

# Unsupervised Learning

Shan-Hung Wu

*shwu@cs.nthu.edu.tw*

Department of Computer Science,  
National Tsing Hua University, Taiwan

Machine Learning

# Outline

- ① Unsupervised Learning
- ② Self-Supervised Learning
- ③ Autoencoders & Manifold Learning
- ④ Generative Adversarial Networks
  - The Basics
  - Challenges
  - More GANs

# Outline

## ① Unsupervised Learning

## ② Self-Supervised Learning

## ③ Autoencoders & Manifold Learning

## ④ Generative Adversarial Networks

- The Basics
- Challenges
- More GANs

# Unsupervised Learning

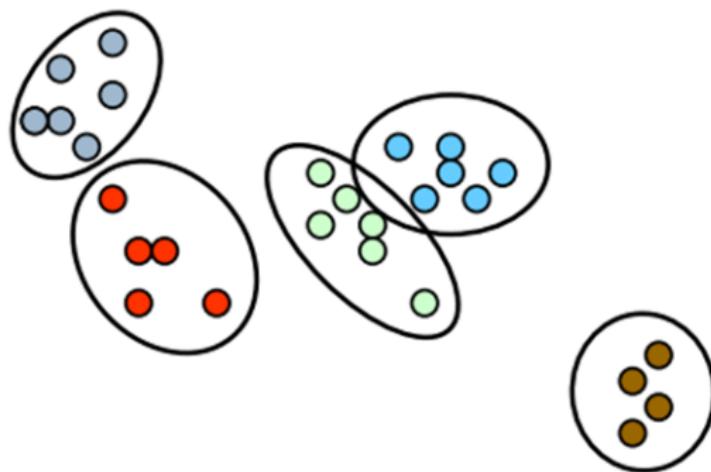
- Dataset:  $\mathbb{X} = \{\mathbf{x}^{(i)}\}_i$ , where  $\mathbf{x}^{(i)}$ 's are i.i.d. samples of  $\mathbf{x}$ 
  - No supervision (e.g., labels)

# Unsupervised Learning

- Dataset:  $\mathbb{X} = \{\mathbf{x}^{(i)}\}_i$ , where  $\mathbf{x}^{(i)}$ 's are i.i.d. samples of  $\mathbf{x}$ 
  - No supervision (e.g., labels)
- What can we learn?

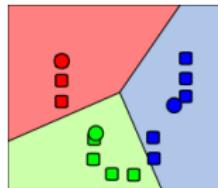
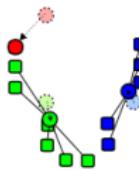
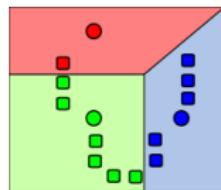
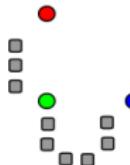
# Clustering I

- Goal: to divide  $x^{(i)}$ 's into  $K$  groups/**clusters**
  - Based on some pairwise similarity/distance measure



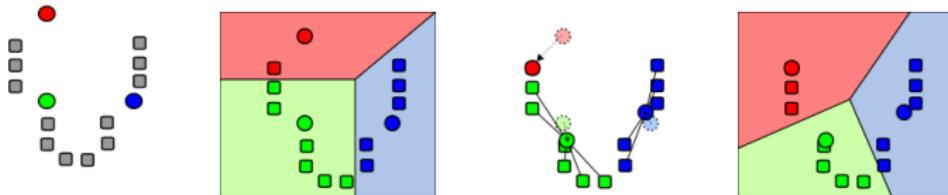
# Clustering II

- $K$ -means algorithm ( $K$  fixed):

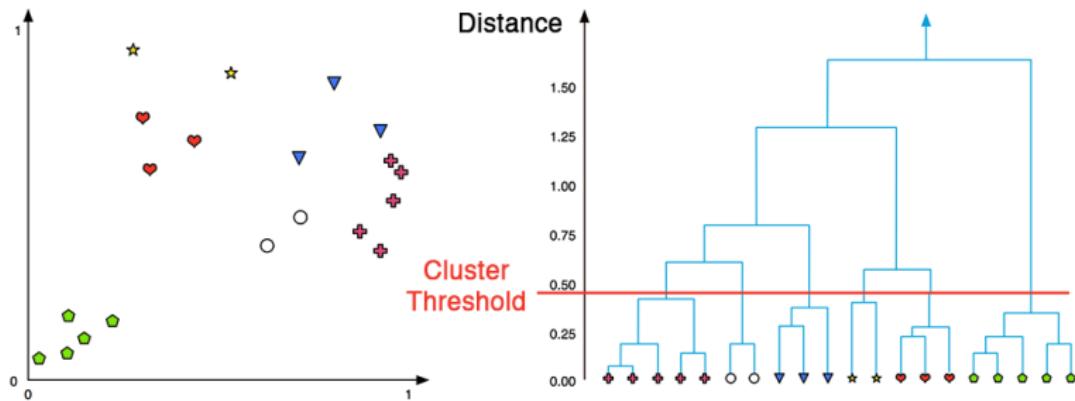


# Clustering II

- K-means algorithm ( $K$  fixed):



- Hierarchical clustering (variable  $K$ ):

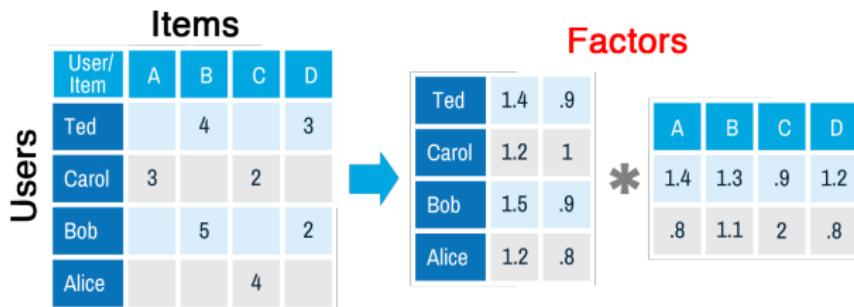


# Factorization and Recommendation

- Goal: to uncover the *factors* behind  $\mathbb{X}$

# Factorization and Recommendation

- Goal: to uncover the *factors* behind  $\mathbb{X}$
- Commonly used in the recommender systems
- Let  $X$ ,  $X_{i,:} = \mathbf{x}^{(i)}$ , be a rating matrix

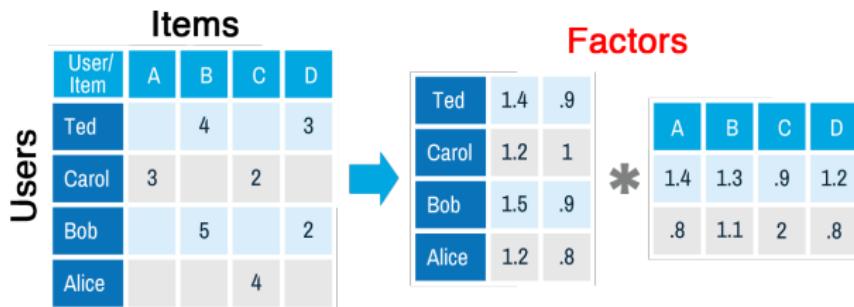


- Non-negative matrix factorization (NMF) [12, 13]:

$$\arg \min_{W \geq O, H \geq O} \|X - WH\|_F$$

# Factorization and Recommendation

- Goal: to uncover the *factors* behind  $\mathbb{X}$
- Commonly used in the recommender systems
- Let  $X$ ,  $X_{i,:} = \mathbf{x}^{(i)}$ , be a rating matrix



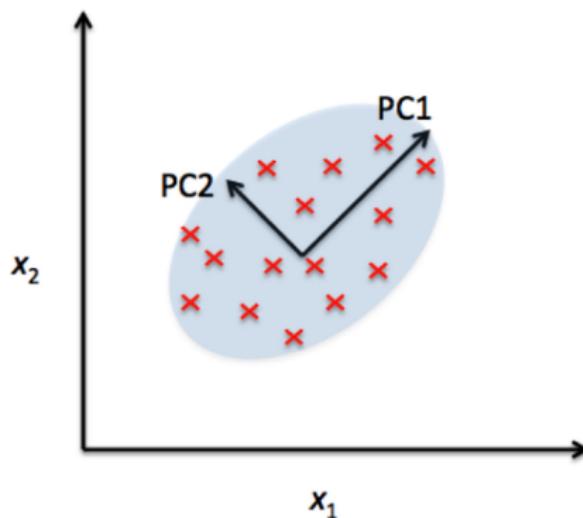
- Non-negative matrix factorization (NMF) [12, 13]:

$$\arg \min_{W \geq O, H \geq O} \|X - WH\|_F$$

- $X^* = W^*H^*$  a dense matrix and can be used to predict user interests

# Dimension Reduction

- Goal: to learn a low dimensional representation  $\mathbf{z}$  of  $\mathbf{x}$ 
  - E.g., PCA



# Self-Supervised Learning

- Goal: to learn a model that is able to “fill in the blanks”
  - Also called “predictive learning”

 $x^{(i)}$  $y^{(i)}$

# Self-Supervised Learning

- Goal: to learn a model that is able to “fill in the blanks”
  - Also called “predictive learning”
- Links unsupervised tasks with supervised models

 $x^{(i)}$  $y^{(i)}$

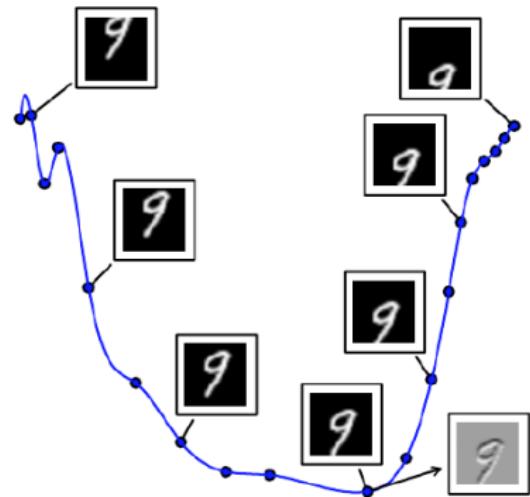
# Self-Supervised Learning

- Goal: to learn a model that is able to “fill in the blanks”
  - Also called “predictive learning”
- Links unsupervised tasks with supervised models
- For supervised tasks: more easy data collection
- For unsupervised tasks: better representations of  $\mathbf{x}$  and/or  $\mathbf{y}$  thanks to DL models

 $\mathbf{x}^{(i)}$  $\mathbf{y}^{(i)}$

# Manifold Learning

- Goal: to learn the underlying manifold of  $\mathbf{x}$ 
  - E.g., given a point  $x^{(i)}$ , output the tangent vector of  $x^{(i)}$

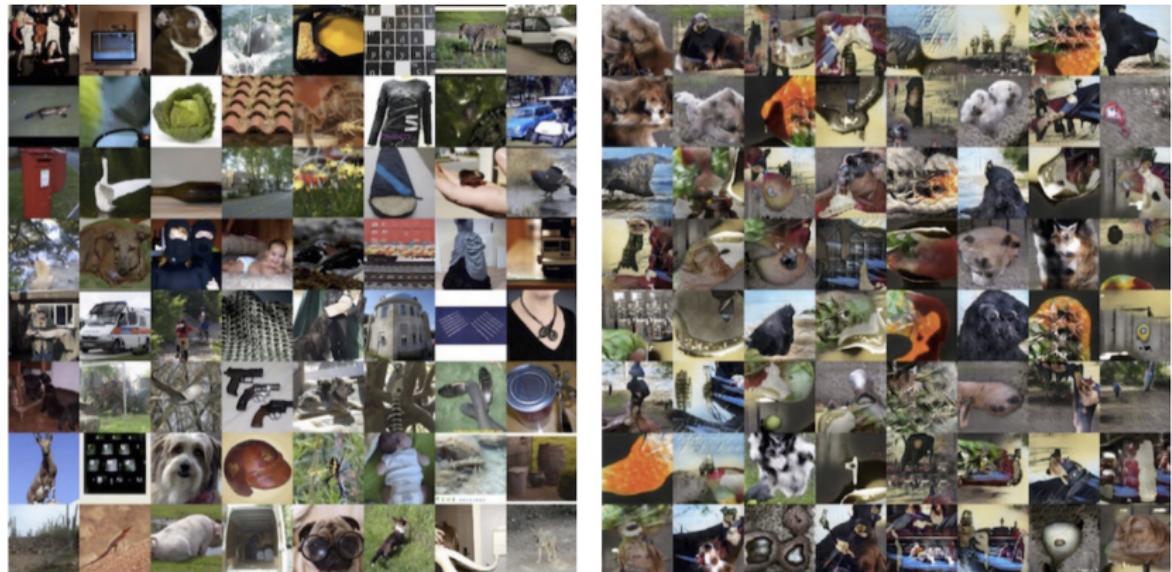


# Data Synthesis/Generation I

- Goal: to generate new samples of  $\mathbf{x}$

# Data Synthesis/Generation I

- Goal: to generate new samples of  $\mathbf{x}$
- *Generative adversarial networks* (GANs)



# Data Synthesis/Generation II

- Conditional GANs, e.g., text to image synthesis

*“This bird is completely red with black wings and pointy beak.”*



# Outline

① Unsupervised Learning

② Self-Supervised Learning

③ Autoencoders & Manifold Learning

④ Generative Adversarial Networks

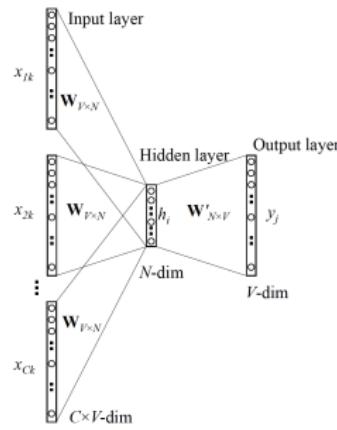
- The Basics
- Challenges
- More GANs

# Self-Supervised Learning

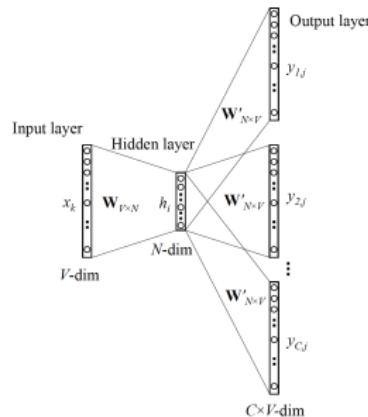
- Goal: to learn a model for blank filling

# Self-Supervised Learning

- Goal: to learn a model for blank filling
- E.g., word2vec [17, 16]: “... *the cat sat on...*”



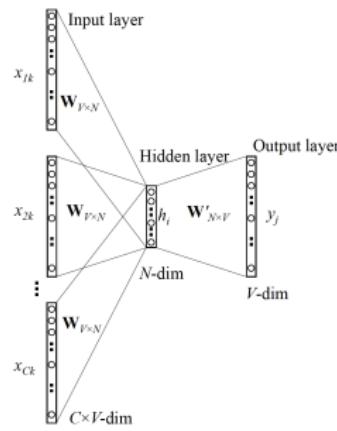
CBOW



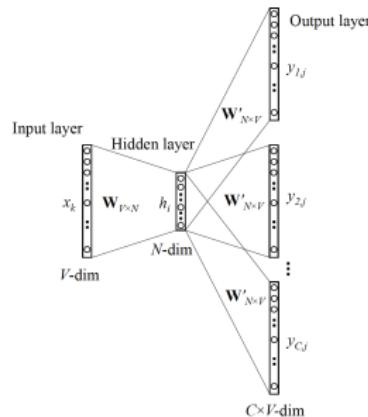
Skip Gram

# Self-Supervised Learning

- Goal: to learn a model for blank filling
- E.g., word2vec [17, 16]: "... *the cat sat on...*"



CBOW



Skip Gram

- Latent representation  $\mathbf{h}$  encodes the ***semantics*** of a word
  - No need for synonym dictionary; big data tell that already

# Doc2Vec

- How to encode a document?

# Doc2Vec

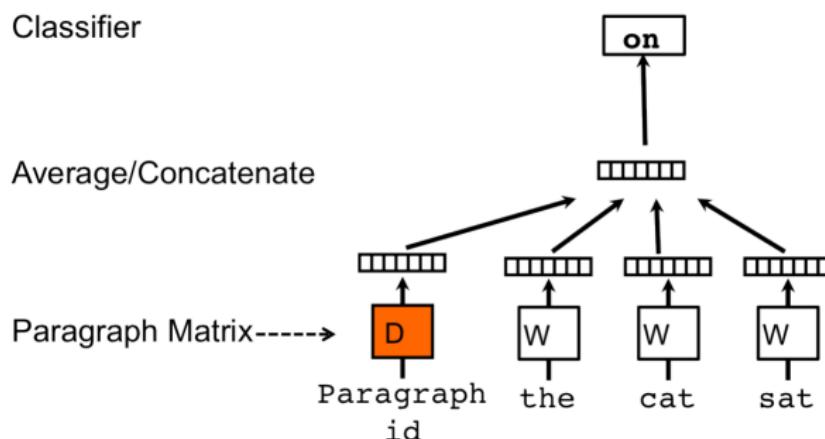
- How to encode a document?
  - Bag of words (TF-IDF), average word2vec, etc.

# Doc2Vec

- How to encode a document?
  - Bag of words (TF-IDF), average word2vec, etc.
- Do **not** capture the semantics due to sentence/paragraph/doc structure
  - “*John likes Mary*”  $\neq$  “*Mary likes John*”
- Self-supervised learning for docs?

# Doc2Vec

- How to encode a document?
  - Bag of words (TF-IDF), average word2vec, etc.
- Do **not** capture the semantics due to sentence/paragraph/doc structure
  - “John likes Mary”  $\neq$  “Mary likes John”
- Self-supervised learning for docs?
- Doc2vec [10]: to capture the **context** not explained by words



# Filling Images

- How?

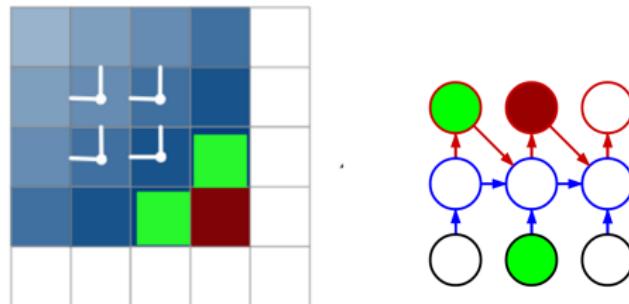


# Filling Images

- How?



- PixelRNN [25], PixelCNN [24]



# More

- Predicting the future by watching unlabeled videos [14, 7, 27]:



# More

- Predicting the future by watching unlabeled videos [14, 7, 27]:



- Better representations/predictions without the need for labels

# Outline

① Unsupervised Learning

② Self-Supervised Learning

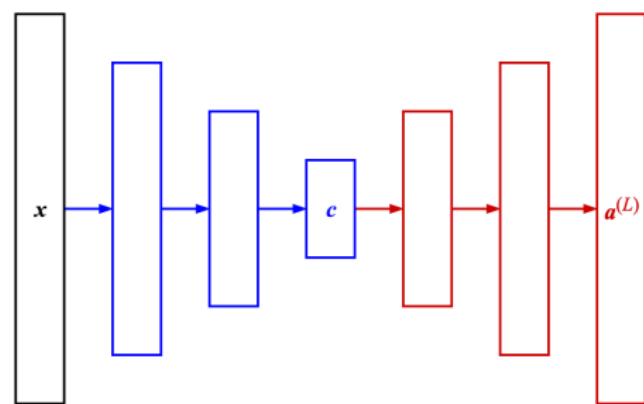
③ Autoencoders & Manifold Learning

④ Generative Adversarial Networks

- The Basics
- Challenges
- More GANs

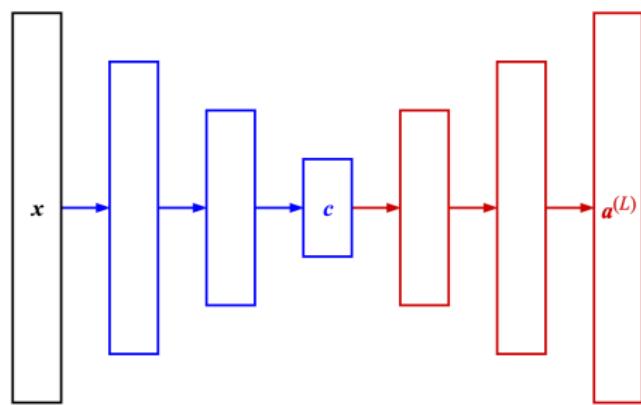
# Autoencoders I

- **Encoder**: to learn a low dimensional representation  $c$  (called **code**) of input  $x$
- **Decoder**: to reconstruct  $x$  from  $c$



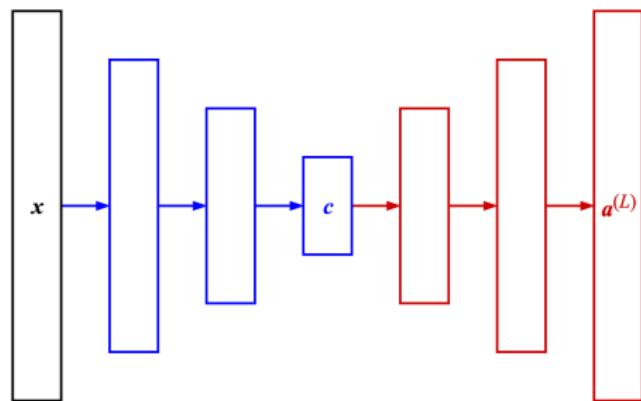
# Autoencoders I

- **Encoder**: to learn a low dimensional representation  $\mathbf{c}$  (called **code**) of input  $\mathbf{x}$
- **Decoder**: to reconstruct  $\mathbf{x}$  from  $\mathbf{c}$
- Cost function:  $\arg \min_{\Theta} -\log P(\mathbb{X} | \Theta) = \arg \min_{\Theta} -\sum_n \log P(x^{(n)} | \Theta)$



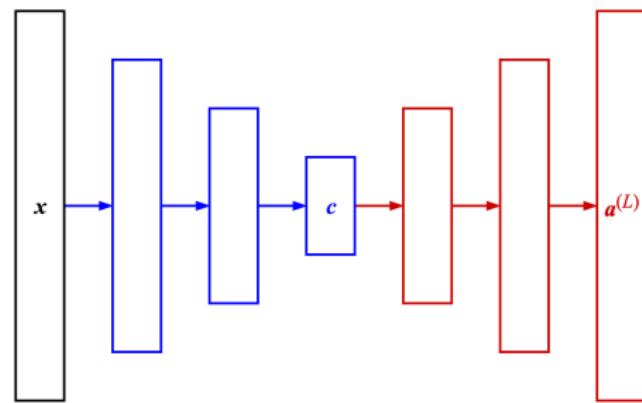
# Autoencoders I

- **Encoder**: to learn a low dimensional representation  $\mathbf{c}$  (called **code**) of input  $\mathbf{x}$
- **Decoder**: to reconstruct  $\mathbf{x}$  from  $\mathbf{c}$
- Cost function:  $\arg \min_{\Theta} -\log P(\mathbb{X} | \Theta) = \arg \min_{\Theta} -\sum_n \log P(x^{(n)} | \Theta)$
- Sigmoid output units  $a_j^{(L)} = \hat{\rho}_j$  for  $x_j \sim \text{Bernoulli}(\rho_j)$ 
  - $P(x_j^{(n)} | \Theta) = (a_j^{(L)})^{x_j^{(n)}} (1 - a_j^{(L)})^{(1-x_j^{(n)})}$



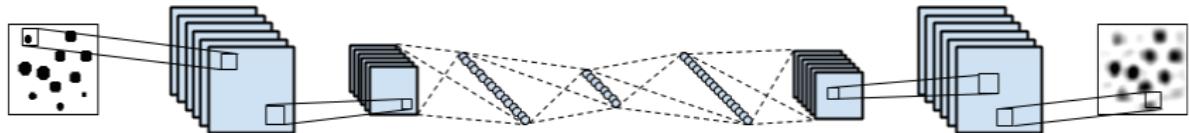
# Autoencoders I

- **Encoder**: to learn a low dimensional representation  $\mathbf{c}$  (called **code**) of input  $\mathbf{x}$
- **Decoder**: to reconstruct  $\mathbf{x}$  from  $\mathbf{c}$
- Cost function:  $\arg \min_{\Theta} -\log P(\mathbb{X} | \Theta) = \arg \min_{\Theta} -\sum_n \log P(\mathbf{x}^{(n)} | \Theta)$
- Sigmoid output units  $a_j^{(L)} = \hat{\rho}_j$  for  $x_j \sim \text{Bernoulli}(\rho_j)$ 
  - $P(x_j^{(n)} | \Theta) = (a_j^{(L)})^{x_j^{(n)}} (1 - a_j^{(L)})^{(1-x_j^{(n)})}$
- Linear output units  $\mathbf{a}^{(L)} = \mathbf{z}^{(L)} = \hat{\mu}$  for  $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ 
  - $-\log P(\mathbf{x}^{(n)} | \Theta) = \|\mathbf{x}^{(n)} - \mathbf{a}^{(L)}\|^2$



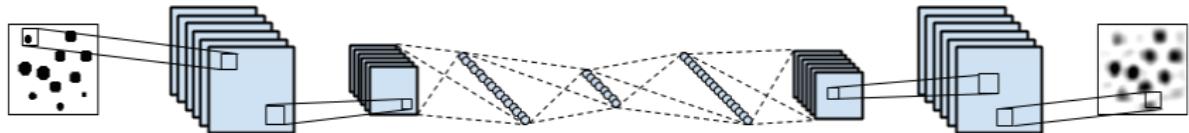
# Convolutional Autoencoders

- Convolution + deconvolution layers:



# Convolutional Autoencoders

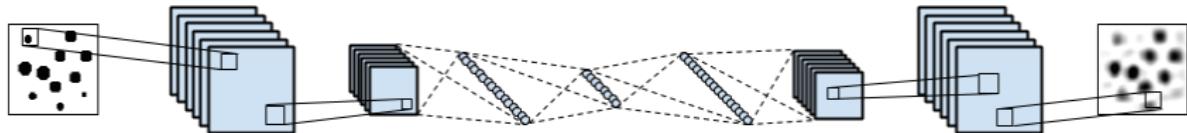
- Convolution + deconvolution layers:



- Decoder is a simplified DeconvNet [28] trained from scratch:

# Convolutional Autoencoders

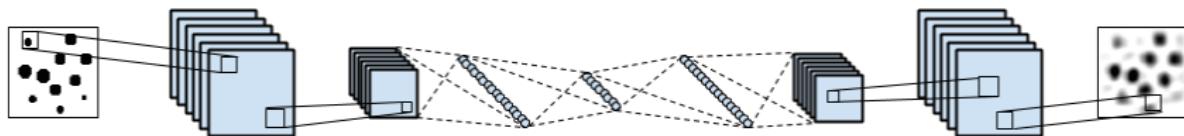
- Convolution + deconvolution layers:



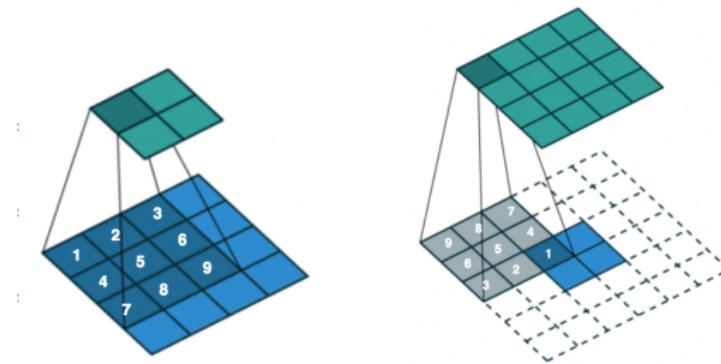
- Decoder is a simplified DeconvNet [28] trained from scratch:
  - Uppooling → upsampling (no need to remember max positions)

# Convolutional Autoencoders

- Convolution + deconvolution layers:



- Decoder is a simplified DeconvNet [28] trained from scratch:
  - Uppooling → upsampling (no need to remember max positions)
  - Deconvolution → convolution



# Codes & Reconstructed x

- A 32-bit code can roughly represents a  $32 \times 32$  (1024 dimensional) MNIST image

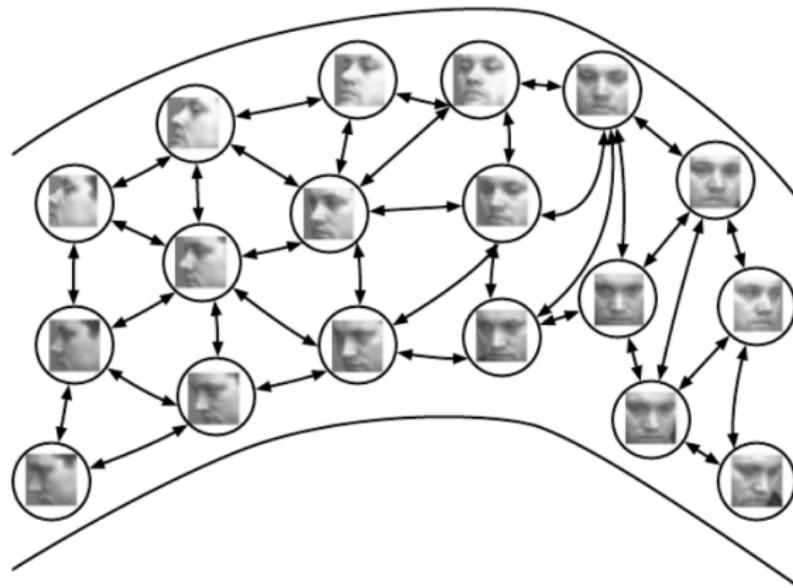


# Manifolds I

- In many applications, data concentrate around one or more low-dimensional *manifolds*

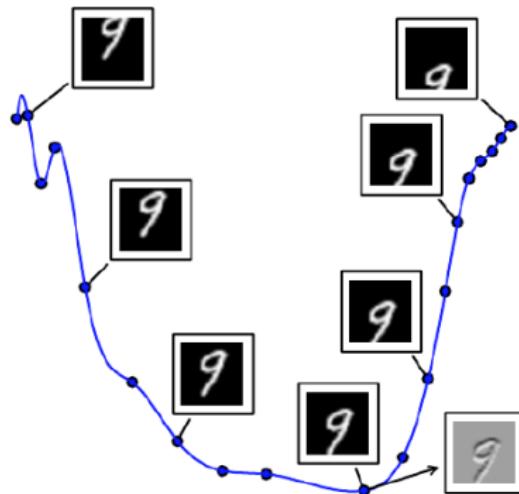
# Manifolds I

- In many applications, data concentrate around one or more low-dimensional **manifolds**
- A manifold is a topological space that are **linear locally**



# Manifolds II

- For each point  $x$  on a manifold, we have its **tangent space** spanned by **tangent vectors**
  - Local directions specify how one can change  $x$  infinitesimally while staying on the manifold



# Learning Manifolds I

- How to make  $\mathbf{c}$  produced by autoencoders denote a *coordinate* of a dimensional manifold?

# Learning Manifolds I

- How to make  $\mathbf{c}$  produced by autoencoders denote a *coordinate* of a dimensional manifold?
- Contractive autoencoder [20]: regularizes the code  $\mathbf{c}$  such that it is invariant to local changes of  $\mathbf{x}$ :

$$\Omega(\mathbf{c}) = \sum_n \left\| \frac{\partial \mathbf{c}^{(n)}}{\partial \mathbf{x}^{(n)}} \right\|_F^2$$

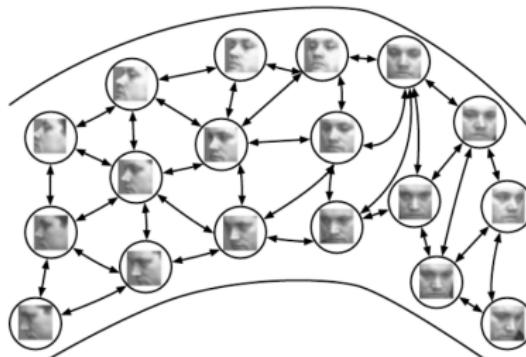
- $\partial \mathbf{c}^{(n)} / \partial \mathbf{x}^{(n)}$  is a Jacobian matrix

# Learning Manifolds I

- How to make  $\mathbf{c}$  produced by autoencoders denote a *coordinate* of a dimensional manifold?
- Contractive autoencoder [20]: regularizes the code  $\mathbf{c}$  such that it is invariant to local changes of  $\mathbf{x}$ :

$$\Omega(\mathbf{c}) = \sum_n \left\| \frac{\partial \mathbf{c}^{(n)}}{\partial \mathbf{x}^{(n)}} \right\|_F^2$$

- $\partial \mathbf{c}^{(n)} / \partial \mathbf{x}^{(n)}$  is a Jacobian matrix
- Hence,  $\mathbf{c}$  represents only the variations needed to reconstruct  $\mathbf{x}$ 
  - I.e.,  $\mathbf{c}$  changes most along tangent vectors



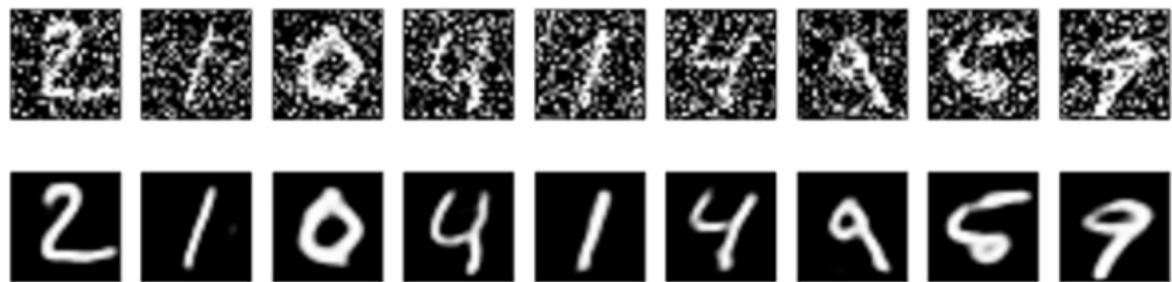
# Learning Manifolds II

- In practice, it is easier to train a denoising autoencoder [26]:



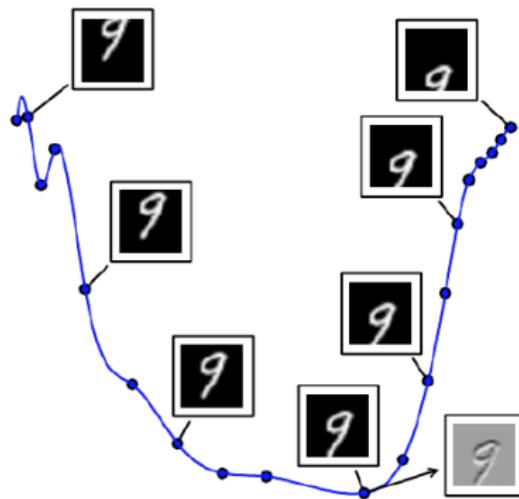
# Learning Manifolds II

- In practice, it is easier to train a denoising autoencoder [26]:
  - Encoder: to encode  $\mathbf{x}$  **with** random noises
  - Decoder: to reconstruct  $\mathbf{x}$  **without** noises



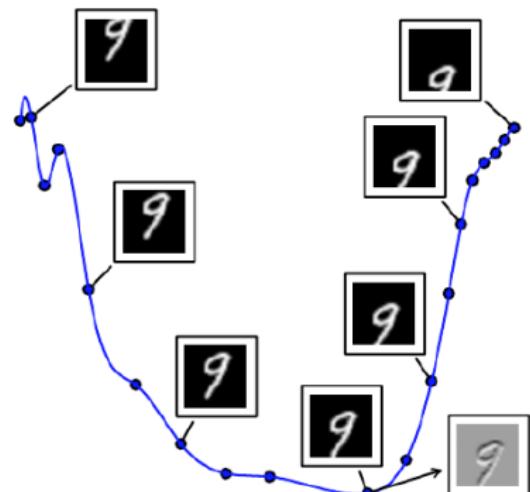
# Getting Tangent Vectors I

- The code  $c$  represents a coordinate on a low dimensional manifold
  - E.g., the blue line
- How to get the tangent vectors of a given  $c$ ?



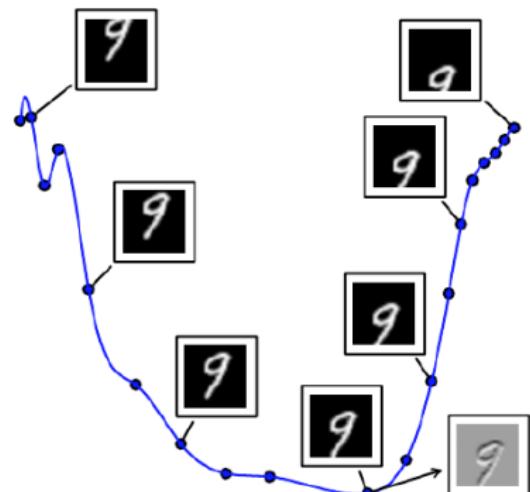
# Getting Tangent Vectors II

- Recall: directions in the input space that *changes c most* should be tangent vectors



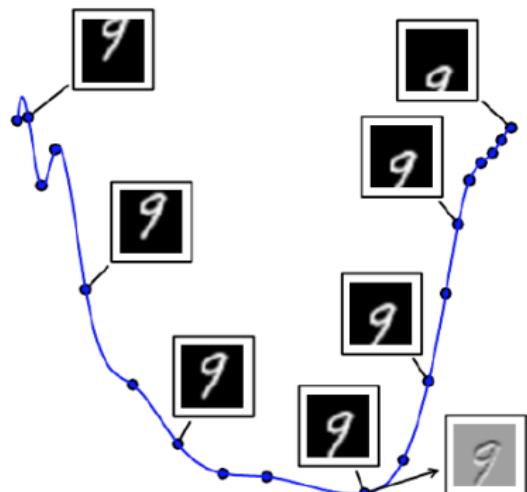
# Getting Tangent Vectors II

- Recall: directions in the input space that *changes  $c$  most* should be tangent vectors
- Given a point  $x$ , let  $c$  be the code of  $x$  and  $J(x) = \frac{\partial c}{\partial x}$  be the Jacobian matrix of  $c$  at  $x$



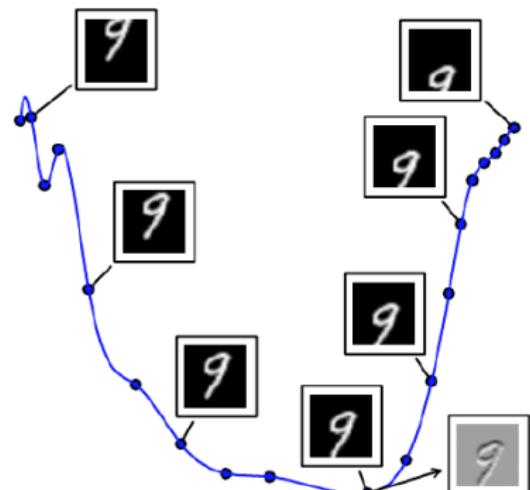
# Getting Tangent Vectors II

- Recall: directions in the input space that *changes  $c$  most* should be tangent vectors
- Given a point  $x$ , let  $c$  be the code of  $x$  and  $J(x) = \frac{\partial c}{\partial x}$  be the Jacobian matrix of  $c$  at  $x$ 
  - $J(x)$  summarizes how  $c$  changes in terms of  $x$



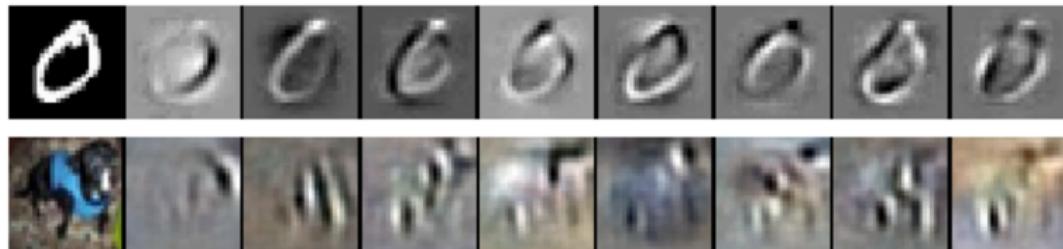
# Getting Tangent Vectors II

- Recall: directions in the input space that *changes  $c$  most* should be tangent vectors
  - Given a point  $x$ , let  $c$  be the code of  $x$  and  $J(x) = \frac{\partial c}{\partial x}$  be the Jacobian matrix of  $c$  at  $x$ 
    - $J(x)$  summarizes how  $c$  changes in terms of  $x$
- Decompose  $J(x)$  using SVD such that  $J(x) = UDV^\top$
  - Let tangent vectors be *rows of  $V$  corresponding to the largest singular values in  $D$*



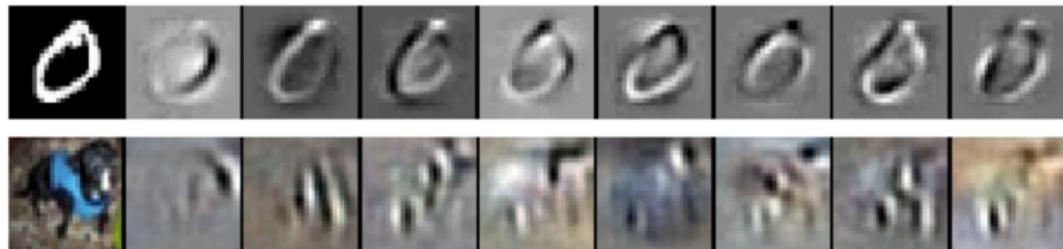
# Getting Tangent Vectors III

- In practice,  $J(x)$  usually has few large singular values
- Tangent vectors found by contractive/denoising autoencoders:



# Getting Tangent Vectors III

- In practice,  $J(\mathbf{x})$  usually has few large singular values
- Tangent vectors found by contractive/denoising autoencoders:



- Can be used by Tangent Prop [23]:
- Let  $\{\mathbf{v}^{(i,j)}\}_j$  be tangent vectors of each example  $\mathbf{x}^{(i)}$
- Trains an NN classifier  $f$  with cost penalty:  $\Omega[f] = \sum_{i,j} \nabla_{\mathbf{x}} f(\mathbf{x}^{(i)})^\top \mathbf{v}^{(i,j)}$ 
  - Points in the same manifold share the same label

# Outline

① Unsupervised Learning

② Self-Supervised Learning

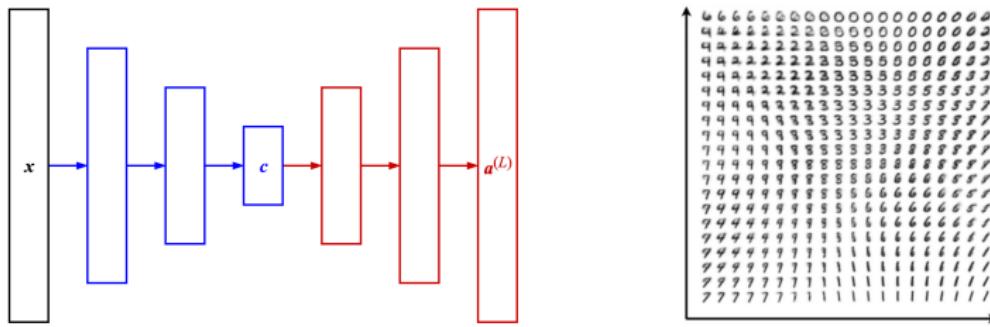
③ Autoencoders & Manifold Learning

④ Generative Adversarial Networks

- The Basics
- Challenges
- More GANs

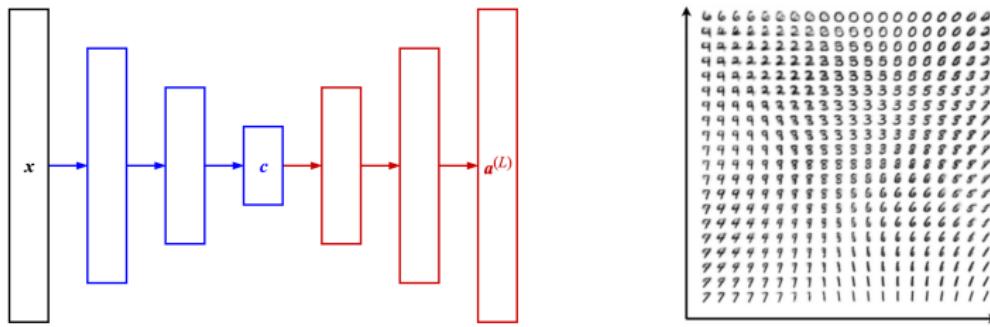
# Decoder as Data Generator

- Decoder of an autoencoder can be used to generate data points even with *synthetic codes*



# Decoder as Data Generator

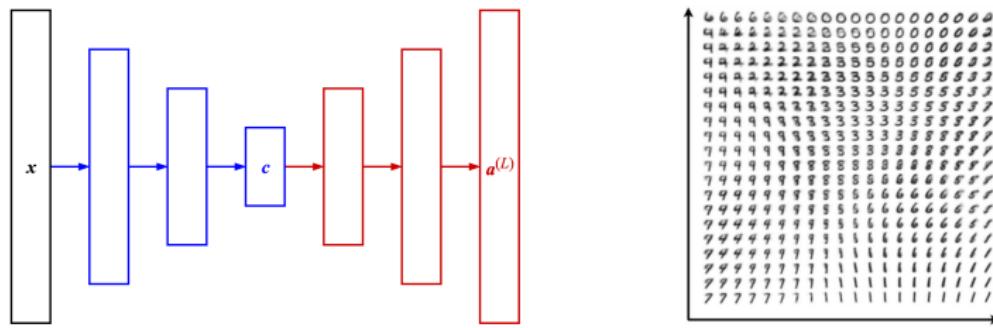
- Decoder of an autoencoder can be used to generate data points even with *synthetic codes*



- Problems:
  - Same  $c$ , same output

# Decoder as Data Generator

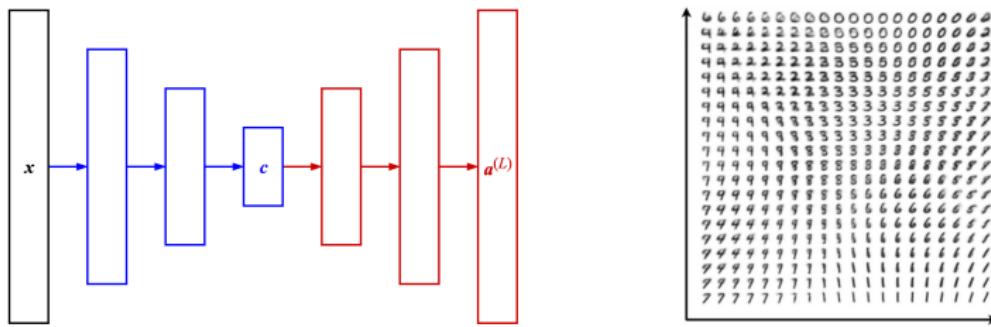
- Decoder of an autoencoder can be used to generate data points even with *synthetic codes*



- Problems:
  - Same  $c$ , same output → dropout layers, variational autoencoders [9]

# Decoder as Data Generator

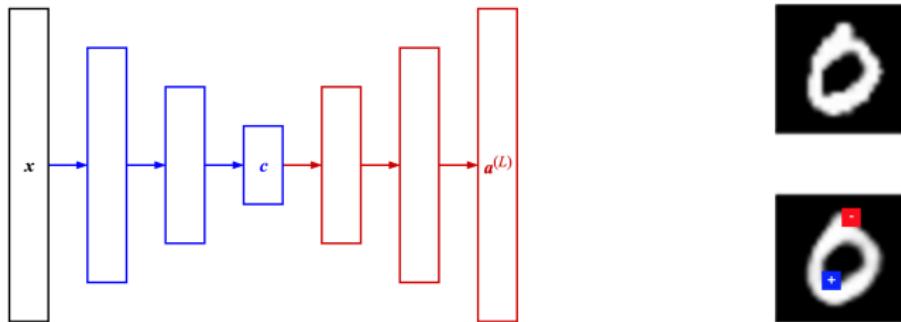
- Decoder of an autoencoder can be used to generate data points even with *synthetic codes*



- Problems:
  - Same  $c$ , same output → dropout layers, variational autoencoders [9]
  - Blurry images*

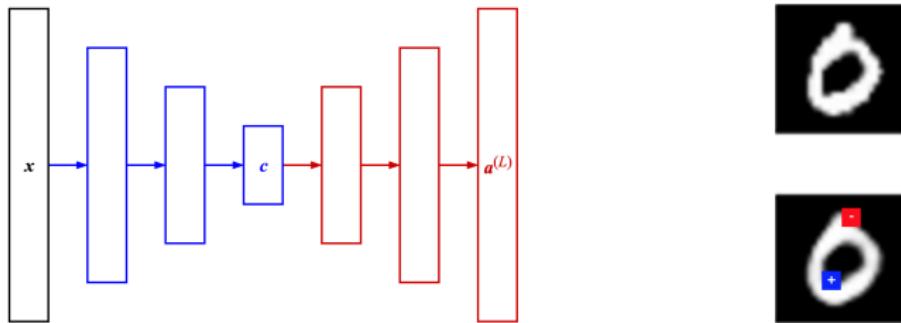
# Why Blurry Images?

- Cost function:  $\arg \min_{\Theta} -\log P(\mathbb{X} | \Theta) = \arg \min_{\Theta} -\sum_n \log P(x^{(n)} | \Theta)$
- Image generation: linear output units  $a^{(L)} = z^{(L)} = \hat{\mu}$  for  $x \sim \mathcal{N}(\mu, \Sigma)$ 
  - $-\log P(x^{(n)} | \Theta) = \|x^{(n)} - a^{(L)}\|^2$



# Why Blurry Images?

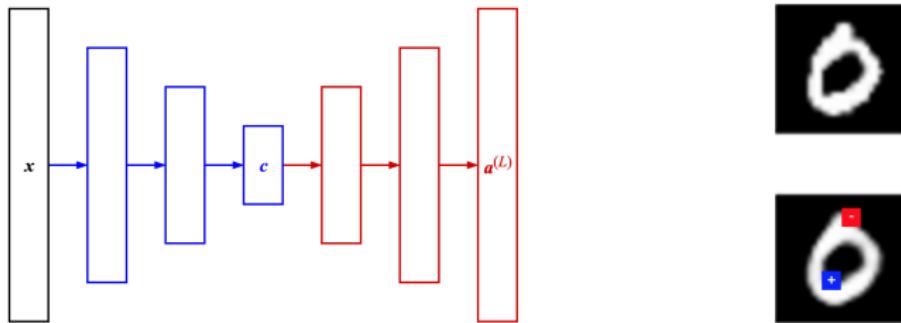
- Cost function:  $\arg \min_{\Theta} -\log P(\mathbb{X} | \Theta) = \arg \min_{\Theta} -\sum_n \log P(x^{(n)} | \Theta)$
- Image generation: linear output units  $a^{(L)} = z^{(L)} = \hat{\mu}$  for  $x \sim \mathcal{N}(\mu, \Sigma)$ 
  - $-\log P(x^{(n)} | \Theta) = \|x^{(n)} - a^{(L)}\|^2$



- Better assuming distribution for  $x$ ?

# Why Blurry Images?

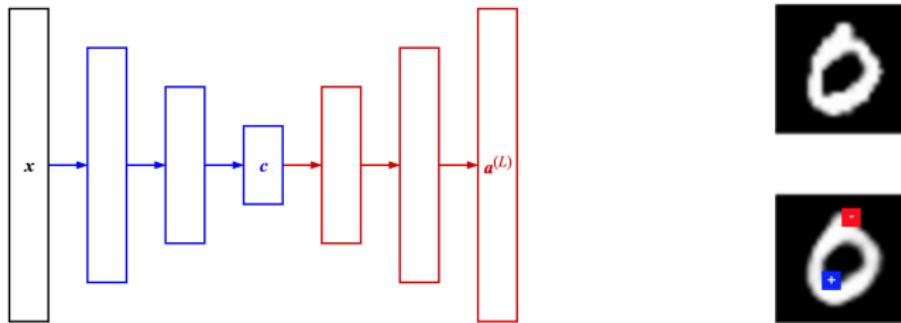
- Cost function:  $\arg \min_{\Theta} -\log P(\mathbb{X} | \Theta) = \arg \min_{\Theta} -\sum_n \log P(x^{(n)} | \Theta)$
- Image generation: linear output units  $a^{(L)} = z^{(L)} = \hat{\mu}$  for  $x \sim \mathcal{N}(\mu, \Sigma)$ 
  - $-\log P(x^{(n)} | \Theta) = \|x^{(n)} - a^{(L)}\|^2$



- Better assuming distribution for  $x$ ?  $P(x)$  may be very complex
- Better “goodness” measure?

# Why Blurry Images?

- Cost function:  $\arg \min_{\Theta} -\log P(\mathbb{X} | \Theta) = \arg \min_{\Theta} -\sum_n \log P(x^{(n)} | \Theta)$
- Image generation: linear output units  $a^{(L)} = z^{(L)} = \hat{\mu}$  for  $x \sim \mathcal{N}(\mu, \Sigma)$ 
  - $-\log P(x^{(n)} | \Theta) = \|x^{(n)} - a^{(L)}\|^2$



- Better assuming distribution for  $x$ ?  $P(x)$  may be very complex
- Better “goodness” measure? Why not use an NN to tell if a generated image is of good quality?

# Outline

① Unsupervised Learning

② Self-Supervised Learning

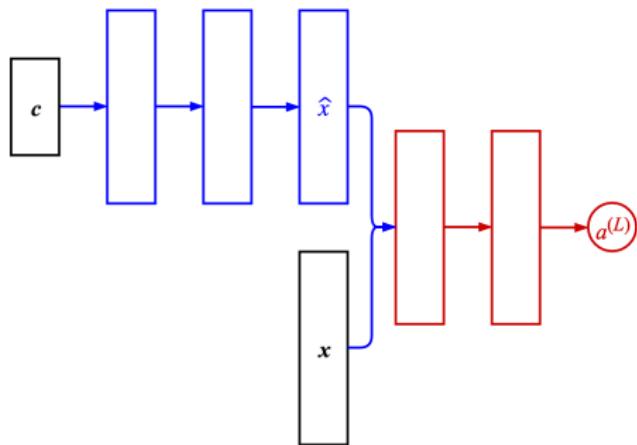
③ Autoencoders & Manifold Learning

④ Generative Adversarial Networks

- The Basics
- Challenges
- More GANs

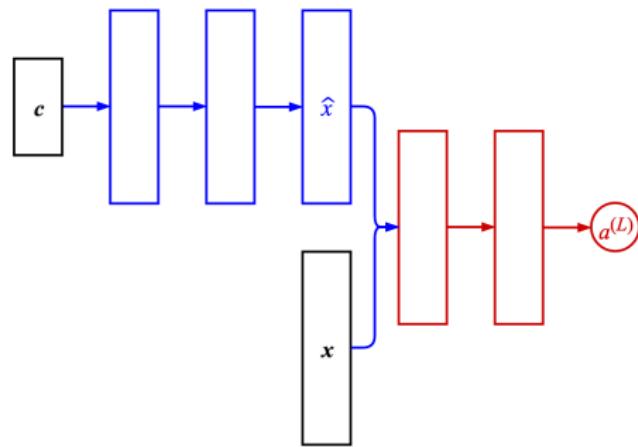
# Generative Adversarial Networks (GANs)

- *Generative adversarial network (GAN) [4]:*



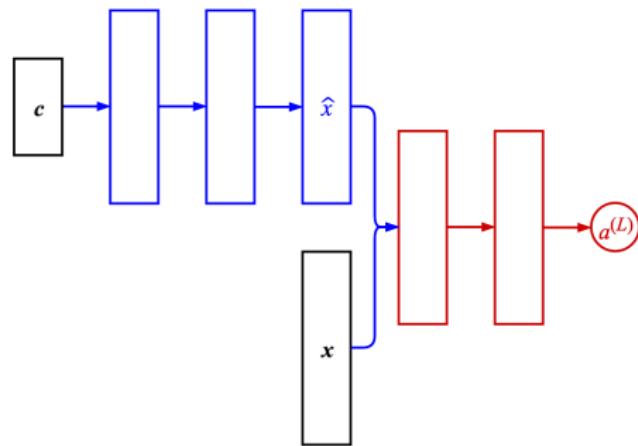
# Generative Adversarial Networks (GANs)

- **Generative adversarial network (GAN)** [4]:
- **Generator  $g$** : to generate data points from random codes
  - No need for “encoder” since the task is data synthesis



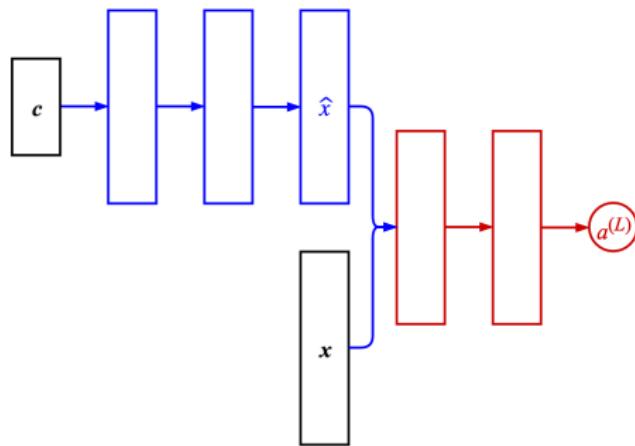
# Generative Adversarial Networks (GANs)

- **Generative adversarial network (GAN)** [4]:
- **Generator  $g$** : to generate data points from random codes
  - No need for “encoder” since the task is data synthesis
- **Discriminator  $f$** : to separate generated points from real ones
  - Weights for  $x$  and  $\hat{x}$  are tied
  - A binary classifier with Sigmoid output unit  $a^{(L)} = \hat{p}$  for  $P(y = \text{true point} | \mathbf{x}) \sim \text{Bernoulli}(\rho)$



# Generative Adversarial Networks (GANs)

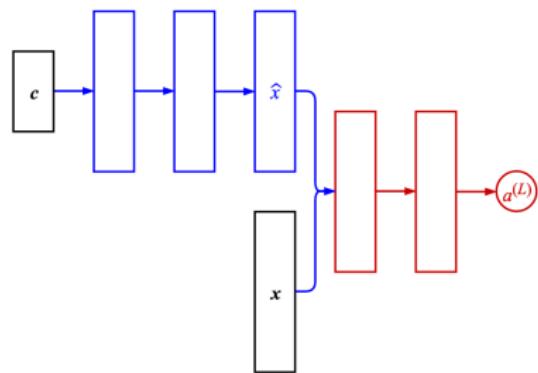
- **Generative adversarial network (GAN)** [4]:
- **Generator  $g$** : to generate data points from random codes
  - No need for “encoder” since the task is data synthesis
- **Discriminator  $f$** : to separate generated points from real ones
  - Weights for  $x$  and  $\hat{x}$  are tied
  - A binary classifier with Sigmoid output unit  $a^{(L)} = \hat{p}$  for  $P(y = \text{true point} | \mathbf{x}) \sim \text{Bernoulli}(\rho)$
- Goal: to train a  $g$  that tricks  $f$  into believing  $g(c)$  is real



# Cost Function

- Given  $N$  real training points and  $N$  generated points:

$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \log P(\mathbb{X} | \Theta_g, \Theta_f) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \end{aligned}$$



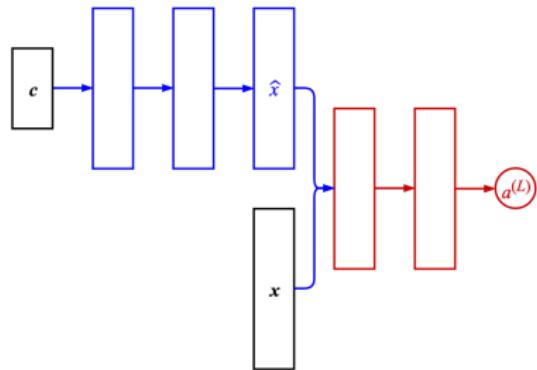
# Cost Function

- Given  $N$  real training points and  $N$  generated points:

$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \log P(\mathbb{X} | \Theta_g, \Theta_f) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_{n=1}^N \log \hat{\rho}^{(n)} + \sum_{m=1}^N \log(1 - \hat{\rho}^{(m)}) \end{aligned}$$

- Recall that  $f$  maximizes the log likelihood

$$\log P(\mathbb{X} | \Theta) \propto \sum_n \log P(y^{(n)} | \mathbf{x}^{(n)}, \Theta) = \sum_n \log \left[ (\hat{\rho}^{(n)})^{y^{(n)}} (1 - \hat{\rho}^{(n)})^{(1-y^{(n)})} \right]$$



# Cost Function

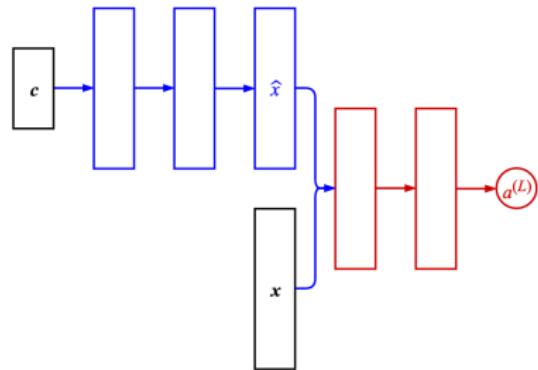
- Given  $N$  real training points and  $N$  generated points:

$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \log P(\mathbb{X} | \Theta_g, \Theta_f) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_{n=1}^N \log \hat{\rho}^{(n)} + \sum_{m=1}^N \log(1 - \hat{\rho}^{(m)}) \end{aligned}$$

- Recall that  $f$  maximizes the log likelihood

$$\log P(\mathbb{X} | \Theta) \propto \sum_n \log P(y^{(n)} | \mathbf{x}^{(n)}, \Theta) = \sum_n \log \left[ (\hat{\rho}^{(n)})^{y^{(n)}} (1 - \hat{\rho}^{(n)})^{(1-y^{(n)})} \right]$$

- Inner **max** first, then outer **min**



# Cost Function

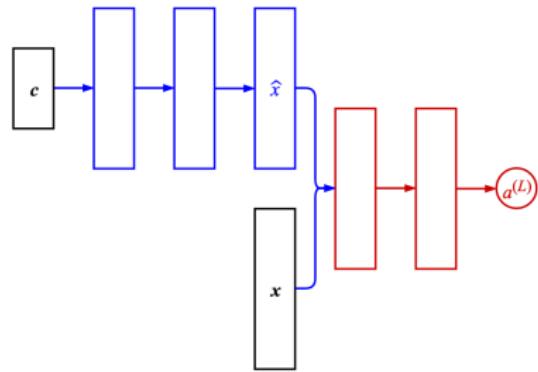
- Given  $N$  real training points and  $N$  generated points:

$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \log P(\mathbb{X} | \Theta_g, \Theta_f) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_{n=1}^N \log \hat{\rho}^{(n)} + \sum_{m=1}^N \log(1 - \hat{\rho}^{(m)}) \end{aligned}$$

- Recall that  $f$  maximizes the log likelihood

$$\log P(\mathbb{X} | \Theta) \propto \sum_n \log P(y^{(n)} | \mathbf{x}^{(n)}, \Theta) = \sum_n \log \left[ (\hat{\rho}^{(n)})^{y^{(n)}} (1 - \hat{\rho}^{(n)})^{(1-y^{(n)})} \right]$$

- Inner **max** first, then outer **min**
- $\hat{\rho}^{(n)}$  depends on  $\Theta_f$  only
- $\hat{\rho}^{(m)}$  depends on both  $\Theta_f$  and  $\Theta_g$



# Training: Alternative SGD

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:

# Training: Alternative SGD

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:

① *Repeat K times (with fixed  $\Theta_g$ ):*

- ① Sample  $N$  real points  $\{\mathbf{x}^{(n)}\}_n$  from  $\mathbb{X}$  and  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- ②  $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$

# Training: Alternative SGD

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:

① *Repeat K times (with fixed  $\Theta_g$ ):*

- ① Sample  $N$  real points  $\{\mathbf{x}^{(n)}\}_n$  from  $\mathbb{X}$  and  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- ②  $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$

② *Execute once (with fixed  $\Theta_f$ ):*

- ① Sample  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- ②  $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$

# Training: Alternative SGD

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:

① *Repeat K times (with fixed  $\Theta_g$ ):*

- ① Sample  $N$  real points  $\{\mathbf{x}^{(n)}\}_n$  from  $\mathbb{X}$  and  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- ②  $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$

② *Execute once (with fixed  $\Theta_f$ ):*

- ① Sample  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- ②  $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$

- Why limiting the steps ( $K$ ) when updating  $\Theta_f$ ?

# Training: Alternative SGD

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:

① *Repeat K times (with fixed  $\Theta_g$ ):*

- ① Sample  $N$  real points  $\{\mathbf{x}^{(n)}\}_n$  from  $\mathbb{X}$  and  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- ②  $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$

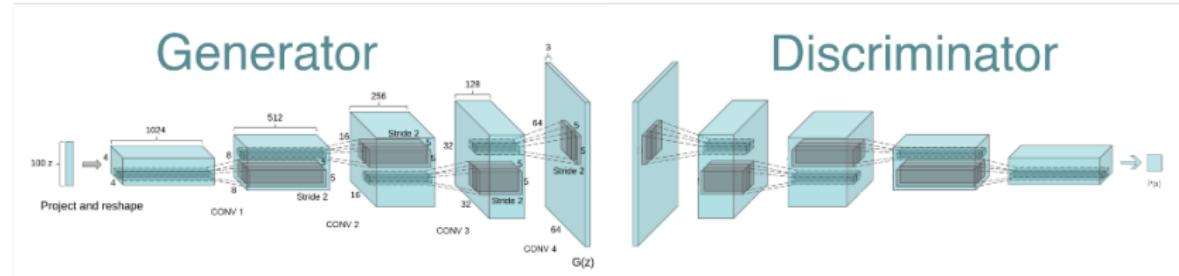
② *Execute once (with fixed  $\Theta_f$ ):*

- ① Sample  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- ②  $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$

- Why limiting the steps ( $K$ ) when updating  $\Theta_f$ ?
  - $f$  may overfit data and give very different values once  $g$  is updated
  - Limiting  $K$  so to prevent  $g$  from being updated for “wrong” target

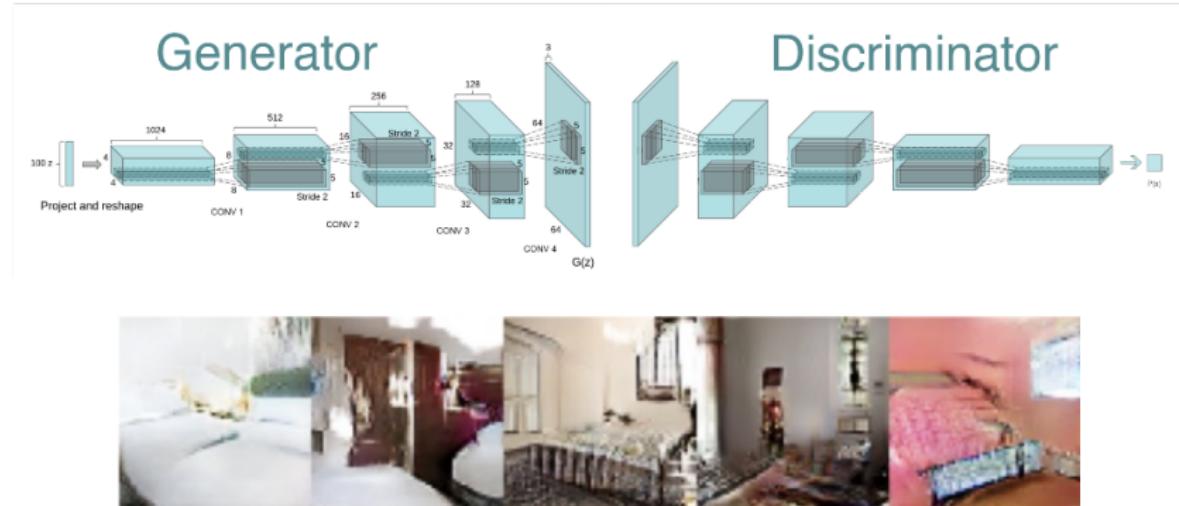
# Results

- Domain-specific architecture, e.g., DC-GAN [18]



# Results

- Domain-specific architecture, e.g., DC-GAN [18]



# GANs Are Hard to Train!

- [Tips for Training Stable GANs](#)
- [Keep Calm and train a GAN. Pitfalls and Tips...](#)
- [10 Lessons I Learned Training GANs for one Year](#)
- [GAN hacks on GitHub](#)

# Outline

① Unsupervised Learning

② Self-Supervised Learning

③ Autoencoders & Manifold Learning

## ④ Generative Adversarial Networks

- The Basics
- Challenges
- More GANs

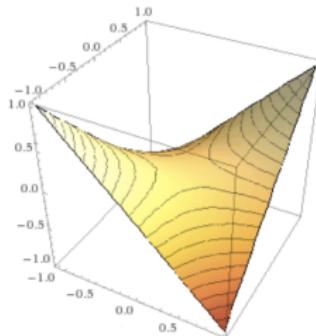
# Challenge: Non-Convergence

- The GAN training may *not* converge

# Challenge: Non-Convergence

- The GAN training may *not* converge
- The goal of GAN is to find a saddle point

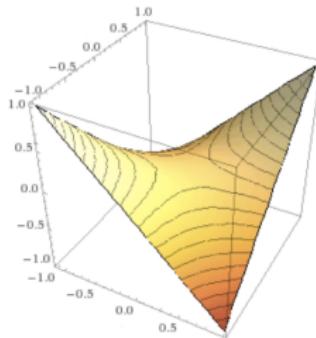
$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$



# Challenge: Non-Convergence

- The GAN training may *not* converge
- The goal of GAN is to find a saddle point

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

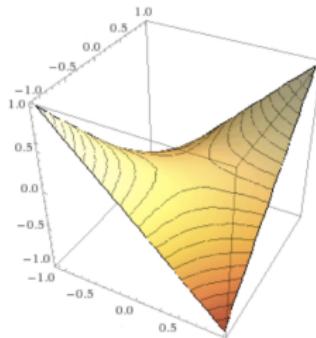


- The updated  $\Theta_f$  and  $\Theta_g$  may cancel each other's progress

# Challenge: Non-Convergence

- The GAN training may *not* converge
- The goal of GAN is to find a saddle point

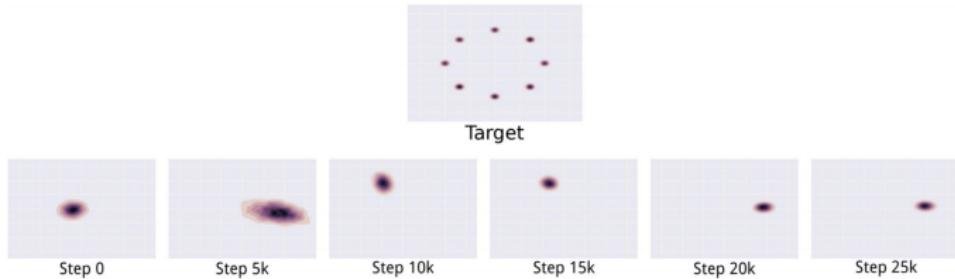
$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$



- The updated  $\Theta_f$  and  $\Theta_g$  may cancel each other's progress
- Requires human monitoring and termination

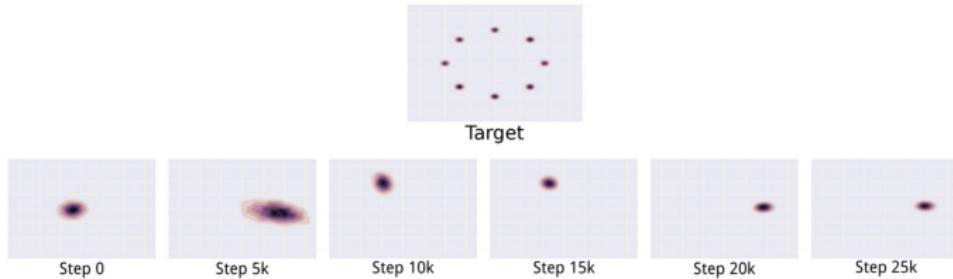
# Mode Collapsing

- Even worse: *mode collapsing*
  - $g$  may oscillate from generating one kind of points to generating another kind of points



# Mode Collapsing

- Even worse: *mode collapsing*
  - $g$  may oscillate from generating one kind of points to generating another kind of points



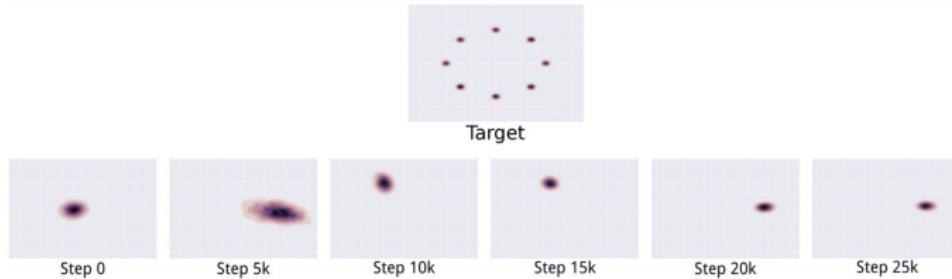
- When  $K$  is small, alternate SGD does not distinguish between  $\min_{\Theta_g} \max_{\Theta_f}$  and  $\max_{\Theta_f} \min_{\Theta_g}$

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- $\max_{\Theta_f} \min_{\Theta_g}$ ?

# Mode Collapsing

- Even worse: *mode collapsing*
  - $g$  may oscillate from generating one kind of points to generating another kind of points



- When  $K$  is small, alternate SGD does not distinguish between  $\min_{\Theta_g} \max_{\Theta_f}$  and  $\max_{\Theta_f} \min_{\Theta_g}$
- $$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(x^{(n)}) + \sum_m \log(1 - f(g(c^{(m)})))$$
- $\max_{\Theta_f} \min_{\Theta_g}$ ?  $g$  is encouraged to map every code to the “mode” that  $f$  believes is most likely to be real

# Solutions

- Minibatch discrimination [22]
  - In  $\max_{\Theta_f} \min_{\Theta_g}$  case,  $g$  collapses because  $\nabla_{\Theta_f} C$  are computed independently for each point
  - Why not augment each  $x^{(n)}/\hat{x}^{(n)}$  with **batch features**?

# Solutions

- Minibatch discrimination [22]
  - In  $\max_{\Theta_f} \min_{\Theta_g}$  case,  $g$  collapses because  $\nabla_{\Theta_f} C$  are computed independently for each point
  - Why not augment each  $x^{(n)}/\hat{x}^{(n)}$  with **batch features**?
  - If  $g$  collapses,  $f$  can tell this from batch features and reject fake points
  - Now,  $g$  needs to generate dissimilar points to fool  $f$

# Solutions

- Minibatch discrimination [22]

- In  $\max_{\Theta_f} \min_{\Theta_g}$  case,  $g$  collapses because  $\nabla_{\Theta_f} C$  are computed independently for each point
- Why not augment each  $x^{(n)}/\hat{x}^{(n)}$  with **batch features**?
- If  $g$  collapses,  $f$  can tell this from batch features and reject fake points
- Now,  $g$  needs to generate dissimilar points to fool  $f$

6	0	6	3	7	3	4	5	2	8
1	0	1	0	7	9	7	1	4	0
7	6	5	3	8	3	6	9	9	6
9	8	4	6	2	3	=	9	8	7
0	1	5	8	4	1	3	7	6	9
1	4	2	3	7	1	1	4	8	5
2	8	1	4	3	5	1	1	2	4
1	7	0	4	3	6	2	4	7	9
6	8	3	1	6	7	5	0	7	6
9	7	7	0	8	1	3	1	6	9

without

5	8	9	9	9	5	0	2	8	1
4	6	6	6	0	7	1	2	7	8
2	0	1	0	7	4	9	5	9	
1	7	3	8	3	2	1	6	3	8
0	2	6	0	6	9	3	6	1	6
5	7	9	9	5	8	6	4	4	5
1	5	4	6	7	8	7	9	7	3
7	4	4	7	9	5	6	8	1	1
1	3	9	0	9	1	1	9	7	
2	6	8	3	0	9	7	1	4	1

with

# Solutions

- Minibatch discrimination [22]

- In  $\max_{\Theta_f} \min_{\Theta_g}$  case,  $g$  collapses because  $\nabla_{\Theta_f} C$  are computed independently for each point
- Why not augment each  $x^{(n)}/\hat{x}^{(n)}$  with **batch features**?
- If  $g$  collapses,  $f$  can tell this from batch features and reject fake points
- Now,  $g$  needs to generate dissimilar points to fool  $f$

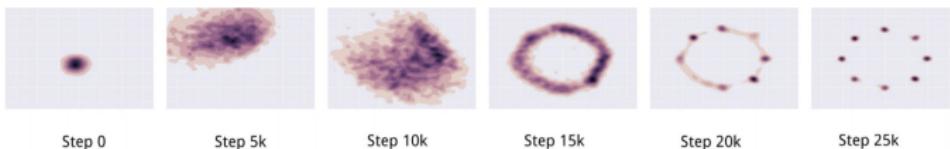
6	3	6	7	3	4	5	2	8	1
1	0	1	0	7	9	7	1	4	0
7	6	5	3	8	3	6	9	1	6
4	8	4	6	2	3	=	0	8	7
0	1	5	8	4	1	3	7	6	9
1	4	2	3	7	1	1	4	8	5
2	8	1	4	3	5	1	1	2	9
1	7	0	4	3	6	8	4	7	9
6	8	3	1	6	7	5	0	7	1
9	7	2	0	8	1	3	1	6	9

without

5	8	9	9	9	5	0	2	8	1
4	6	6	6	0	7	1	2	7	8
2	0	1	0	7	4	9	5	9	
1	7	3	8	3	2	1	6	3	8
0	2	6	0	6	9	3	6	1	6
5	7	9	9	5	8	6	4	4	5
1	5	4	6	7	8	7	1	7	3
7	4	4	7	9	5	6	8	1	1
1	3	9	0	9	1	1	1	6	7
2	6	8	3	0	9	7	1	4	1

with

- Unrolled GANs [15]: to back-propagate through **several max steps** when computing  $\nabla_{\Theta_g} C$



# Challenge: Balance between $g$ and $f$

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$  for  $K$  times
  - $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$

# Challenge: Balance between $g$ and $f$

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$  for  $K$  times
  - $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$
- Why limiting  $K$  when updating  $\Theta_f$ ?
- Too large  $K$ :
- Too small  $K$ :

# Challenge: Balance between $g$ and $f$

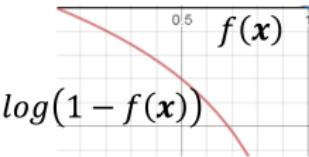
$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$  for  $K$  times
  - $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$
- Why limiting  $K$  when updating  $\Theta_f$ ?
- Too large  $K$ :
  - $f$  may overfit data, making  $g$  updated for “wrong” target  $f$
- Too small  $K$ :

# Challenge: Balance between $g$ and $f$

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$  for  $K$  times
  - $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$
- Why limiting  $K$  when updating  $\Theta_f$ ?
- Too large  $K$ :
  - $f$  may overfit data, making  $g$  updated for “wrong” target  $f$
  - Vanishing gradients:  $\nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$  too small to learn

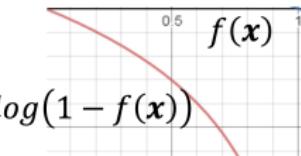


- Too small  $K$ :

# Challenge: Balance between $g$ and $f$

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$  for  $K$  times
  - $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$
- Why limiting  $K$  when updating  $\Theta_f$ ?
- Too large  $K$ :
  - $f$  may overfit data, making  $g$  updated for “wrong” target  $f$
  - Vanishing gradients:  $\nabla_{\Theta_g} [\sum_m \log(1 - f(g(\mathbf{c}^{(m)})))]$  too small to learn



- Too small  $K$ :
  - $g$  updated for “meaningless”  $f$

# Solution: Wasserstein GAN [1]

- Let  $f$  be a regressor **without** the sigmoid output layer
- Cost function:

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))$$

- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:

# Solution: Wasserstein GAN [1]

- Let  $f$  be a regressor **without** the sigmoid output layer
- Cost function:

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))$$

- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:
  - Repeat  $K$  times (with fixed  $\Theta_g$ ):
    - Sample  $N$  real points  $\{\mathbf{x}^{(n)}\}_n$  from  $\mathbb{X}$  and  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    - $\Theta_f \leftarrow \Theta_f + \eta \text{clip}(\nabla_{\Theta_f} [\sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))])$

# Solution: Wasserstein GAN [1]

- Let  $f$  be a regressor **without** the sigmoid output layer
- Cost function:

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))$$

- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:
  - Repeat  $K$  times (with fixed  $\Theta_g$ ):
    - Sample  $N$  real points  $\{\mathbf{x}^{(n)}\}_n$  from  $\mathbb{X}$  and  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    - $\Theta_f \leftarrow \Theta_f + \eta \text{clip}(\nabla_{\Theta_f} [\sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))])$
  - Execute once (with fixed  $\Theta_f$ ):
    - Sample  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    - $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [-\sum_m f(g(\mathbf{c}^{(m)}))]$

# GANs from Information Theory Perspective

- Review the Information Theory first!

# GANs from Information Theory Perspective

- Review the Information Theory first!
- Let  $P_{\text{data}} / P_g$  be distribution of  $\mathbf{x} / \hat{\mathbf{x}} = g(\mathbf{c})$

# GANs from Information Theory Perspective

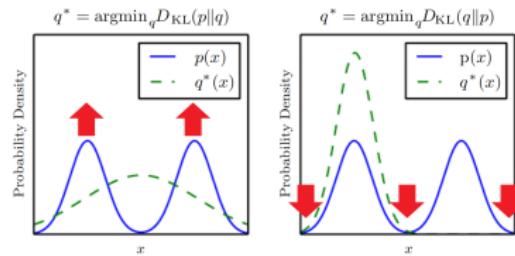
- Review the Information Theory first!
- Let  $P_{\text{data}} / P_g$  be distribution of  $\mathbf{x} / \hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find  $g$ :  $\arg \min_{\Theta_g} D_{\text{KL}}(P_{\text{data}} \| P_g)$

# GANs from Information Theory Perspective

- Review the Information Theory first!
- Let  $P_{\text{data}} / P_g$  be distribution of  $\mathbf{x} / \hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find  $g$ :  $\arg \min_{\Theta_g} D_{\text{KL}}(P_{\text{data}} \| P_g)$ 
  - Why not  $\arg \min_{\Theta_g} D_{\text{KL}}(P_g \| P_{\text{data}})$ ?

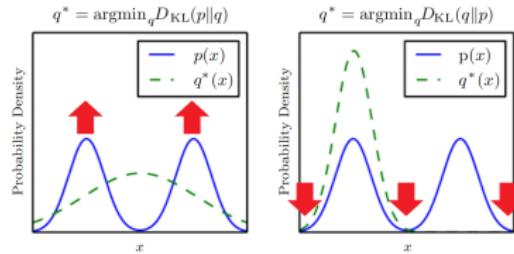
# GANs from Information Theory Perspective

- Review the Information Theory first!
- Let  $P_{\text{data}} / P_g$  be distribution of  $\mathbf{x} / \hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find  $g$ :  $\arg \min_{\Theta_g} D_{\text{KL}}(P_{\text{data}} \| P_g)$ 
  - Why not  $\arg \min_{\Theta_g} D_{\text{KL}}(P_g \| P_{\text{data}})$ ?



# GANs from Information Theory Perspective

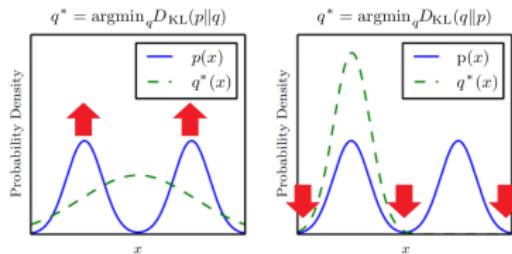
- Review the Information Theory first!
- Let  $P_{\text{data}} / P_g$  be distribution of  $\mathbf{x} / \hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find  $g$ :  $\arg \min_{\Theta_g} D_{\text{KL}}(P_{\text{data}} \| P_g)$ 
  - Why not  $\arg \min_{\Theta_g} D_{\text{KL}}(P_g \| P_{\text{data}})$ ?



- GAN:  $\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(x^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$

# GANs from Information Theory Perspective

- Review the Information Theory first!
- Let  $P_{\text{data}} / P_g$  be distribution of  $\mathbf{x} / \hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find  $g$ :  $\arg \min_{\Theta_g} D_{\text{KL}}(P_{\text{data}} \| P_g)$ 
  - Why not  $\arg \min_{\Theta_g} D_{\text{KL}}(P_g \| P_{\text{data}})$ ?



- GAN:  $\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(x^{(n)}) + \sum_m \log(1 - f(g(c^{(m)})))$
- Actually, the max term measures Jensen-Shannon divergence (a.k.a. symmetric KL divergence):

$$D_{\text{JS}}(P_{\text{data}} \| P_g) = \frac{1}{2} D_{\text{KL}}(P_{\text{data}} \| Q) + \frac{1}{2} D_{\text{KL}}(P_g \| Q), \text{ where } Q = \frac{1}{2}(P_g + P_{\text{data}})$$

# Why Jensen-Shannon Divergence? I

- Given a fixed  $g$ , we have

$$C^* = \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)})))$$

# Why Jensen-Shannon Divergence? I

- Given a fixed  $g$ , we have

$$\begin{aligned} C^* &= \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \\ &= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\mathbf{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\mathbf{x}}^{(m)})) \end{aligned}$$

# Why Jensen-Shannon Divergence? I

- Given a fixed  $g$ , we have

$$\begin{aligned} C^* &= \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \\ &= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\mathbf{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\mathbf{x}}^{(m)})) \\ &\approx \max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [\log f(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim P_g} [\log(1 - f(\mathbf{x}))] \end{aligned}$$

# Why Jensen-Shannon Divergence? I

- Given a fixed  $g$ , we have

$$\begin{aligned} C^* &= \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \\ &= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\mathbf{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\mathbf{x}}^{(m)})) \\ &\approx \max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [\log f(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim P_g} [\log(1 - f(\mathbf{x}))] \\ &= \max_{\Theta_f} \int_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} + \int_{\mathbf{x}} P_g(\mathbf{x}) \log(1 - f(\mathbf{x})) d\mathbf{x} \end{aligned}$$

# Why Jensen-Shannon Divergence? I

- Given a fixed  $g$ , we have

$$\begin{aligned} C^* &= \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \\ &= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\mathbf{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\mathbf{x}}^{(m)})) \\ &\approx \max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [\log f(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim P_g} [\log(1 - f(\mathbf{x}))] \\ &= \max_{\Theta_f} \int_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} + \int_{\mathbf{x}} P_g(\mathbf{x}) \log(1 - f(\mathbf{x})) d\mathbf{x} \\ &= \max_{\Theta_f} \int_{\mathbf{x}} [P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))] d\mathbf{x} \end{aligned}$$

# Why Jensen-Shannon Divergence? I

- Given a fixed  $g$ , we have

$$\begin{aligned} C^* &= \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log(1 - f(g(\mathbf{c}^{(m)}))) \\ &= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\mathbf{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\mathbf{x}}^{(m)})) \\ &\approx \max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [\log f(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim P_g} [\log(1 - f(\mathbf{x}))] \\ &= \max_{\Theta_f} \int_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} + \int_{\mathbf{x}} P_g(\mathbf{x}) \log(1 - f(\mathbf{x})) d\mathbf{x} \\ &= \max_{\Theta_f} \int_{\mathbf{x}} [P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))] d\mathbf{x} \end{aligned}$$

- To have  $C^*$ , we can find  $f$  maximizing

$$P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))$$

for each  $\mathbf{x}$

- Assuming that  $f$  has infinite capacity

# Why Jensen-Shannon Divergence? II

# Why Jensen-Shannon Divergence? II

- Given  $P_{\text{data}}$ ,  $P_g$ , and  $\mathbf{x}$ , what is the  $f(\mathbf{x})$  that maximizes

$$P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))?$$

- $f^*(\mathbf{x}) = \frac{P_{\text{data}}(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x})} \in [0, 1]$  [Proof]
- That is,

$$C^* = \max_{\Theta_f} \int_{\mathbf{x}} [P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))] d\mathbf{x}$$

# Why Jensen-Shannon Divergence? II

- Given  $P_{\text{data}}$ ,  $P_g$ , and  $\mathbf{x}$ , what is the  $f(\mathbf{x})$  that maximizes

$$P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))?$$

- $f^*(\mathbf{x}) = \frac{P_{\text{data}}(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x})} \in [0, 1]$  [Proof]
- That is,

$$\begin{aligned} C^* &= \max_{\Theta_f} \int_{\mathbf{x}} [P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))] d\mathbf{x} \\ &= \int_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x})} d\mathbf{x} \\ &\quad + \int_{\mathbf{x}} P_g(\mathbf{x}) \log \left(1 - \frac{P_g(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x})}\right) d\mathbf{x} \end{aligned}$$

# Why Jensen-Shannon Divergence? II

- Given  $P_{\text{data}}$ ,  $P_g$ , and  $x$ , what is the  $f(x)$  that maximizes

$$P_{\text{data}}(x) \log f(x) + P_g(x) \log(1 - f(x))?$$

- $f^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_g(x)} \in [0, 1]$  [Proof]
- That is,

$$\begin{aligned} C^* &= \max_{\Theta_f} \int_x [P_{\text{data}}(x) \log f(x) + P_g(x) \log(1 - f(x))] dx \\ &= \int_x P_{\text{data}}(x) \log \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_g(x)} dx \\ &\quad + \int_x P_g(x) \log \left(1 - \frac{P_g(x)}{P_{\text{data}}(x) + P_g(x)}\right) dx \\ &= -2 \log 2 + \int_x P_{\text{data}}(x) \log \frac{P_{\text{data}}(x)}{(P_{\text{data}}(x) + P_g(x))/2} dx \\ &\quad + \int_x P_g(x) \log \left(1 - \frac{P_g(x)}{(P_{\text{data}}(x) + P_g(x))/2}\right) dx \end{aligned}$$

# Why Jensen-Shannon Divergence? II

- Given  $P_{\text{data}}$ ,  $P_g$ , and  $\mathbf{x}$ , what is the  $f(\mathbf{x})$  that maximizes

$$P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))?$$

- $f^*(\mathbf{x}) = \frac{P_{\text{data}}(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x})} \in [0, 1]$  [Proof]
- That is,

$$\begin{aligned} C^* &= \max_{\Theta_f} \int_{\mathbf{x}} [P_{\text{data}}(\mathbf{x}) \log f(\mathbf{x}) + P_g(\mathbf{x}) \log(1 - f(\mathbf{x}))] d\mathbf{x} \\ &= \int_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x})} d\mathbf{x} \\ &\quad + \int_{\mathbf{x}} P_g(\mathbf{x}) \log(1 - \frac{P_g(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x})}) d\mathbf{x} \\ &= -2 \log 2 + \int_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}(\mathbf{x})}{(P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x}))/2} d\mathbf{x} \\ &\quad + \int_{\mathbf{x}} P_g(\mathbf{x}) \log(1 - \frac{P_g(\mathbf{x})}{(P_{\text{data}}(\mathbf{x}) + P_g(\mathbf{x}))/2}) d\mathbf{x} \\ &= -2 \log 2 + 2D_{JS}(P_{\text{data}} \| P_g) \end{aligned}$$

# Balance between $g$ and $f$ , Revisited I

- Cost function of GAN:

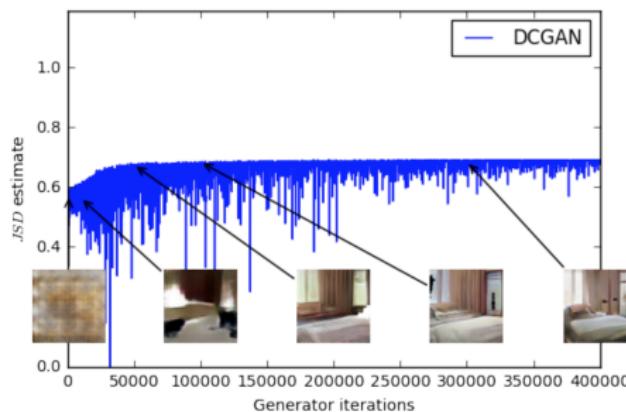
$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log (1 - f(g(\mathbf{c}^{(m)}))) \\ &= \arg \min_{\Theta_g} -2 \log 2 + 2 D_{JS}(P_{\text{data}} \| P_g) \end{aligned}$$

# Balance between $g$ and $f$ , Revisited I

- Cost function of GAN:

$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log (1 - f(g(\mathbf{c}^{(m)}))) \\ &= \arg \min_{\Theta_g} -2 \log 2 + 2 D_{JS}(P_{\text{data}} \| P_g) \end{aligned}$$

- However, no matter how  $g$  changes,  $D_{JS}(P_{\text{data}} \| P_g)$  remains high during the GAN training process

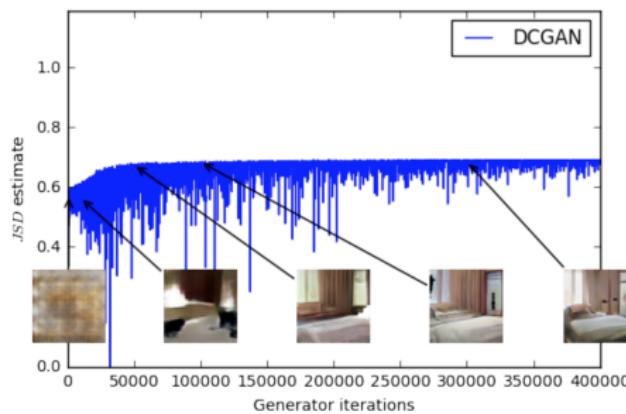


# Balance between $g$ and $f$ , Revisited I

- Cost function of GAN:

$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\mathbf{x}^{(n)}) + \sum_m \log (1 - f(g(\mathbf{c}^{(m)}))) \\ &= \arg \min_{\Theta_g} -2 \log 2 + 2 D_{JS}(P_{\text{data}} \| P_g) \end{aligned}$$

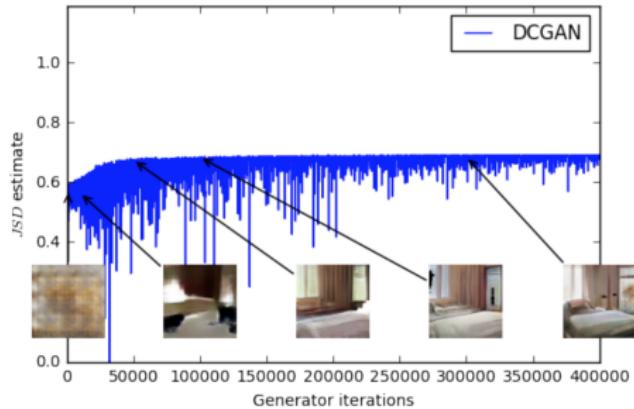
- However, no matter how  $g$  changes,  $D_{JS}(P_{\text{data}} \| P_g)$  remains high during the GAN training process



- There's something wrong with the design of the inner max problem!

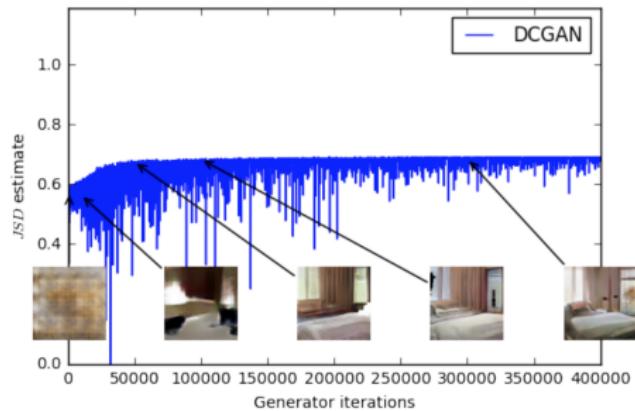
# Balance between $g$ and $f$ , Revisited II

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$



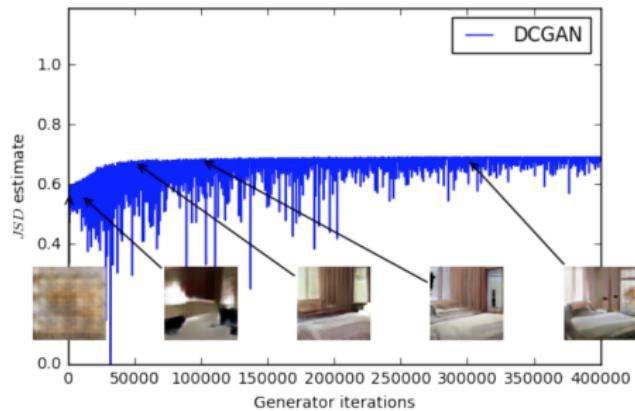
# Balance between $g$ and $f$ , Revisited II

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$
- $0 \leq D_{JS}(P_{\text{data}} \| P_g) \leq \log 2 \approx 0.69$



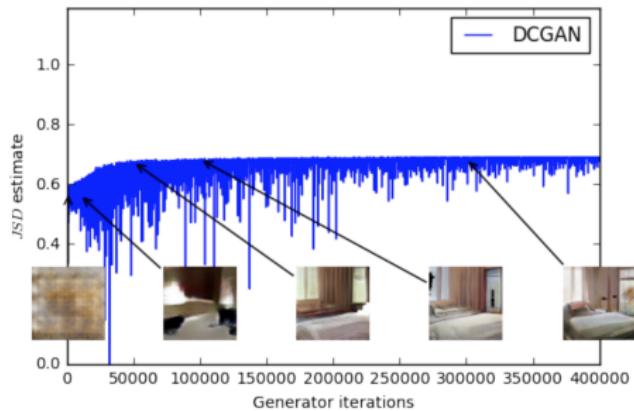
# Balance between $g$ and $f$ , Revisited II

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$
- $0 \leq D_{JS}(P_{\text{data}} \| P_g) \leq \log 2 \approx 0.69$
- When does  $D_{JS}(P_{\text{data}} \| P_g)$  reach its maximum value?



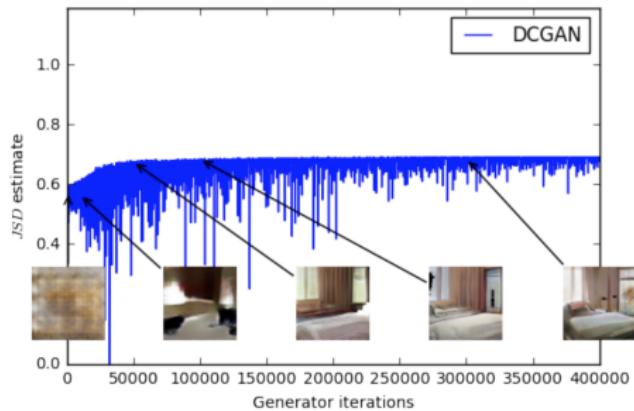
# Balance between $g$ and $f$ , Revisited II

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$
- $0 \leq D_{JS}(P_{\text{data}} \| P_g) \leq \log 2 \approx 0.69$
- When does  $D_{JS}(P_{\text{data}} \| P_g)$  reach its maximum value?
- Let's see the case when  $P_g$  and  $P_{\text{data}}$  are “disjointed”



# Balance between $g$ and $f$ , Revisited II

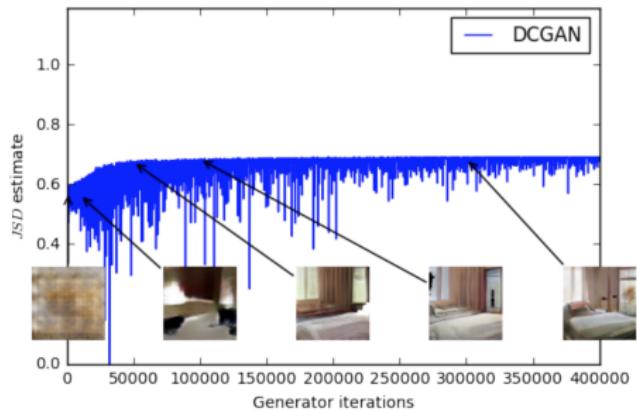
- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$
- $0 \leq D_{JS}(P_{\text{data}} \| P_g) \leq \log 2 \approx 0.69$
- When does  $D_{JS}(P_{\text{data}} \| P_g)$  reach its maximum value?
- Let's see the case when  $P_g$  and  $P_{\text{data}}$  are “disjointed”
- Suppose  $P_g(x) \neq 0 \Leftrightarrow P_{\text{data}}(x) = 0$  and  $P_g(x) = 0 \Leftrightarrow P_{\text{data}}(x) \neq 0$ , we have



$$D_{JS}(P_g \| P_{\text{data}}) = \frac{1}{2} D_{KL}(P_g \| \frac{P_g + P_{\text{data}}}{2}) + \frac{1}{2} D_{KL}(P_{\text{data}} \| \frac{P_g + P_{\text{data}}}{2})$$

# Balance between $g$ and $f$ , Revisited II

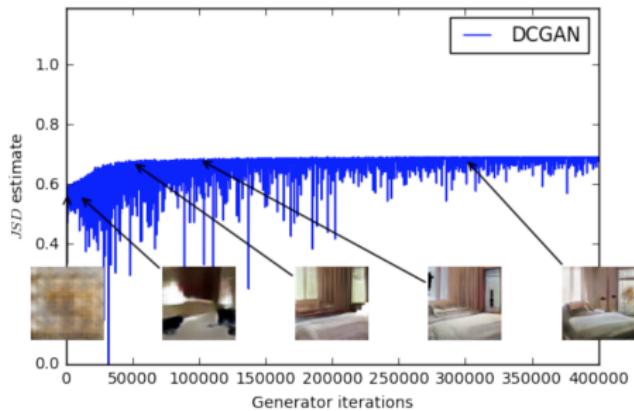
- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$
- $0 \leq D_{JS}(P_{\text{data}} \| P_g) \leq \log 2 \approx 0.69$
- When does  $D_{JS}(P_{\text{data}} \| P_g)$  reach its maximum value?
- Let's see the case when  $P_g$  and  $P_{\text{data}}$  are "disjointed"
- Suppose  $P_g(\mathbf{x}) \neq 0 \Leftrightarrow P_{\text{data}}(\mathbf{x}) = 0$  and  $P_g(\mathbf{x}) = 0 \Leftrightarrow P_{\text{data}}(\mathbf{x}) \neq 0$ , we have



$$\begin{aligned} D_{JS}(P_g \| P_{\text{data}}) &= \frac{1}{2} D_{KL}(P_g \| \frac{P_g + P_{\text{data}}}{2}) + \frac{1}{2} D_{KL}(P_{\text{data}} \| \frac{P_g + P_{\text{data}}}{2}) \\ &= \frac{1}{2} \int_{\mathbf{x}} P_g(\mathbf{x}) \log \frac{2P_g(\mathbf{x})}{P_g(\mathbf{x}) + P_{\text{data}}(\mathbf{x})} d\mathbf{x} \\ &\quad + \frac{1}{2} \int_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{2P_{\text{data}}(\mathbf{x})}{P_g(\mathbf{x}) + P_{\text{data}}(\mathbf{x})} d\mathbf{x} \end{aligned}$$

# Balance between $g$ and $f$ , Revisited II

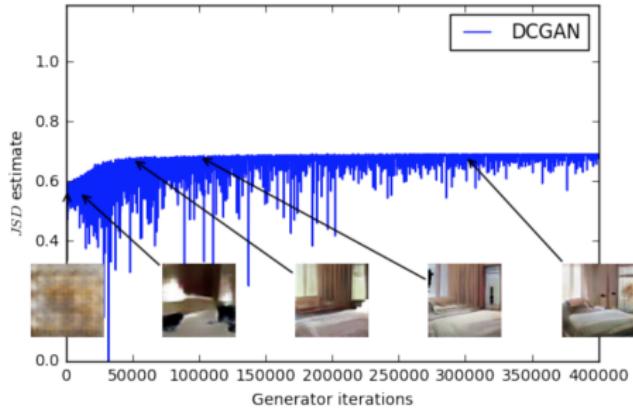
- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$
- $0 \leq D_{JS}(P_{\text{data}} \| P_g) \leq \log 2 \approx 0.69$
- When does  $D_{JS}(P_{\text{data}} \| P_g)$  reach its maximum value?
- Let's see the case when  $P_g$  and  $P_{\text{data}}$  are "disjointed"
- Suppose  $P_g(x) \neq 0 \Leftrightarrow P_{\text{data}}(x) = 0$  and  $P_g(x) = 0 \Leftrightarrow P_{\text{data}}(x) \neq 0$ , we have



$$\begin{aligned} D_{JS}(P_g \| P_{\text{data}}) &= \frac{1}{2} D_{KL}\left(P_g \parallel \frac{P_g + P_{\text{data}}}{2}\right) + \frac{1}{2} D_{KL}\left(P_{\text{data}} \parallel \frac{P_g + P_{\text{data}}}{2}\right) \\ &= \frac{1}{2} \int_x P_g(x) \log \frac{2P_g(x)}{P_g(x) + P_{\text{data}}(x)} dx \\ &\quad + \frac{1}{2} \int_x P_{\text{data}}(x) \log \frac{2P_{\text{data}}(x)}{P_g(x) + P_{\text{data}}(x)} dx \\ &= \frac{1}{2} \log 2 + \frac{1}{2} \log 2 = \log 2 \end{aligned}$$

# Balance between $g$ and $f$ , Revisited II

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$
- $0 \leq D_{JS}(P_{\text{data}} \| P_g) \leq \log 2 \approx 0.69$
- When does  $D_{JS}(P_{\text{data}} \| P_g)$  reach its maximum value?
- Let's see the case when  $P_g$  and  $P_{\text{data}}$  are "disjointed"
- Suppose  $P_g(x) \neq 0 \Leftrightarrow P_{\text{data}}(x) = 0$  and  $P_g(x) = 0 \Leftrightarrow P_{\text{data}}(x) \neq 0$ , we have

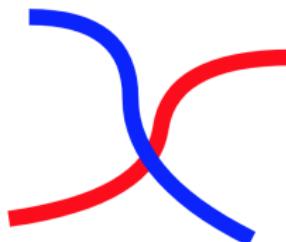


$$\begin{aligned} D_{JS}(P_g \| P_{\text{data}}) &= \frac{1}{2} D_{KL}(P_g \| \frac{P_g + P_{\text{data}}}{2}) + \frac{1}{2} D_{KL}(P_{\text{data}} \| \frac{P_g + P_{\text{data}}}{2}) \\ &= \frac{1}{2} \int_x P_g(x) \log \frac{2P_g(x)}{P_g(x) + P_{\text{data}}(x)} dx \\ &\quad + \frac{1}{2} \int_x P_{\text{data}}(x) \log \frac{2P_{\text{data}}(x)}{P_g(x) + P_{\text{data}}(x)} dx \\ &= \frac{1}{2} \log 2 + \frac{1}{2} \log 2 = \log 2 \end{aligned}$$

- Are  $P_g$  and  $P_{\text{data}}$  really disjointed during the GAN training?

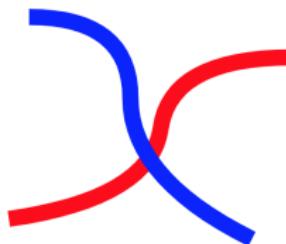
# Disjoining $P_g$ and $P_{\text{data}}$

- In a high dimensional space,  $\mathbf{x}$  and  $g(\mathbf{z})$  may reside in low dimensional manifolds
  - $P_g$  and  $P_{\text{data}}$  may have values only on the manifolds



# Disjoining $P_g$ and $P_{\text{data}}$

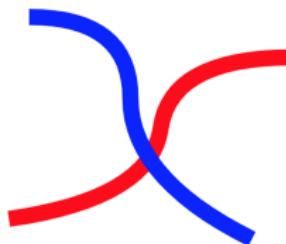
- In a high dimensional space,  $\mathbf{x}$  and  $g(\mathbf{z})$  may reside in low dimensional manifolds
  - $P_g$  and  $P_{\text{data}}$  may have values only on the manifolds



- $P_g$  and  $P_{\text{data}}$  can be very different initially during GAN training

# Disjoining $P_g$ and $P_{\text{data}}$

- In a high dimensional space,  $\mathbf{x}$  and  $g(\mathbf{z})$  may reside in low dimensional manifolds
  - $P_g$  and  $P_{\text{data}}$  may have values only on the manifolds



- $P_g$  and  $P_{\text{data}}$  can be very different initially during GAN training
- The intersections where  $P_g(\mathbf{x}) \neq 0$  and  $P_{\text{data}}(\mathbf{x}) \neq 0$  can be neglected
  - $P_g(\mathbf{x}) \neq 0 \Leftrightarrow P_{\text{data}}(\mathbf{x}) = 0$  and  $P_g(\mathbf{x}) = 0 \Leftrightarrow P_{\text{data}}(\mathbf{x}) \neq 0$  almost surely
  - Maximum JS divergence at all time

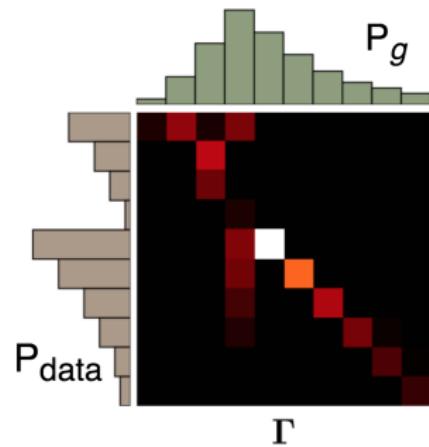
# Better Divergence Measure?

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$ 
  - $D_{JS}(P_{\text{data}} \| P_g)$  does **not** bring  $P_g$  and  $P_{\text{data}}$  closer during GAN training

# Better Divergence Measure?

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$ 
  - $D_{JS}(P_{\text{data}} \| P_g)$  does **not** bring  $P_g$  and  $P_{\text{data}}$  closer during GAN training
- Wasserstein (or earth-mover) distance:

$$\begin{aligned} W(P_{\text{data}}, P_g) &= \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} E_{(x, \hat{x}) \sim Q} [\|x - \hat{x}\|] \\ &= \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} \int_{(x, \hat{x})} Q(x, \hat{x}) \|x - \hat{x}\| d(x, \hat{x}) \end{aligned}$$

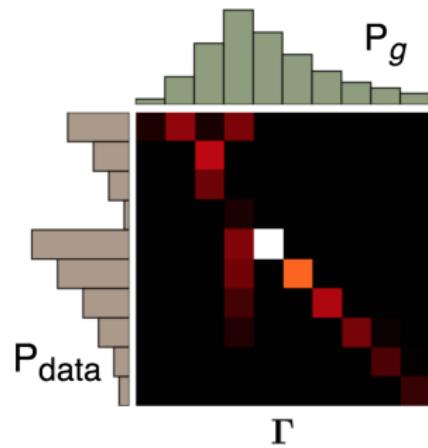


# Better Divergence Measure?

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$ 
  - $D_{JS}(P_{\text{data}} \| P_g)$  does **not** bring  $P_g$  and  $P_{\text{data}}$  closer during GAN training
- Wasserstein (or earth-mover) distance:

$$\begin{aligned} W(P_{\text{data}}, P_g) &= \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} E_{(x, \hat{x}) \sim Q} [\|x - \hat{x}\|] \\ &= \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} \int_{(x, \hat{x})} Q(x, \hat{x}) \|x - \hat{x}\| d(x, \hat{x}) \end{aligned}$$

- Intuitively, the minimal “cost” to change  $P_{\text{data}}$  into  $P_g$

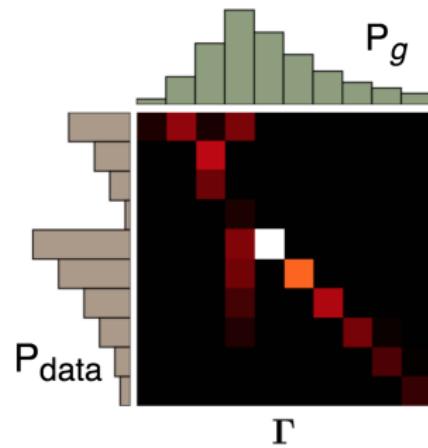


# Better Divergence Measure?

- GAN:  $\arg \min_{\Theta_g} D_{JS}(P_{\text{data}} \| P_g)$ 
  - $D_{JS}(P_{\text{data}} \| P_g)$  does **not** bring  $P_g$  and  $P_{\text{data}}$  closer during GAN training
- Wasserstein (or earth-mover) distance:

$$\begin{aligned} W(P_{\text{data}}, P_g) &= \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} E_{(x, \hat{x}) \sim Q} [\|x - \hat{x}\|] \\ &= \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} \int_{(x, \hat{x})} Q(x, \hat{x}) \|x - \hat{x}\| d(x, \hat{x}) \end{aligned}$$

- Intuitively, the minimal “cost” to change  $P_{\text{data}}$  into  $P_g$
- $W(P_{\text{data}}, P_g)$  measures the “divergence” between  $P_g$  and  $P_{\text{data}}$  **even when they are disjointed**



# Wasserstein GAN I

- W-GAN:  $\arg \min_{\Theta_g} W(P_{\text{data}}, P_g)$ 
  - $W(P_{\text{data}}, P_g) = \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} E_{(\mathbf{x}, \hat{\mathbf{x}}) \sim Q} [\|\mathbf{x} - \hat{\mathbf{x}}\|]$

# Wasserstein GAN I

- W-GAN:  $\arg \min_{\Theta_g} W(P_{\text{data}}, P_g)$ 
  - $W(P_{\text{data}}, P_g) = \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} E_{(\mathbf{x}, \hat{\mathbf{x}}) \sim Q} [\|\mathbf{x} - \hat{\mathbf{x}}\|]$
- Unfortunately,  $W(P_{\text{data}}, P_g)$  is hard to solve directly

# Wasserstein GAN II

## Theorem

Consider  $f$ 's that are Lipschitz continuous with constant 1, i.e.,

$$|f(\mathbf{x}) - f(\hat{\mathbf{x}})| \leq 1 \cdot \|\mathbf{x} - \hat{\mathbf{x}}\|, \forall \mathbf{x}, \hat{\mathbf{x}},$$

we have <sup>a</sup>

$$W(P_{\text{data}}, P_g) = \sup_f E_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - E_{\mathbf{x} \sim P_g} [f(\mathbf{x})]$$

# Wasserstein GAN II

## Theorem

Consider  $f$ 's that are Lipschitz continuous with constant 1, i.e.,

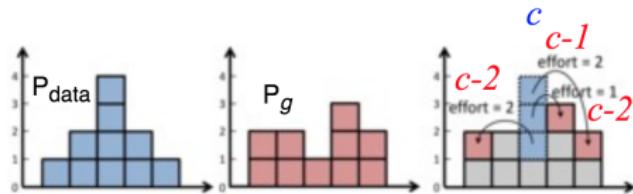
$$|f(\mathbf{x}) - f(\hat{\mathbf{x}})| \leq 1 \cdot \|\mathbf{x} - \hat{\mathbf{x}}\|, \forall \mathbf{x}, \hat{\mathbf{x}},$$

we have <sup>a</sup>

$$\begin{aligned} W(P_{\text{data}}, P_g) &= \sup_f E_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - E_{\mathbf{x} \sim P_g} [f(\mathbf{x})] \\ &= \sup_f \int_{\mathbf{x}} (P_{\text{data}}(\mathbf{x}) - P_g(\mathbf{x})) f(\mathbf{x}) d\mathbf{x}. \end{aligned}$$

---

<sup>a</sup><https://vincentherrmann.github.io/blog/wasserstein/>



# Wasserstein GAN II

## Theorem

Consider  $f$ 's that are Lipschitz continuous with constant 1, i.e.,

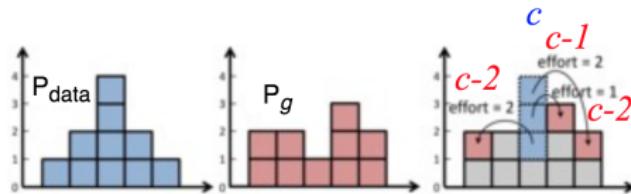
$$|f(\mathbf{x}) - f(\hat{\mathbf{x}})| \leq 1 \cdot \|\mathbf{x} - \hat{\mathbf{x}}\|, \forall \mathbf{x}, \hat{\mathbf{x}},$$

we have <sup>a</sup>

$$\begin{aligned} W(P_{\text{data}}, P_g) &= \sup_f E_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - E_{\mathbf{x} \sim P_g} [f(\mathbf{x})] \\ &= \sup_f \int_{\mathbf{x}} (P_{\text{data}}(\mathbf{x}) - P_g(\mathbf{x})) f(\mathbf{x}) d\mathbf{x}. \end{aligned}$$

---

<sup>a</sup><https://vincentherrmann.github.io/blog/wasserstein/>



- W-GAN [1]:  $\arg \min_{\Theta_g} \max_{\Theta_f} E_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - E_{\mathbf{x} \sim P_g} [f(\mathbf{x})]$

# Alternate SGD for W-GAN

$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim P_g} [f(\mathbf{x})] \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)})) \end{aligned}$$

- $f$  a regressor **without** the sigmoid output layer

# Alternate SGD for W-GAN

$$\begin{aligned} & \arg \min_{\Theta_g} \max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim P_g} [f(\mathbf{x})] \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)})) \end{aligned}$$

- $f$  a regressor **without** the sigmoid output layer
- Initialize  $\Theta_g$  for  $g$  and  $\Theta_f$  for  $f$
- At each SGD step/iteration:
  - ① Repeat  $K$  times (with fixed  $\Theta_g$ ):
    - ① Sample  $N$  real points  $\{\mathbf{x}^{(n)}\}_n$  from  $\mathbb{X}$  and  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    - ②  $\Theta_f \leftarrow \Theta_f + \eta \text{clip}(\nabla_{\Theta_f} [\sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))])$
  - ② Execute once (with fixed  $\Theta_f$ ):
    - ① Sample  $N$  codes from  $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    - ②  $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [-\sum_m f(g(\mathbf{c}^{(m)}))]$

# Why Gradient Clipping?

- Update rule for  $\Theta_f$ :

$$\Theta_f \leftarrow \Theta_f + \eta \text{clip}(\nabla_{\Theta_f} [\sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))])$$

- Gradient clipping:  $\forall w \in \Theta_f$ ,  $\text{clip}(w) = \max(\min(w, \tau), -\tau)$  for some threshold  $\tau > 0$
- Why?

# Why Gradient Clipping?

- Update rule for  $\Theta_f$ :

$$\Theta_f \leftarrow \Theta_f + \eta \text{clip}(\nabla_{\Theta_f} [\sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))])$$

- Gradient clipping:  $\forall w \in \Theta_f$ ,  $\text{clip}(w) = \max(\min(w, \tau), -\tau)$  for some threshold  $\tau > 0$
- Why?
- In W-GAN, we have :

$$\begin{aligned} & \arg \min_{\Theta_g} W(P_{\text{data}}, P_g) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)})) \end{aligned}$$

*only if f is 1-Lipschitz continuous*

# Why Gradient Clipping?

- Update rule for  $\Theta_f$ :

$$\Theta_f \leftarrow \Theta_f + \eta \text{clip}(\nabla_{\Theta_f} [\sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)}))])$$

- Gradient clipping:  $\forall w \in \Theta_f$ ,  $\text{clip}(w) = \max(\min(w, \tau), -\tau)$  for some threshold  $\tau > 0$
- Why?
- In W-GAN, we have :

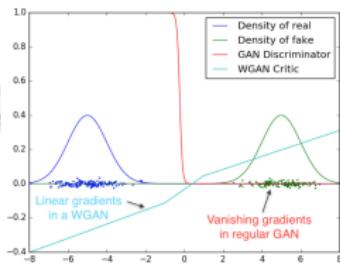
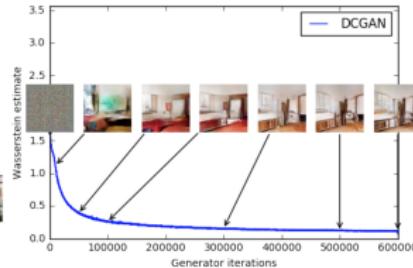
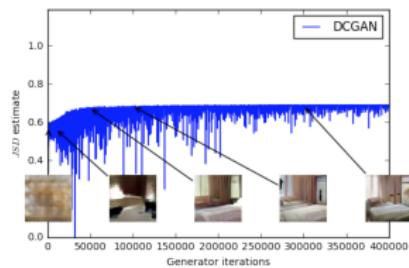
$$\begin{aligned} & \arg \min_{\Theta_g} W(P_{\text{data}}, P_g) \\ &= \arg \min_{\Theta_g} \max_{\Theta_f} \sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)})) \end{aligned}$$

*only if f is 1-Lipschitz continuous*

- Heuristics for making  $f$  1-Lipchitz

# Advantages of W-GAN

- “Corrects” the inner max problem
  - Wasserstein distance guides  $g$  even when  $P_g$  and  $P_{\text{data}}$  “disjointed”
  - Training less sensitive to  $K$  (balance between  $g$  and  $f$ )
- The max value can be used as a “stop” indicator
- $f$  a regressor, avoids vanishing gradients for  $g$



# Improved W-GAN I

- In practice, W-GAN training converges slowly and is unstable to  $\tau$

# Improved W-GAN I

- In practice, W-GAN training converges slowly and is unstable to  $\tau$
- W-GAN use a small  $\tau$  to make  $f$  1-Lipschitz continuous
- However, too small a  $\tau$  severely limits the capacity of  $f$  such it cannot actually maximize

$$\max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim P_g} [f(\mathbf{x})]$$

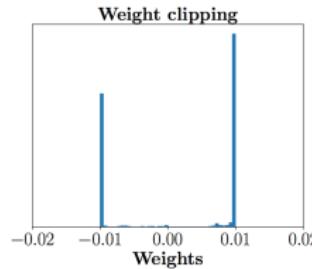
- $g$  is not updated for minimizing  $W(P_{\text{data}}, P_g)$

# Improved W-GAN I

- In practice, W-GAN training converges slowly and is unstable to  $\tau$
- W-GAN use a small  $\tau$  to make  $f$  1-Lipschitz continuous
- However, too small a  $\tau$  severely limits the capacity of  $f$  such it cannot actually maximize

$$\max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim P_g} [f(\mathbf{x})]$$

- $g$  is not updated for minimizing  $W(P_{\text{data}}, P_g)$
- Distribution of weight values of  $f$  ( $\tau = 0.01$ ):



- Exploding and vanishing gradients

# Improved W-GAN II

- If  $f$  is 1-Lipschitz, then  $\|\nabla f(\mathbf{x})\| \leq 1$  for all  $\mathbf{x}$
- Why not just penalize  $\|\nabla f(\mathbf{x})\| > 1$  for all  $\mathbf{x}$ ?
- Cost function:

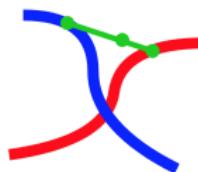
$$\begin{aligned} \arg \min_{\Theta_g} \max_{\Theta_f} E_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - E_{\mathbf{x} \sim P_g} [f(\mathbf{x})] \\ - \lambda E_{\mathbf{x} \sim P_{\text{penalty}}} [\max(0, \|\nabla f(\mathbf{x})\| - 1)] \end{aligned}$$

# Improved W-GAN II

- If  $f$  is 1-Lipschitz, then  $\|\nabla f(\mathbf{x})\| \leq 1$  for all  $\mathbf{x}$
- Why not just penalize  $\|\nabla f(\mathbf{x})\| > 1$  for all  $\mathbf{x}$ ?
- Cost function:

$$\arg \min_{\Theta_g} \max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim P_g} [f(\mathbf{x})] \\ - \lambda \mathbb{E}_{\mathbf{x} \sim P_{\text{penalty}}} [\max(0, \|\nabla f(\mathbf{x})\| - 1)]$$

- $P_{\text{penalty}}$ ?



# Improved W-GAN II

- If  $f$  is 1-Lipschitz, then  $\|\nabla f(\mathbf{x})\| \leq 1$  for all  $\mathbf{x}$
- Why not just penalize  $\|\nabla f(\mathbf{x})\| > 1$  for all  $\mathbf{x}$ ?
- Cost function:

$$\begin{aligned} \arg \min_{\Theta_g} \max_{\Theta_f} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim P_g} [f(\mathbf{x})] \\ - \lambda \mathbb{E}_{\mathbf{x} \sim P_{\text{penalty}}} [\max(0, \|\nabla f(\mathbf{x})\| - 1)] \end{aligned}$$

- $P_{\text{penalty}}$ ?

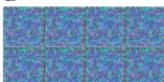


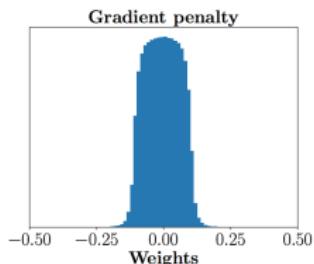
- W-GAN-GP [5]:

$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n f(\mathbf{x}^{(n)}) - \sum_m f(g(\mathbf{c}^{(m)})) - \lambda \sum_p (\|\nabla f(\mathbf{x}^{(p)})\| - 1)^2$$

- The larger  $f(\mathbf{x}^{(p)})$  the better (subject to  $\|\nabla f(\mathbf{x}^{(p)})\| \leq 1$ )
- Faster convergence

# Results

DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline ( $G$ : DCGAN, $D$ : DCGAN)			
			
$G$ : No BN and a constant number of filters, $D$ : DCGAN			
			
$G$ : 4-layer 512-dim ReLU MLP, $D$ : DCGAN			
			
No normalization in either $G$ or $D$			
			
Gated multiplicative nonlinearities everywhere in $G$ and $D$			
			
$tanh$ nonlinearities everywhere in $G$ and $D$			
			
101-layer ResNet $G$ and $D$			
			



# Challenge: Global Coherence

# Challenge: Global Coherence

- Large images generated by GANs usually lack global coherency

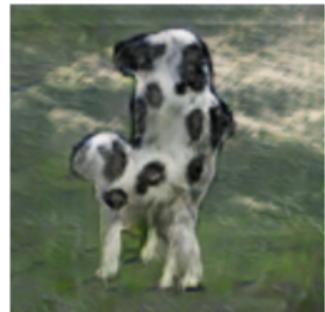
Counting



Perspective



Shape



# Challenge: Global Coherence

- Large images generated by GANs usually lack global coherency

Counting



Perspective



Shape



- A CNN, when used as  $f$ , detects *existence* of patterns more than their *relative positions*
  - $f$  loses track of the position of a pattern after several pooling layers
  - Relative position of patterns in  $\mathbb{X}$  may change due to different view angels
- Solutions?

# Challenge: Global Coherence

- Large images generated by GANs usually lack global coherency

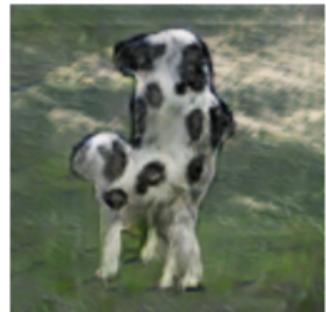
Counting



Perspective

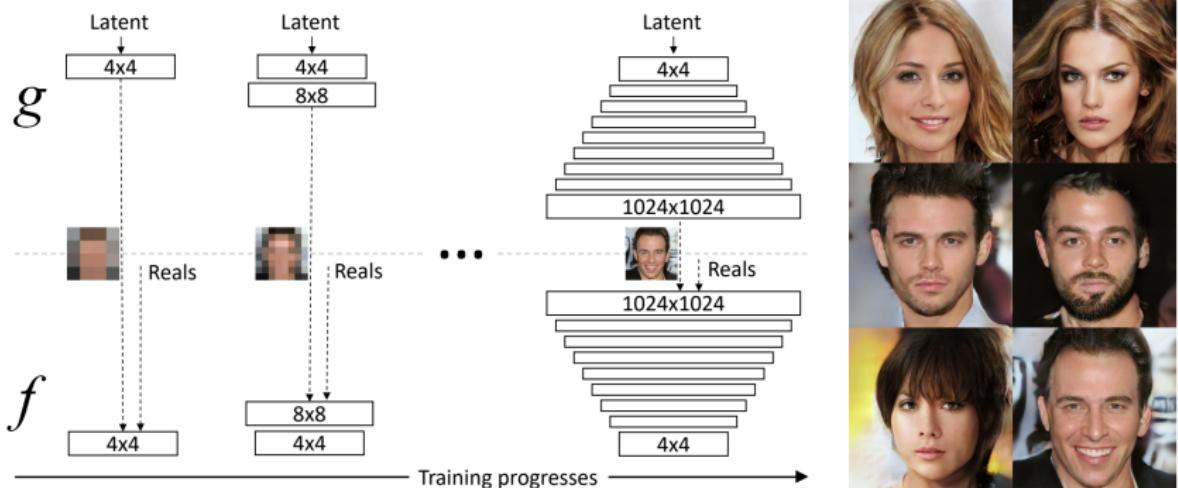


Shape



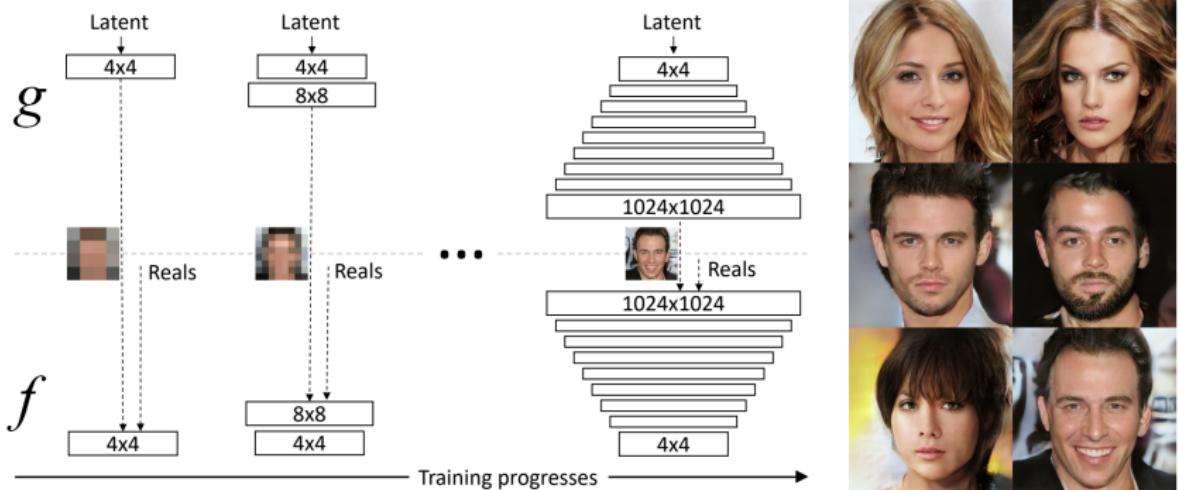
- A CNN, when used as  $f$ , detects *existence* of patterns more than their *relative positions*
  - $f$  loses track of the position of a pattern after several pooling layers
  - Relative position of patterns in  $\mathbb{X}$  may change due to different view angels
- Solutions? A better  $f$ , such as the CapsuleNet [21]?

# Progressive Growing of GANs [8]



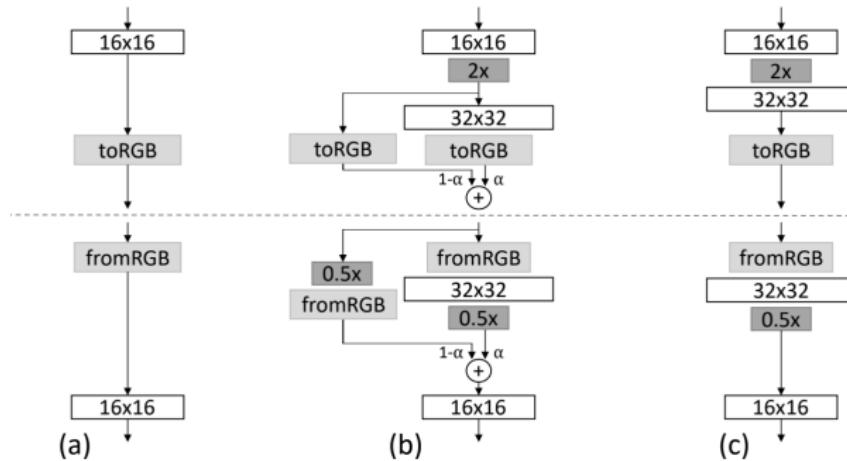
- Incrementally adds new layers in sequential GAN trainings
  - Convolution + upsampling for  $g$  each time
  - Convolution + downpooling for  $f$  each time
- Real images are downscaled to match the current resolution of  $g$

# Progressive Growing of GANs [8]



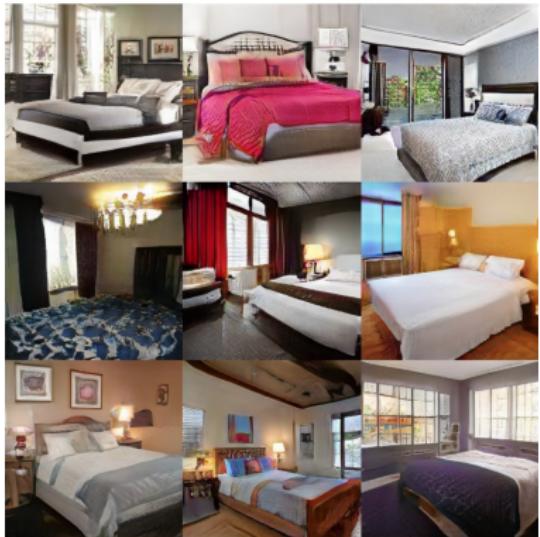
- Incrementally adds new layers in sequential GAN trainings
  - Convolution + upsampling for  $g$  each time
  - Convolution + downpooling for  $f$  each time
- Real images are downscaled to match the current resolution of  $g$
- Goal: to let new layers ***add details without ruining the context***

# Transition when Adding a New Layer



- Gradually increases  $\alpha$ 
  - New convolution layer in  $g/f$  learns to generate/detect details first
  - Then learns the “context”

# Results



- Minibatch discrimination [22] + W-GAN-GP [5] + progressive growing [8] + other tricks
- 2 ~ 6 times faster

# Outline

① Unsupervised Learning

② Self-Supervised Learning

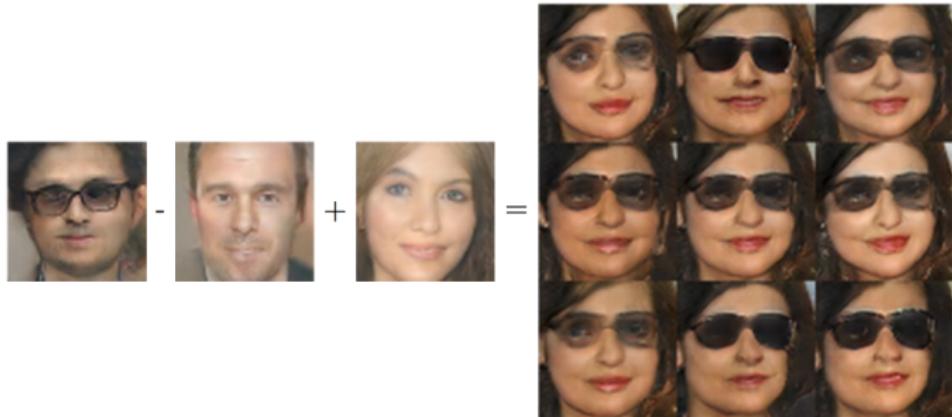
③ Autoencoders & Manifold Learning

## ④ Generative Adversarial Networks

- The Basics
- Challenges
- More GANs

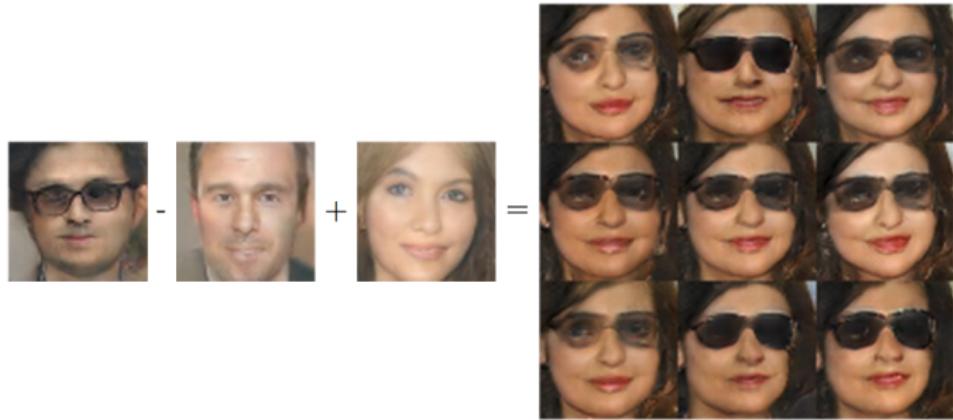
# Code Space Arithmetics

- DC-GAN [18] can learn to use codes in meaningful ways:



# Code Space Arithmetics

- DC-GAN [18] can learn to use codes in meaningful ways:



- Finding codes for images with constraints [29, 2] [▶ Demo 1](#) [▶ Demo 2](#)

$$\arg \min_{\mathbf{c}} \|mask(g(\mathbf{c})) - \text{constraint}\|_F$$

# Conditional GAN I

- Text to image synthesis [19]:  $\mathbb{X} = \{(\mathbf{x}^{(n)}, \phi^{(n)})\}_n$   
*"This bird is completely red with black wings and pointy beak."*



- How?

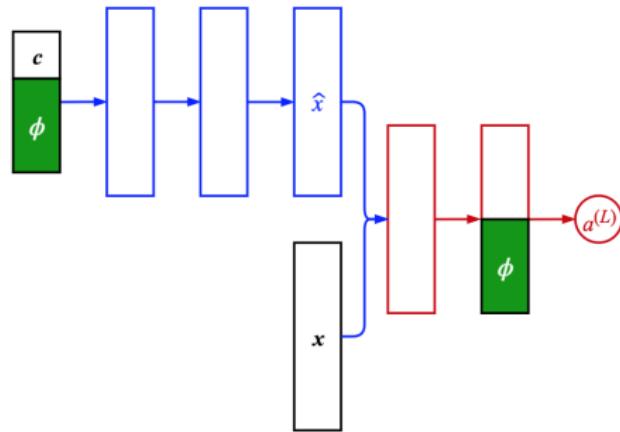
# Conditional GAN I

- Text to image synthesis [19]:  $\mathbb{X} = \{(x^{(n)}, \phi^{(n)})\}_n$

*"This bird is completely red with black wings and pointy beak."*

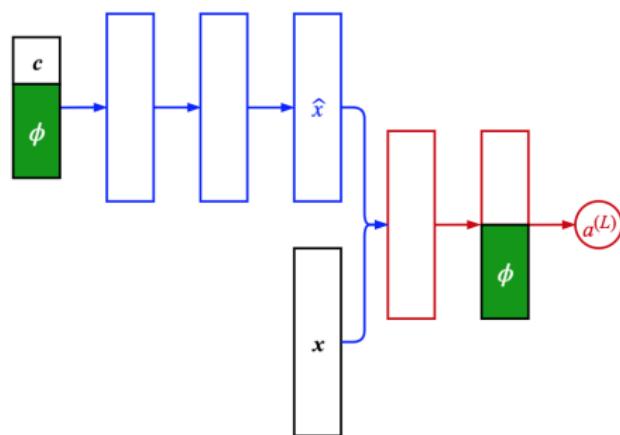


- How?



# Conditional GAN II

- Pitfall:  $g$  and  $f$  can choose to ignore the condition  $\phi$  altogether
- Solution?



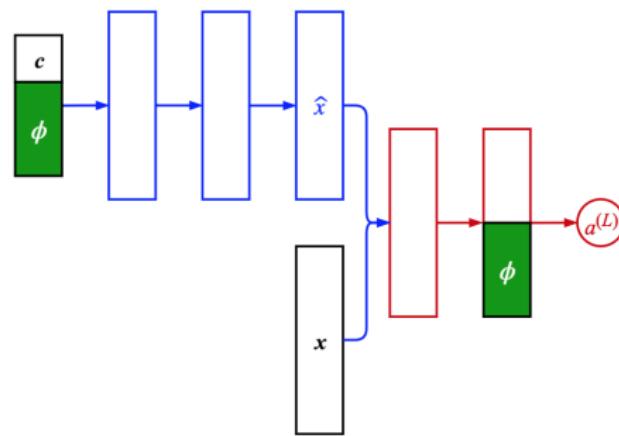
# Conditional GAN II

- Pitfall:  $g$  and  $f$  can choose to ignore the condition  $\phi$  altogether
- Solution? Conditioned labeling

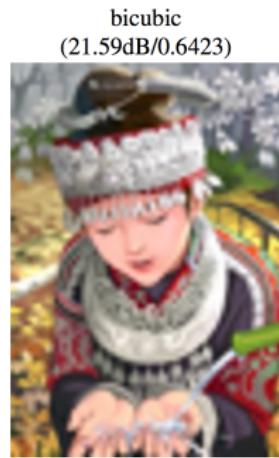
$$(\mathbf{x}^{(n)}, \phi^{(n)}) \Rightarrow \text{true}$$

$$(\mathbf{x}^{(n)}, \phi') \Rightarrow \text{false}, \forall \phi' \neq \phi^{(n)}$$

$$(\hat{\mathbf{x}}^{(m)}, \phi^{(m)}) \Rightarrow \text{false}$$

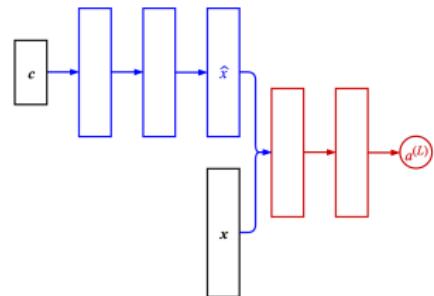


# Super Resolution [11]



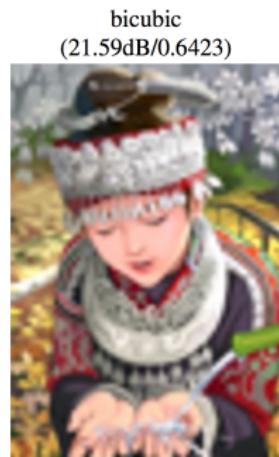
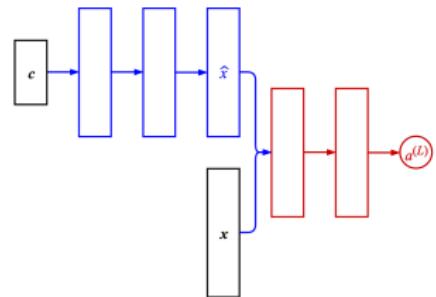
# Super Resolution [11]

- $g$ : low res img  $\rightarrow$  high res img
  - Training:  $c$ 's are downscaled images



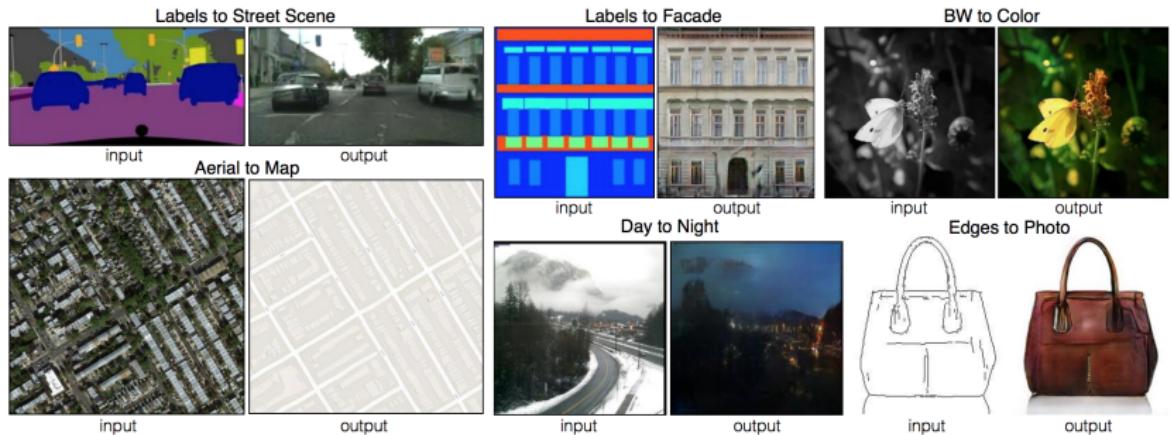
# Super Resolution [11]

- $g$ : low res img  $\rightarrow$  high res img
  - Training:  $c$ 's are downscaled images
- No “creativity,”  $f$  acts as a better loss metric



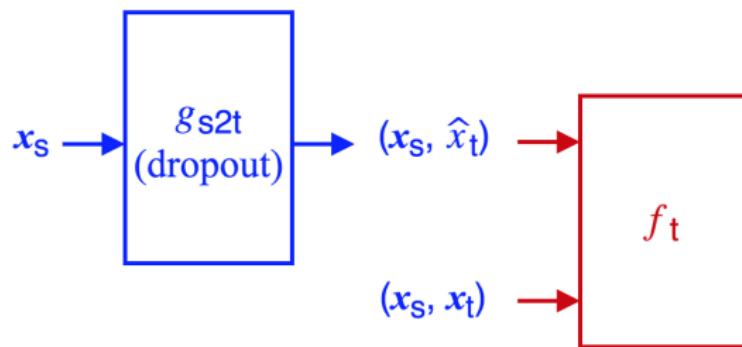
# Image-to-Image Translation [6] I

- Given an image  $x_{\text{src}}$  in source domain, generate image(s)  $x_{\text{target}}$  in target domain
  - $x_{\text{src}}$  and  $x_{\text{target}}$  are semantically aligned



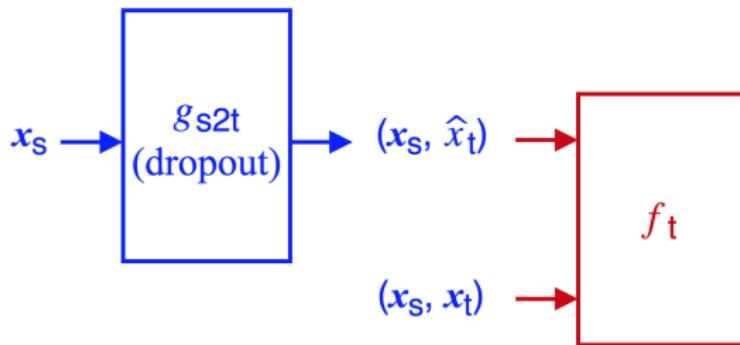
# Image-to-Image Translation [6] II

- Based on conditional GAN:
  - $\mathbf{c} = \mathbf{x}_{\text{src}}$ ;  $g(\mathbf{c}) = \mathbf{x}_{\text{target}}$
  - Conditioned labels
  - Uses dropout layers to create diversity (if needed)



# Image-to-Image Translation [6] II

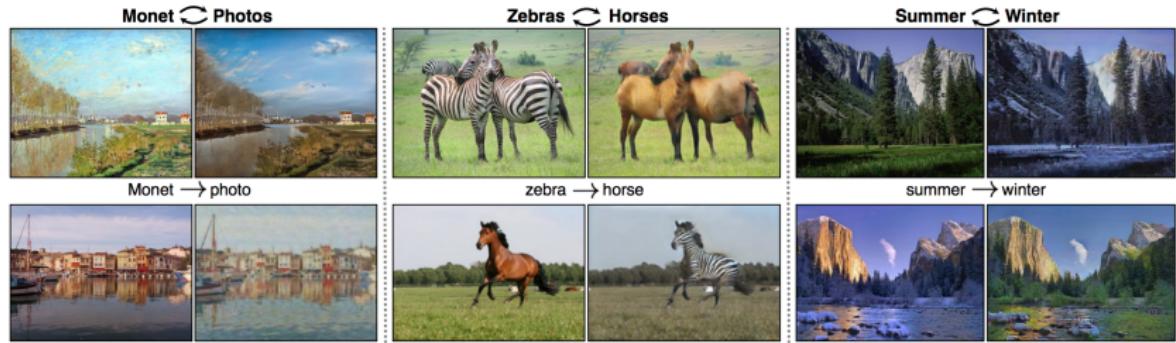
- Based on conditional GAN:
  - $\mathbf{c} = \mathbf{x}_{\text{src}}$ ;  $g(\mathbf{c}) = \mathbf{x}_{\text{target}}$
  - Conditioned labels
  - Uses dropout layers to create diversity (if needed)
- Requires **paired** examples  $\mathbb{X} = \{(\mathbf{x}_{\text{src}}^{(n)}, \mathbf{x}_{\text{target}}^{(n)})\}_n$



# Unpaired Image-to-Image Translation I

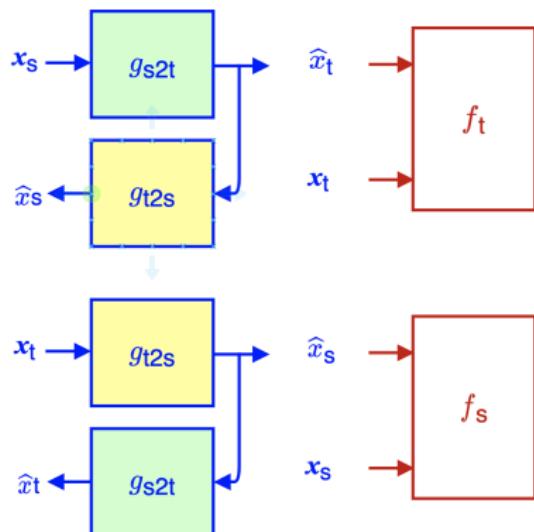
- What if the images in different domains are *unpaired*?

- $\mathbb{X} = \{\mathbf{x}_{\text{src}}^{(n)}\}_n \cup \{\mathbf{x}_{\text{target}}^{(n)}\}_n$



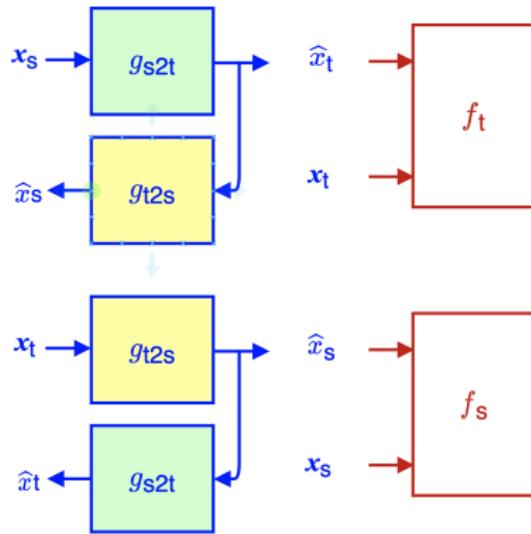
# Unpaired Image-to-Image Translation II

- Cycle GAN [30]: to train two generators  $g_{\text{src2target}}$  and  $g_{\text{target2src}}$  simultaneously in two GANs



# Unpaired Image-to-Image Translation II

- Cycle GAN [30]: to train two generators  $g_{\text{src2target}}$  and  $g_{\text{target2src}}$  simultaneously in two GANs
- Add a loss term  $\sum_n \|x_{\text{src}}^{(n)} - g_{\text{target2src}}(g_{\text{src2target}}(x_{\text{src}}^{(n)}))\|_F$  and  $\sum_n \|x_{\text{target}}^{(n)} - g_{\text{src2target}}(g_{\text{target2src}}(x_{\text{target}}^{(n)}))\|_F$  for  $g_{\text{src2target}}$  and  $g_{\text{target2src}}$



# How Does Cycle GAN Work?

# How Does Cycle GAN Work?

- Ideal:  $g_{\text{src2target}}$  and  $g_{\text{target2src}}$  learns to translate images

# How Does Cycle GAN Work?

- Ideal:  $g_{\text{src2target}}$  and  $g_{\text{target2src}}$  learns to translate images
- Reality:  $g_{\text{src2target}}$  and  $g_{\text{target2src}}$  learns to hide information [3]



# How Does Cycle GAN Work?

- Ideal:  $g_{\text{src2target}}$  and  $g_{\text{target2src}}$  learns to translate images
- Reality:  $g_{\text{src2target}}$  and  $g_{\text{target2src}}$  learns to hide information [3]



- *Unsupervised DNN models, including GANs, may not work as one may expect*

# Reference I

- [1] Martin Arjovsky, Soumith Chintala, and Léon Bottou.  
Wasserstein generative adversarial networks.  
In *International Conference on Machine Learning*, pages 214–223, 2017.
- [2] Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston.  
Neural photo editing with introspective adversarial networks.  
*arXiv preprint arXiv:1609.07093*, 2016.
- [3] Casey Chu, Andrey Zhmoginov, and Mark Sandler.  
Cyclegan, a master of steganography.  
*arXiv preprint arXiv:1712.02950*, 2017.
- [4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.  
Generative adversarial nets.  
In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

## Reference II

- [5] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville.  
Improved training of wasserstein gans.  
*arXiv preprint arXiv:1704.00028*, 2017.
- [6] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros.  
Image-to-image translation with conditional adversarial networks.  
*arXiv preprint arXiv:1611.07004*, 2016.
- [7] Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu.  
Video pixel networks.  
*arXiv preprint arXiv:1610.00527*, 2016.
- [8] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen.  
Progressive growing of gans for improved quality, stability, and variation.  
*arXiv preprint arXiv:1710.10196*, 2017.

# Reference III

- [9] Diederik P Kingma and Max Welling.  
Auto-encoding variational bayes.  
*arXiv preprint arXiv:1312.6114*, 2013.
- [10] Quoc V Le and Tomas Mikolov.  
Distributed representations of sentences and documents.  
In *ICML*, volume 14, pages 1188–1196, 2014.
- [11] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al.  
Photo-realistic single image super-resolution using a generative adversarial network.  
*arXiv preprint arXiv:1609.04802*, 2016.
- [12] Daniel D Lee and H Sebastian Seung.  
Learning the parts of objects by non-negative matrix factorization.  
*Nature*, 401(6755):788–791, 1999.

## Reference IV

- [13] Daniel D Lee and H Sebastian Seung.  
Algorithms for non-negative matrix factorization.  
In *Advances in neural information processing systems*, pages 556–562, 2001.
- [14] William Lotter, Gabriel Kreiman, and David Cox.  
Deep predictive coding networks for video prediction and unsupervised learning.  
*arXiv preprint arXiv:1605.08104*, 2016.
- [15] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein.  
Unrolled generative adversarial networks.  
*arXiv preprint arXiv:1611.02163*, 2016.
- [16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean.  
Efficient estimation of word representations in vector space.  
*arXiv preprint arXiv:1301.3781*, 2013.

## Reference V

- [17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean.  
Distributed representations of words and phrases and their compositionality.  
In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [18] Alec Radford, Luke Metz, and Soumith Chintala.  
Unsupervised representation learning with deep convolutional generative adversarial networks.  
*arXiv preprint arXiv:1511.06434*, 2015.
- [19] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee.  
Generative adversarial text to image synthesis.  
*arXiv preprint arXiv:1605.05396*, 2016.

## Reference VI

- [20] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio.  
Contractive auto-encoders: Explicit invariance during feature extraction.  
In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 833–840, 2011.
- [21] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton.  
Dynamic routing between capsules.  
In *Advances in Neural Information Processing Systems*, pages 3857–3867, 2017.
- [22] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen.  
Improved techniques for training gans.  
In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

## Reference VII

- [23] Patrice Simard, Bernard Victorri, Yann LeCun, and John S Denker. Tangent prop-a formalism for specifying selected invariances in an adaptive network.  
In *NIPS*, volume 91, pages 895–903, 1991.
- [24] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al.  
Conditional image generation with pixelcnn decoders.  
In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.
- [25] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu.  
Pixel recurrent neural networks.  
In *International Conference on Machine Learning*, pages 1747–1756, 2016.

## Reference VIII

- [26] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol.  
Extracting and composing robust features with denoising autoencoders.  
In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.
- [27] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba.  
Anticipating the future by watching unlabeled video.  
*arXiv preprint arXiv:1504.08023*, 2015.
- [28] Matthew D Zeiler and Rob Fergus.  
Visualizing and understanding convolutional networks.  
In *European conference on computer vision*, pages 818–833. Springer, 2014.

# Reference IX

- [29] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros.  
Generative visual manipulation on the natural image manifold.  
In *European Conference on Computer Vision*, pages 597–613.  
Springer, 2016.
- [30] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros.  
Unpaired image-to-image translation using cycle-consistent adversarial  
networks.  
*arXiv preprint arXiv:1703.10593*, 2017.