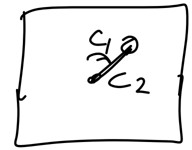
noticeable changes in X

Contrastive regularizer

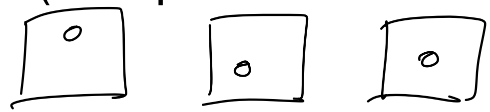
detects which latent code C_i

was the same in a paired (x_1, x_2)

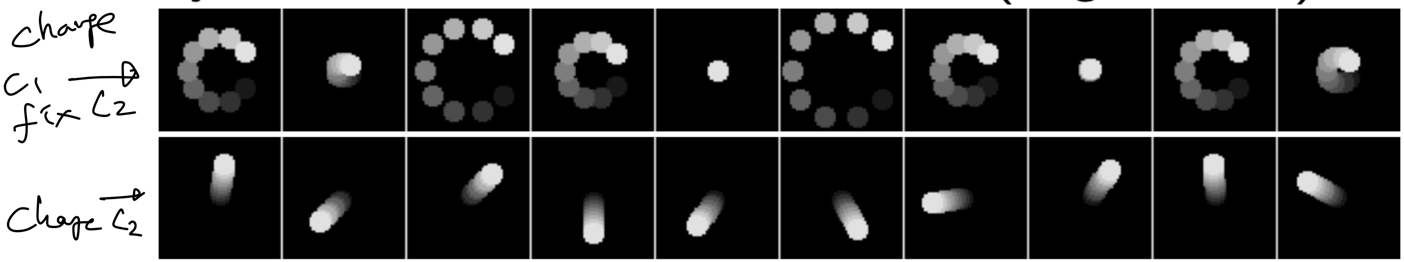
But is still challenging



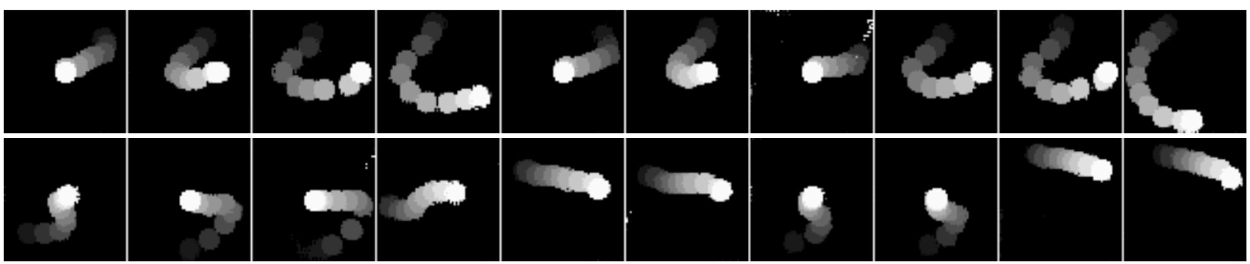
- Synthetic training data (with planted disentangled representation)



Synthetic data with two attributes (angle,radius)

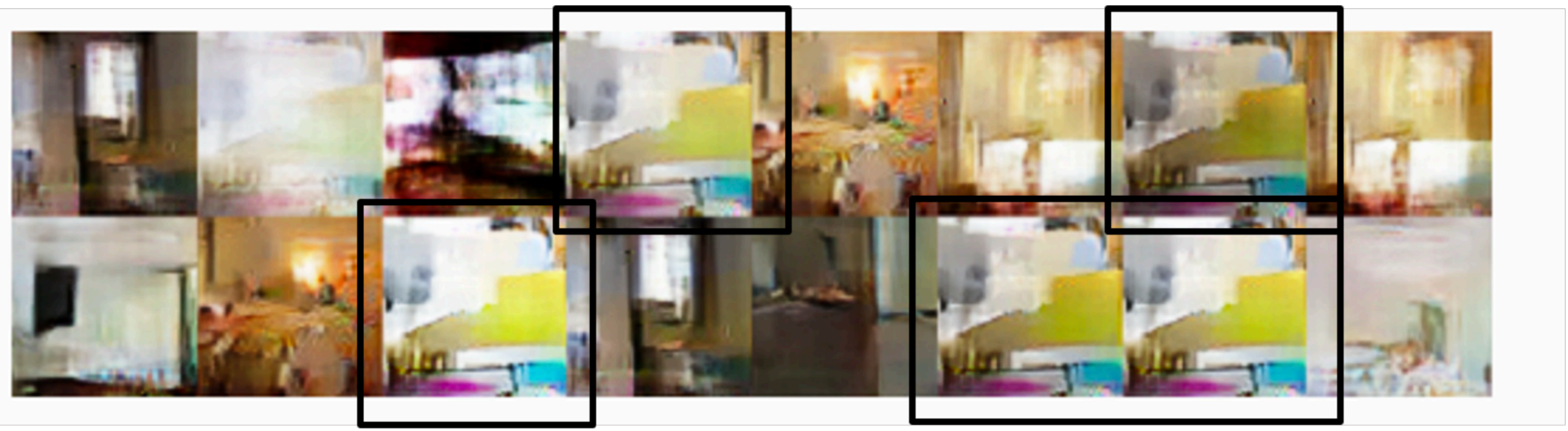


- Trained Disentangled GAN (latent traversal)



Challenges in training GANs

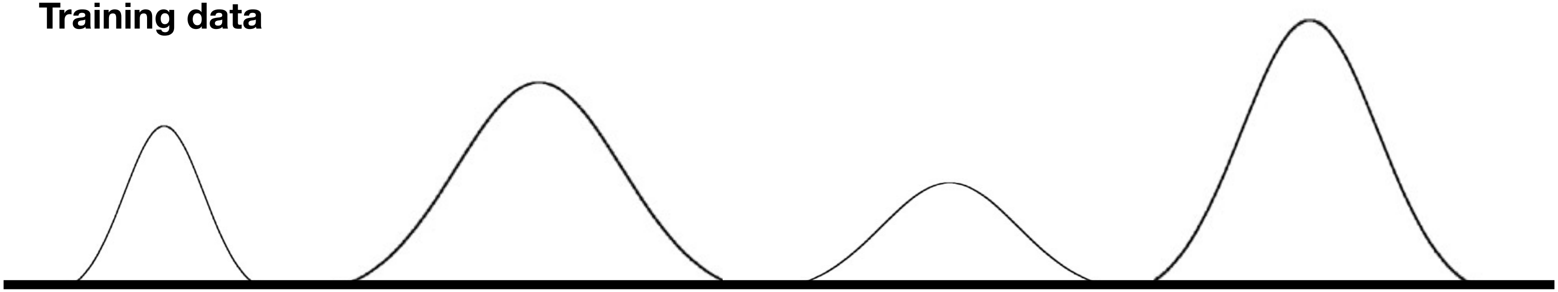
- GAN training suffers from **mode collapse**
- this refers to the phenomenon where the generated samples are not as diverse as the training samples



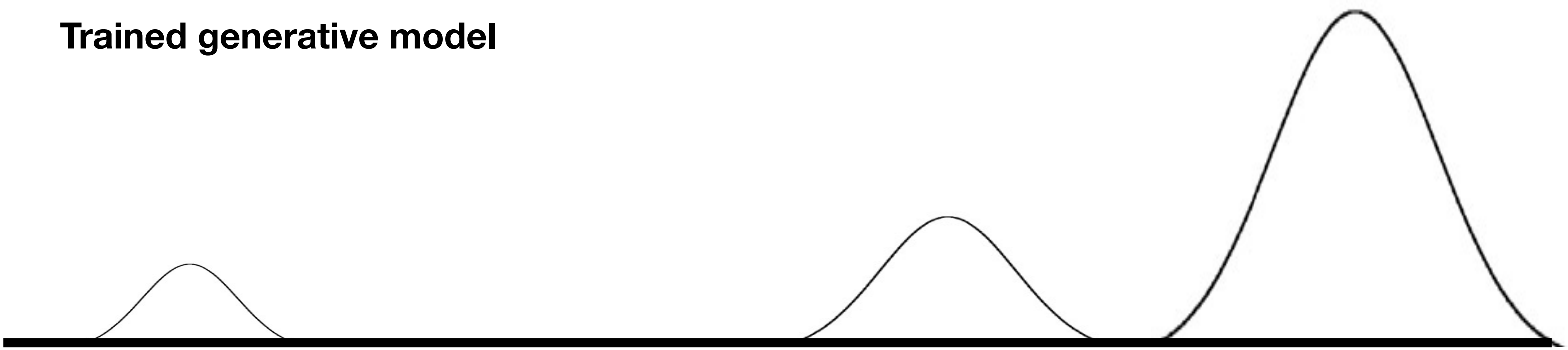
Arjovsky et al., 2017

Mode collapse

Training data

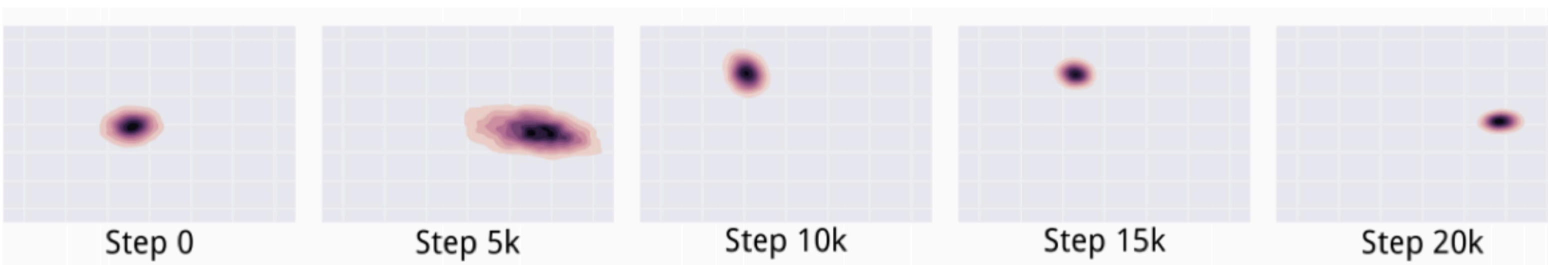


Trained generative model



Mode collapse

- True distribution is a mixture of Gaussians



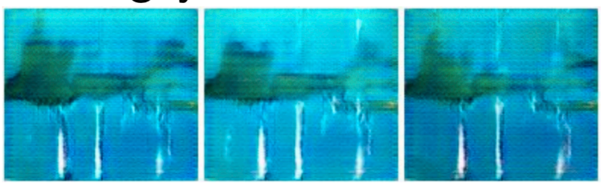
Source: Metz et al., 2017

- The generator distribution keeps oscillating between different modes

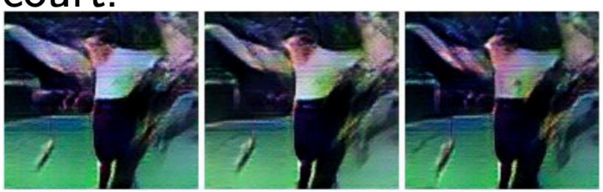
Mode collapse

- “A man in a orange jacket with sunglasses and a hat ski down a hill.”

- “This guy is in black trunks and swimming underwater.”



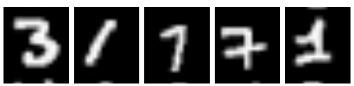
- “A tennis player in a blue polo shirt is looking down at the green court.”



[“Generating interpretable images with controllable structure”, by Reed et al., 2016]

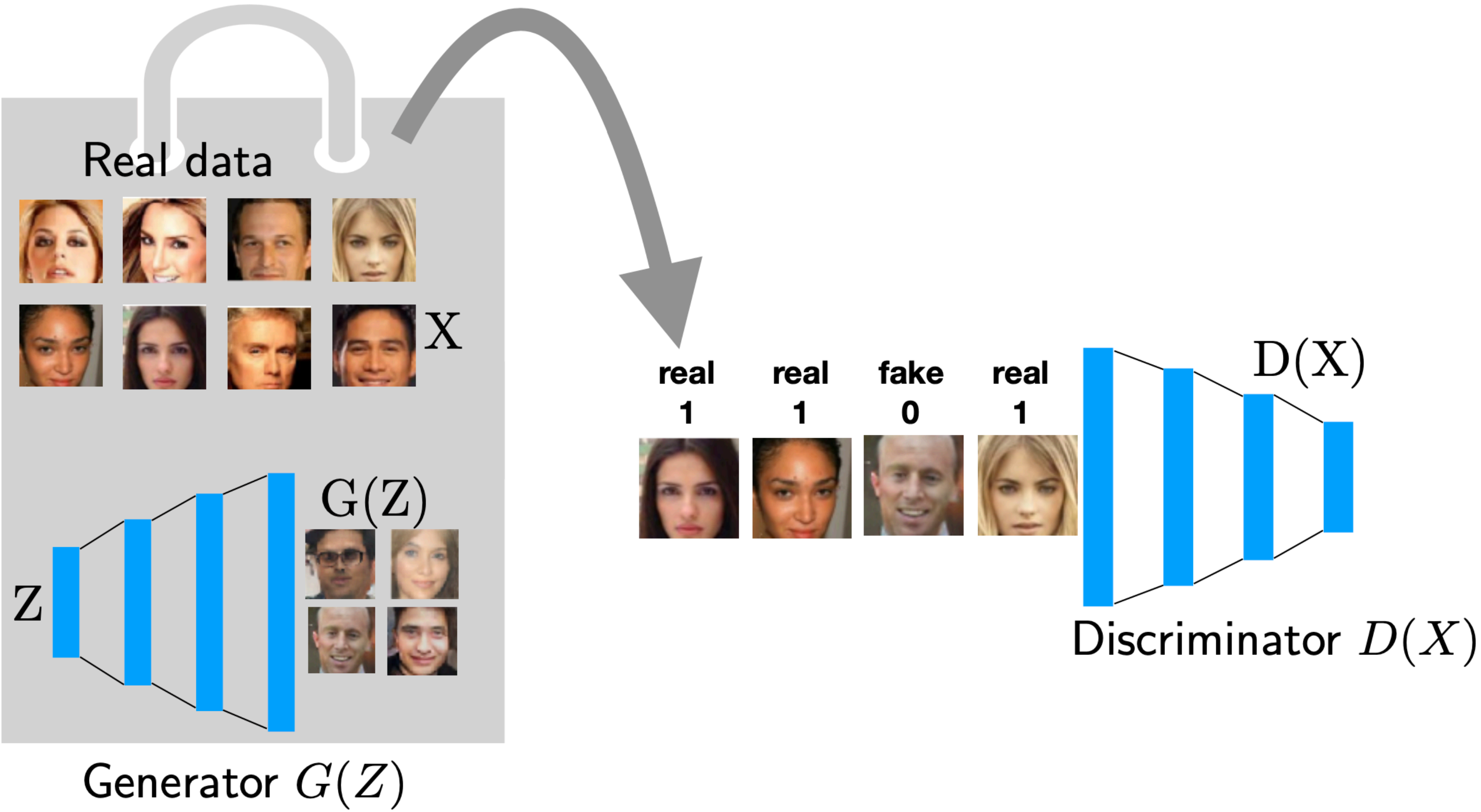
Principled approach to mode collapse

- Lack of diversity is easier to detect if we see multiple samples
- Consider MNIST hand-written digits
  - If we have a generator that generates 1,3,5,7 perfectly, it is hard to tell from a single sample that mode collapse has happened
  - But easier to tell from a collection of, say, 5 samples all from wither training data or all from generated data



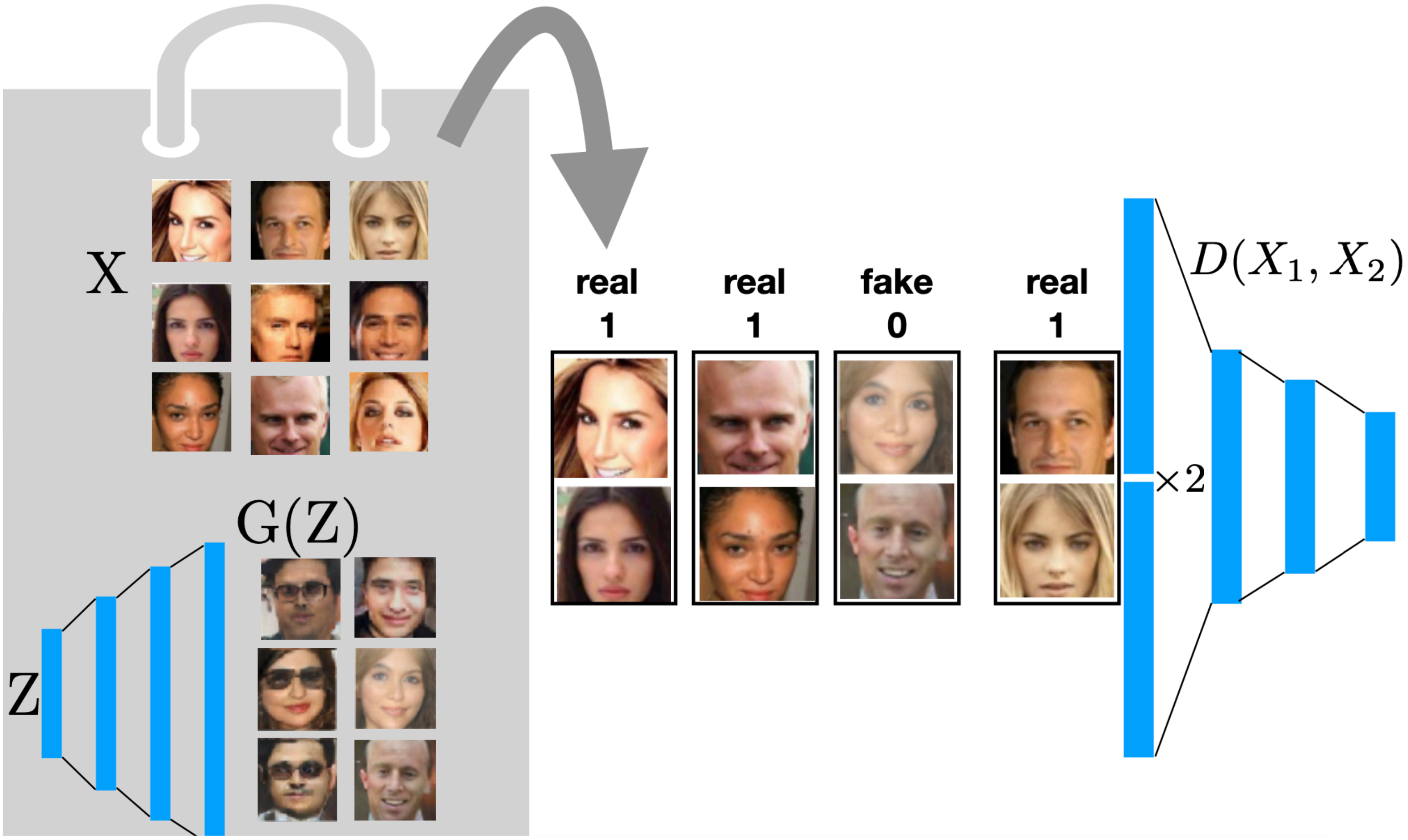
Principled approach to mode collapse

- Turning this intuition into a training algorithm:

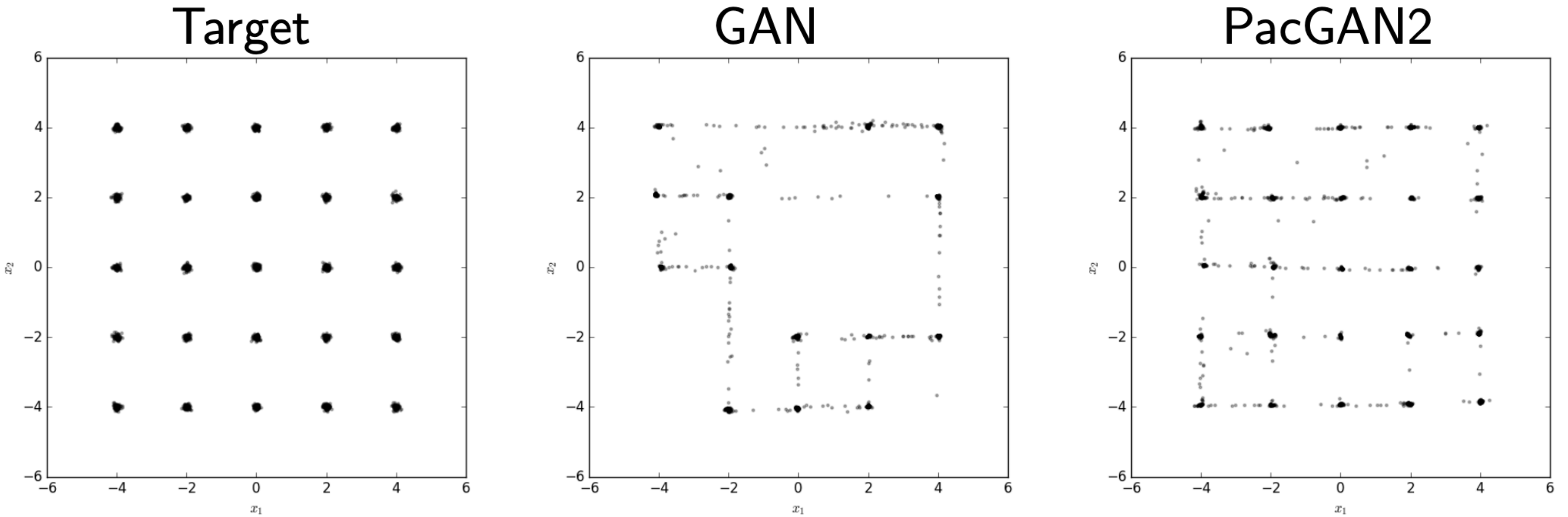


Principled approach to mode collapse: PacGAN, 2018

- Turning this intuition into a training algorithm:



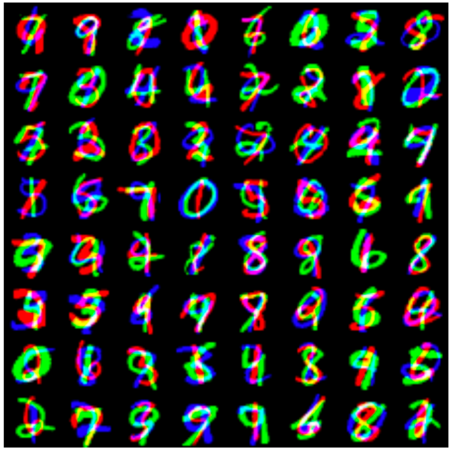
Principled approach to mode collapse



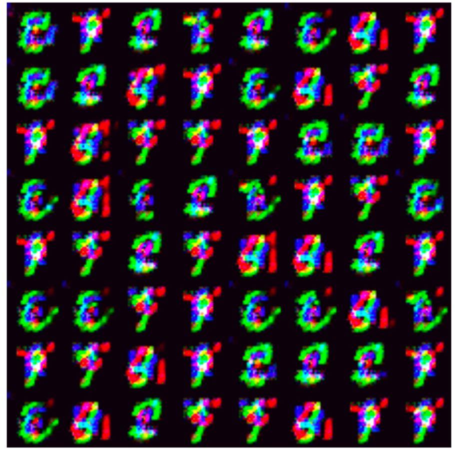
	Modes (Max 25)
GAN	17.3
PacGAN2	23.8
PacGAN3	24.6
PacGAN4	24.8

Principled approach to mode collapse

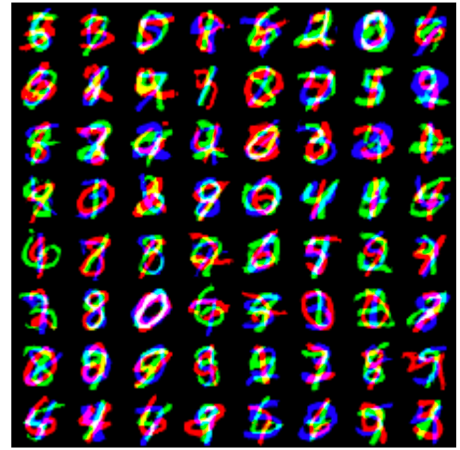
Real data



DCGAN



PacDCGAN2

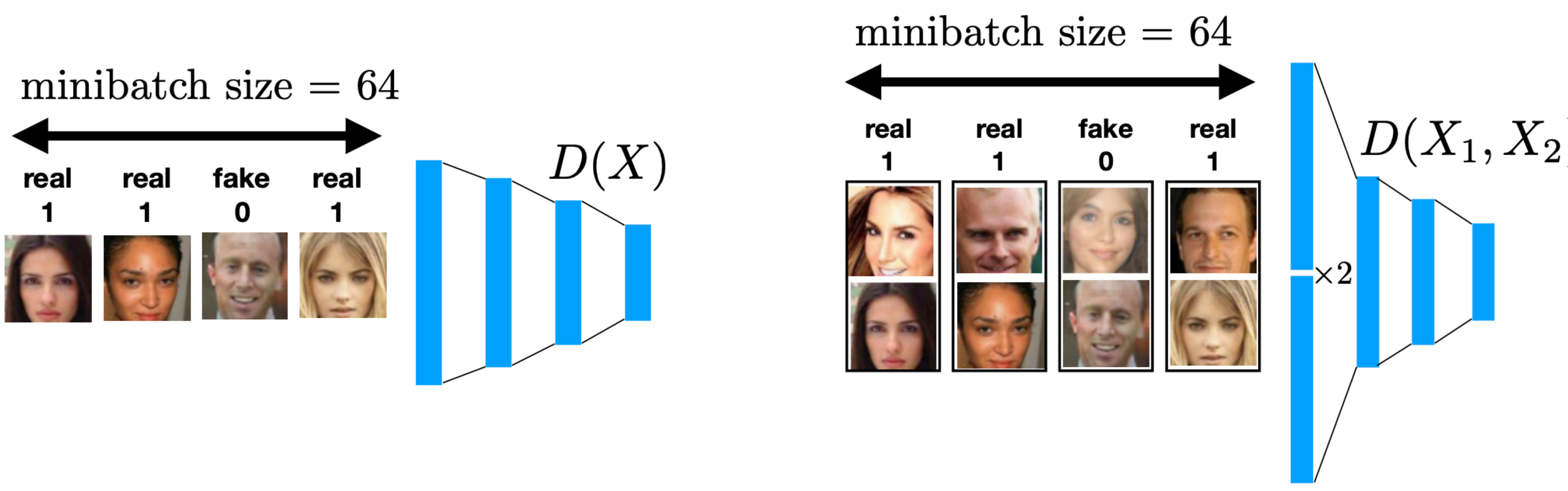


	Modes (Max 1000)
DCGAN	99.0
ALI	16.0
Unrolled GAN	48.7
VEEGAN	150.0
PacDCGAN2	1000.0
PacDCGAN3	1000.0
PacDCGAN4	1000.0

Principled approach to mode collapse

- Could PacGAN be cheating, as it is a larger discriminator network?

1. Discriminator size

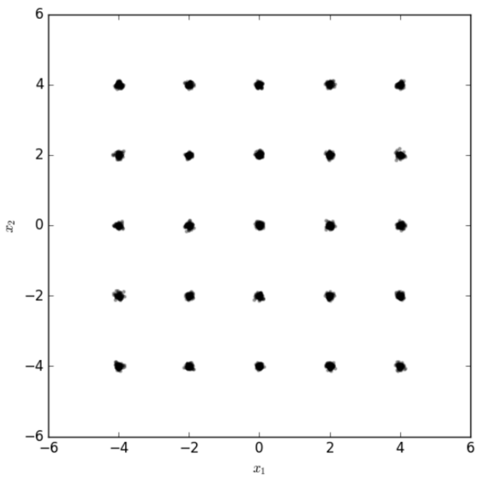


GAN PacGAN2

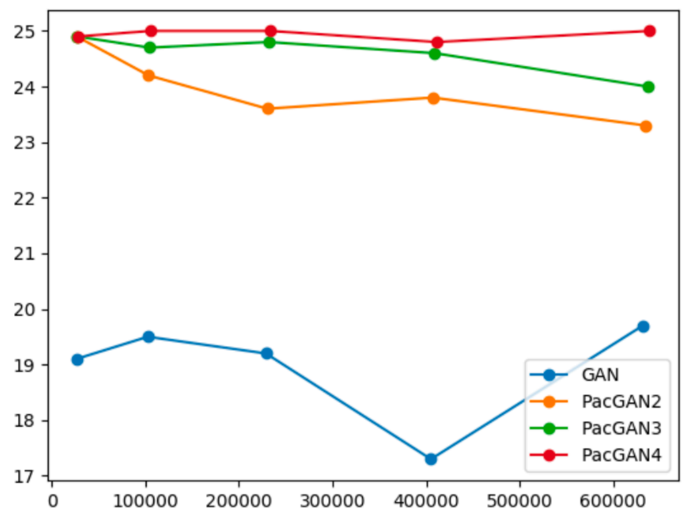
Principled approach to mode collapse

- Could PacGAN be cheating, as it is a larger discriminator network?

1. Discriminator size



\# modes captured

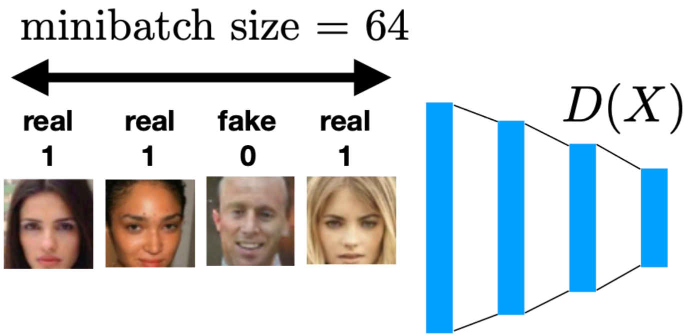


\# of parameters in $D(\cdot)$

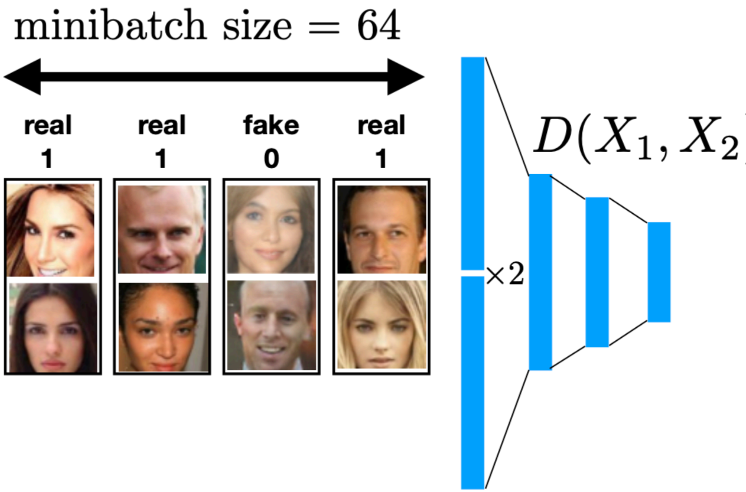
Principled approach to mode collapse

- Could PacGAN be cheating, as it uses more samples at each mini-batch?

1. Discriminator size



GAN

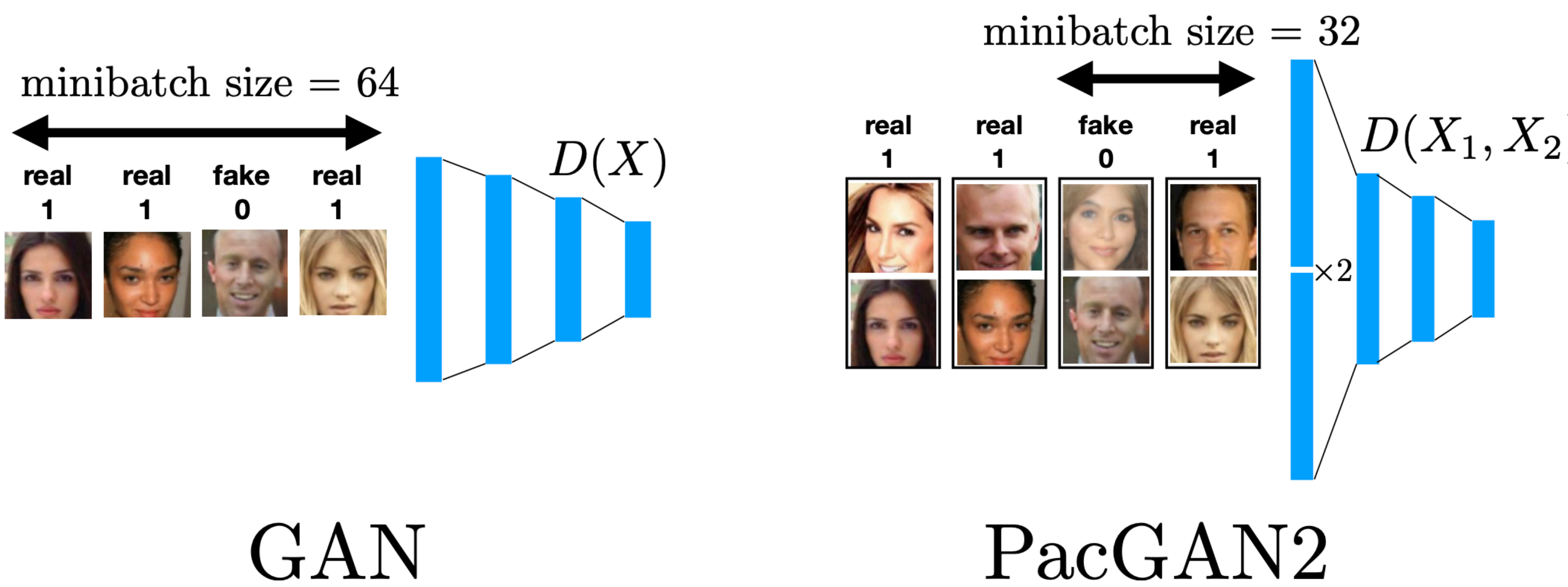


PacGAN2

Principled approach to mode collapse

- Could PacGAN be cheating, as it uses more samples at each mini-batch?

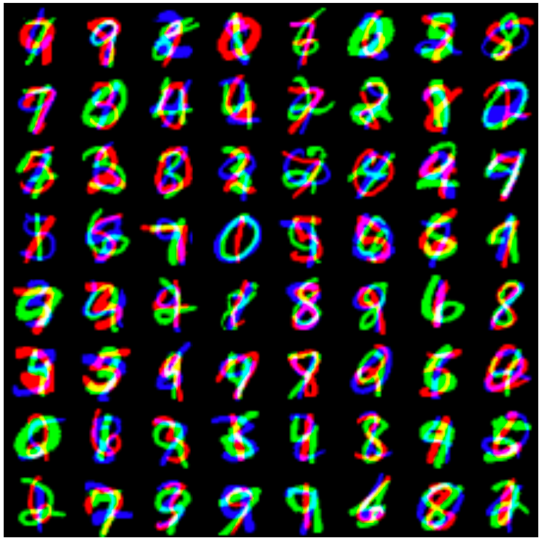
2. Minibatch size



Principled approach to mode collapse

- Could PacGAN be cheating, as it uses more samples at each mini-batch?

2. Minibatch size



	Modes
DCGAN	99.0
PacDCGAN2	1000.0

Theoretical intuition behind PacGAN

- Typical Gan training loss is

$$\min_{w} \max_{\theta} \sum_{x_i \sim P(\cdot)} \log D_\theta(x_i) + \sum_{z_i \sim N(0,\mathbf{I})} \log(1 - D_\theta(G_W(z_i)))$$

- We will consider

$$\min_{w} \max_{\theta} \sum_{x_i \sim P(\cdot)} D_\theta(x_i) + \sum_{z_i \sim N(0,\mathbf{I})} (1 - D_\theta(G_W(z_i)))$$

$$\text{subject to} \quad |D_\theta(x)| \leq 1 \;, \qquad \text{for all } x$$

Theoretical intuition behind PacGAN

- We will consider

$$\min_{w} \max_{\theta} \sum_{x_i \sim P(\cdot)} D_\theta(x_i) + \sum_{z_i \sim N(0,\mathbf{I})} (1 - D_\theta(G_W(z_i)))$$

$$\text{subject to} \quad |D_\theta(x)| \leq 1 \,, \qquad \text{for all } x$$

- this is a finite sample approximation of the following expectation

$$\min_{w} \max_{\theta} \mathbb{E}_{x \sim P(\cdot)}\left[D_\theta(x) \right] + \mathbb{E}_{z \sim N(0,\mathbf{I})}\left[1 - D_\theta(G_W(z)) \right]$$

- let $Q(\,\cdot\,)$ denote the distribution of the generator $G_w(z_i)$

$$\min_{Q(\cdot)} \max_{\theta} \mathbb{E}_{x \sim P(\cdot)}\left[D_\theta(x) \right] + \mathbb{E}_{x \sim Q(\cdot)}\left[1 - D_\theta(x) \right]$$

$$\text{subject to} \quad |D_\theta(x)| \leq 1 \,, \qquad \text{for all } x$$

- at this point, we can solve the maximization w.r.t. D_θ assuming it can represent any functions (for the purpose of theoretical analysis)
  - the optimal solution is

$$D_\theta(x) \;=\; \begin{cases} +1 & \text{if } P(x) \geq Q(x) \\ -1 & \text{if } P(x) < Q(x) \end{cases}$$

Theoretical intuition behind PacGAN

$$\min_{Q(\cdot)} \max_{\theta} \quad \mathbb{E}_{x\sim P(\cdot)}\big[D_\theta(x) \big] + \mathbb{E}_{x\sim Q(\cdot)}\big[1 - D_\theta(x) \big]$$

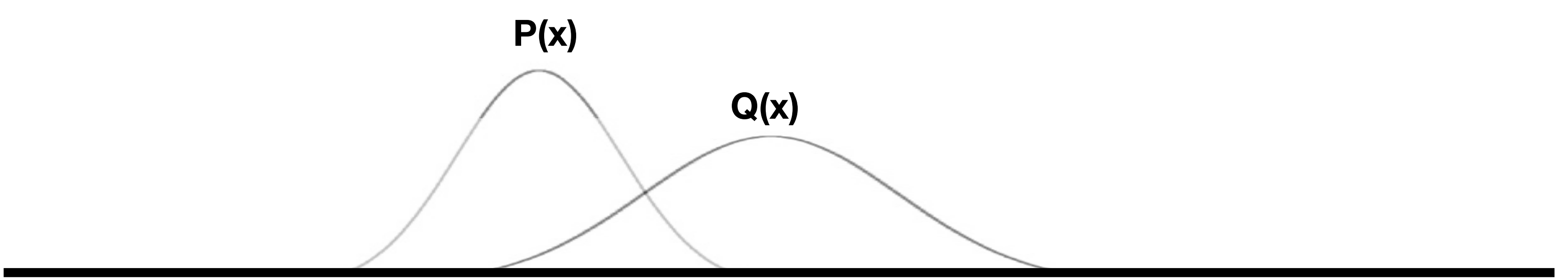
$$\text{subject to} \quad |D_\theta(x)| \le 1 , \qquad \text{for all } x$$

- at this point, we can solve the maximization w.r.t. D_θ assuming it can represent any functions (for the purpose of theoretical analysis)
  - the optimal solution is

$$D_\theta(x) \;=\; \begin{cases} +1 & \text{if } P(x) \ge Q(x) \\ -1 & \text{if } P(x) < Q(x) \end{cases}$$

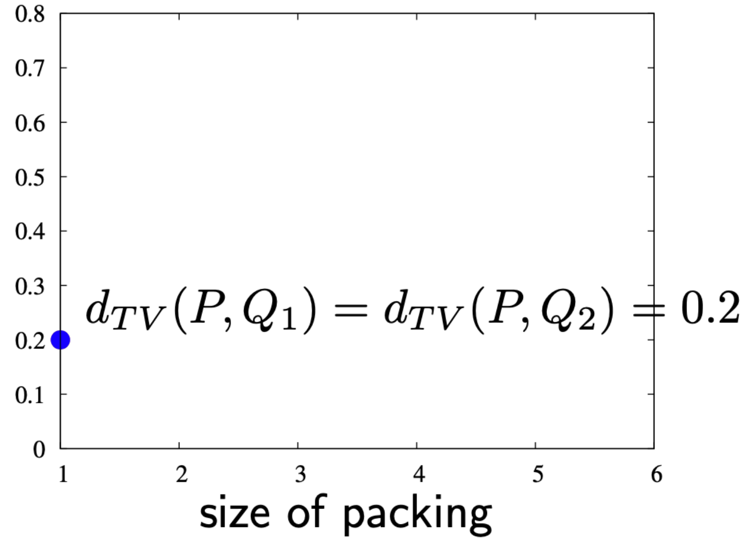
- Plugging this back in to the loss, we get

$$\min_{Q(\cdot)} \; D_{\text{TV}}(P, Q) \;=\; \mathbb{E}_{x\sim P(\cdot)}\Big[\Big| 1 - \frac{Q(x)}{P(x)} \Big| \Big]$$



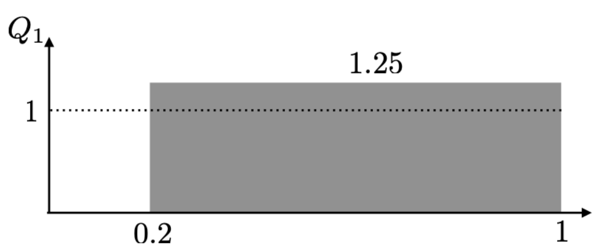
Theoretical intuition behind PacGAN

Target distribution P

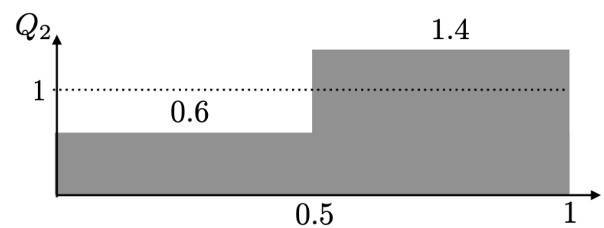


Generator Q_1
with mode collapse

Generator Q_2
without mode collapse



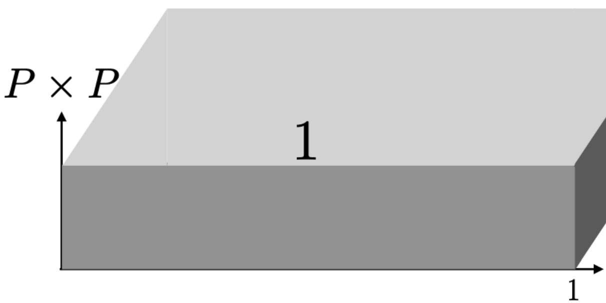
$d_{\mathrm{TV}}(P, Q_1) = 0.2$

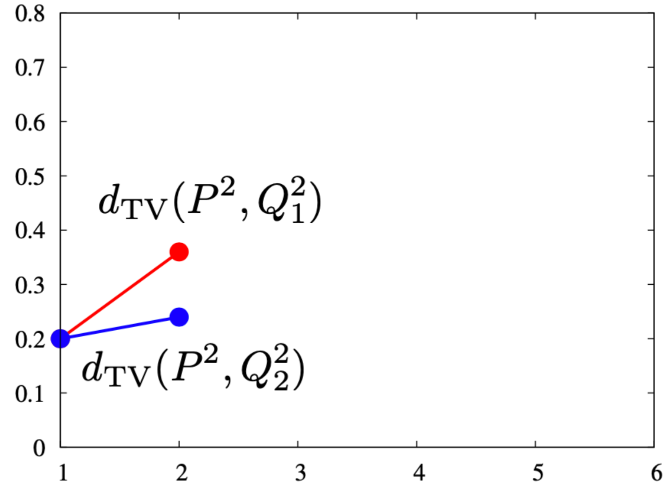


$d_{\mathrm{TV}}(P, Q_2) = 0.2$

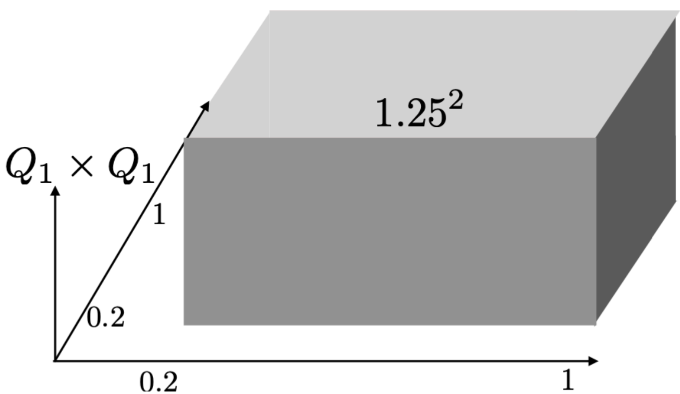
Theoretical intuition behind PacGAN

Target distribution P

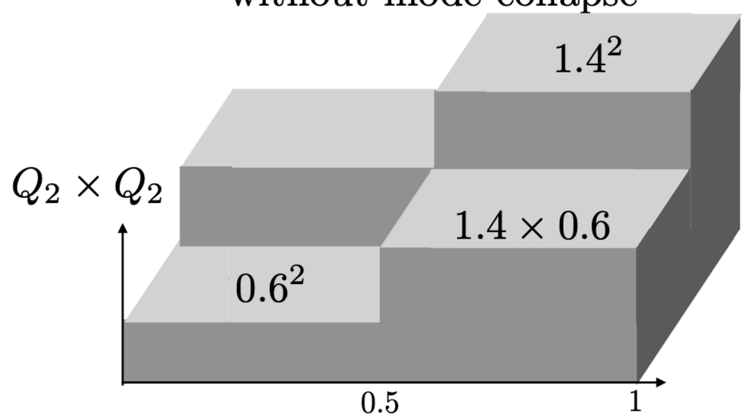




Generator Q_1
with mode collapse



Generator Q_2
without mode collapse

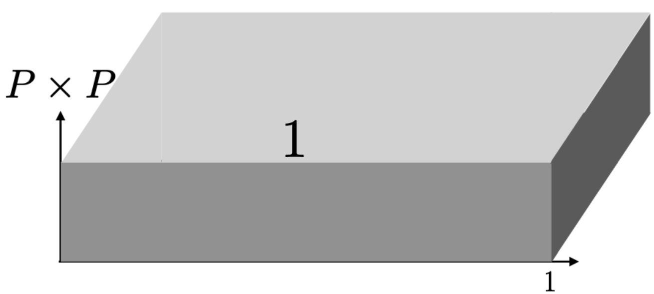


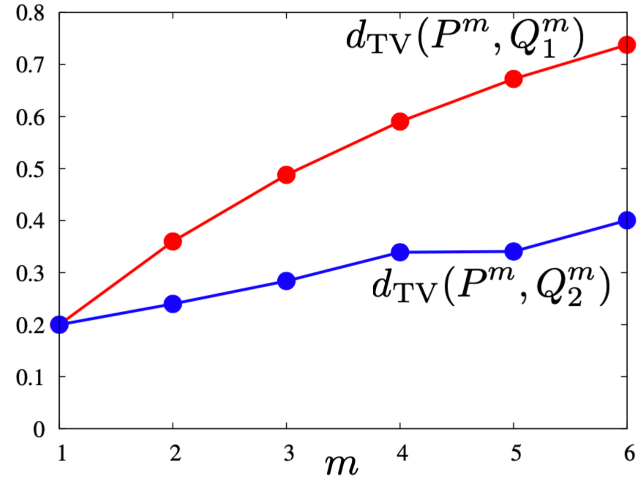
$d_{\mathrm{TV}}(P \times P, Q_1 \times Q_1) = 0.36$

$d_{\mathrm{TV}}(P \times P, Q_2 \times Q_2) = 0.24$

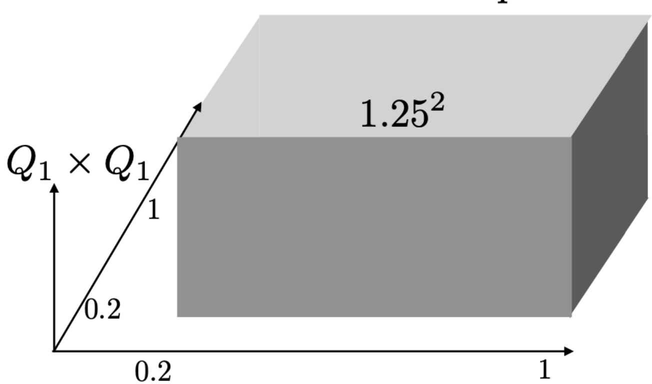
Theoretical intuition behind PacGAN

Target distribution P

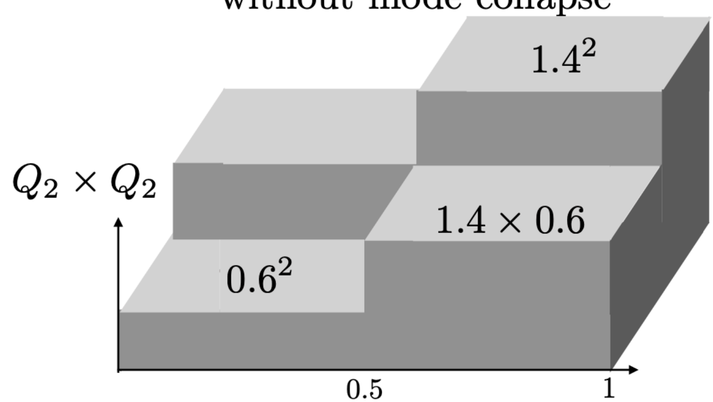




Generator Q_1
with mode collapse



Generator Q_2
without mode collapse



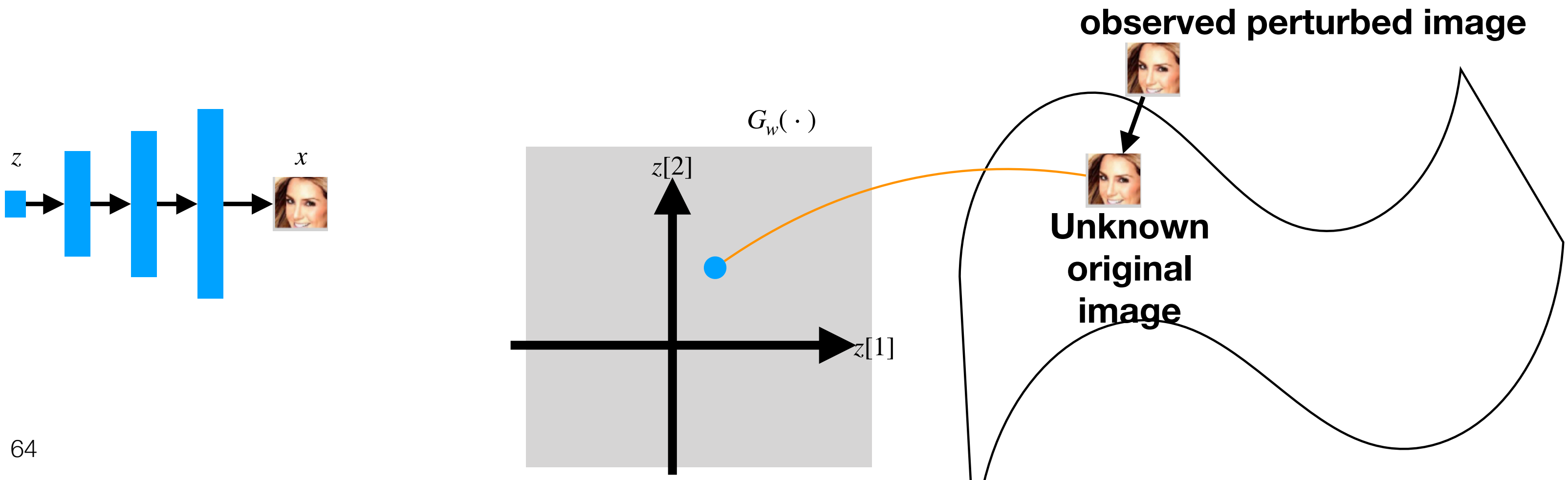
$d_{\mathrm{TV}}(P \times P, Q_1 \times Q_1) = 0.36$

$d_{\mathrm{TV}}(P \times P, Q_2 \times Q_2) = 0.24$

Deep Image prior

- in standard de-noising/inpainting with **trained** GAN
  we want to recover original image from some distortion

- if we have a GAN trained on similar class of images, then we can use the latent space and the manifold of natural images to recover the image as follows

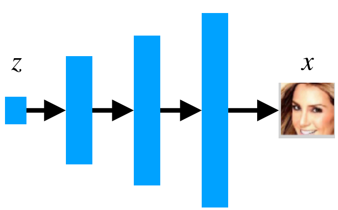


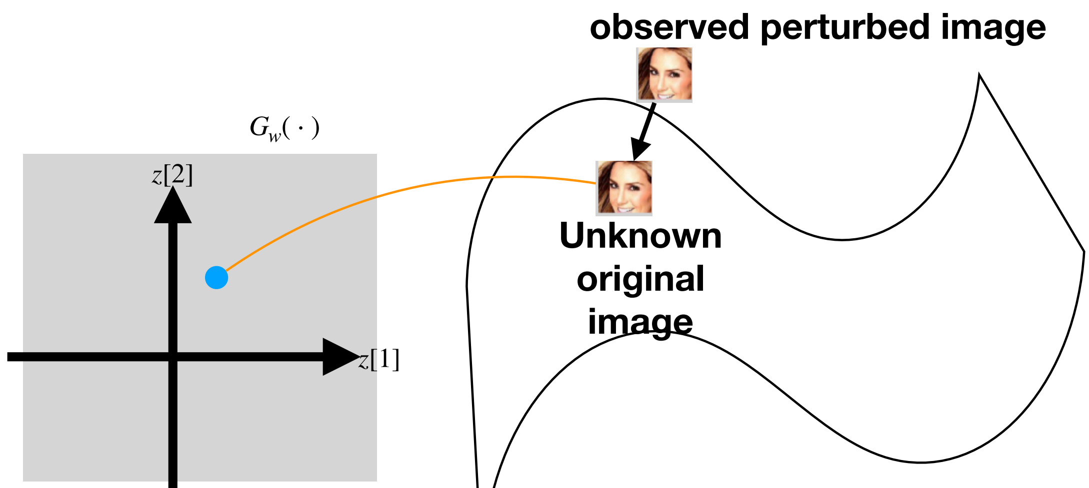
Deep Image prior

- Given a trained generator W that knows the manifold of natural images, find the latent vector z that

$$\text{minimize}_z \quad \ell\Big(G_w(z$$  $$\Big)$$

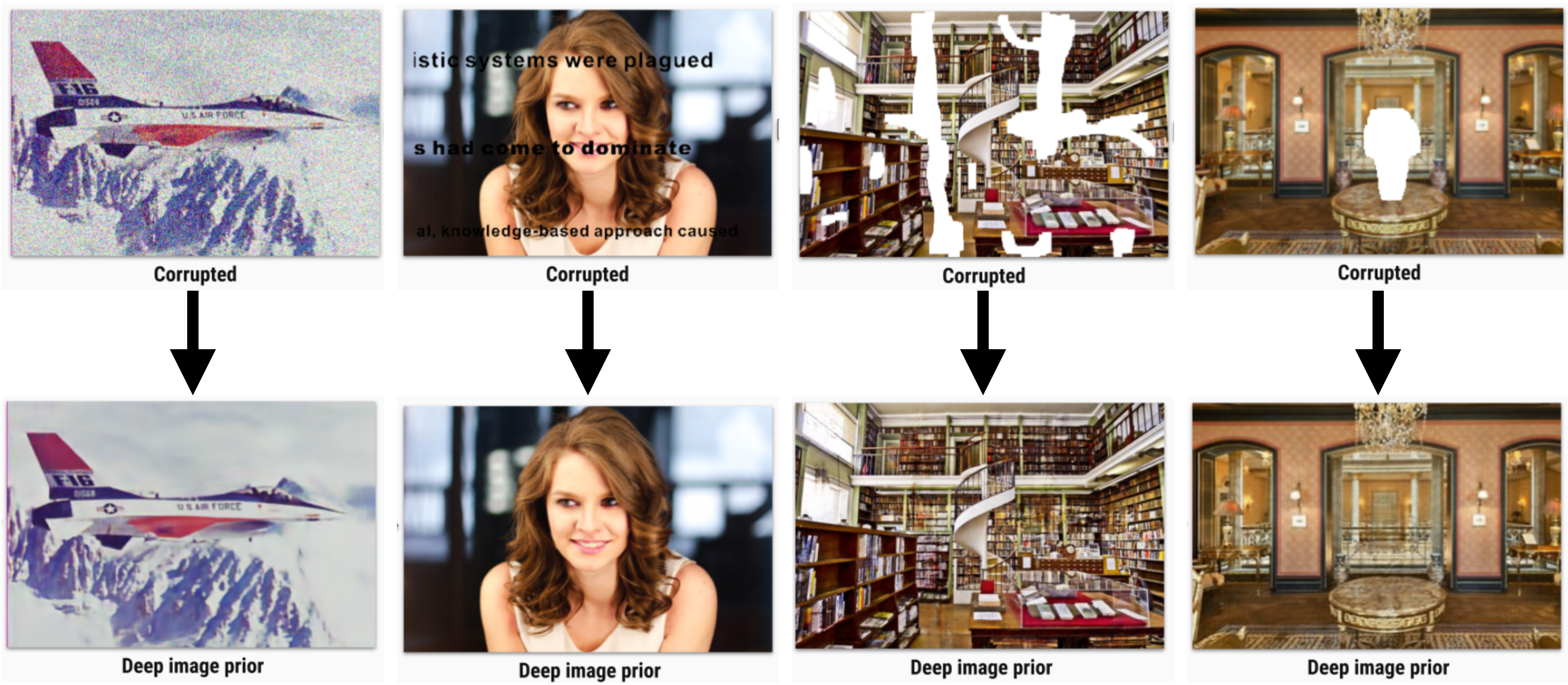
- let $G_w(z)$ be the recovered image





Deep image prior

- **deep image prior** does amazing recovery, **without training**



Deep image prior

- fix z to be something random and find w that

$$\text{minimize}_z \quad \ell\Big(C$$

$$\Big)$$

and let $G_w(z)$ be the recovered image

https://www.youtube.com/watch?v=kSLJriaOumA&feature=youtu.be

Questions?

Questions?