Unsupervised Learning

Shan-Hung Wu
shwu@cs.nthu.edu.tw

Department of Computer Science,
National Tsing Hua University, Taiwan

Machine Learning

Outline

Outline

Unsupervised Learning

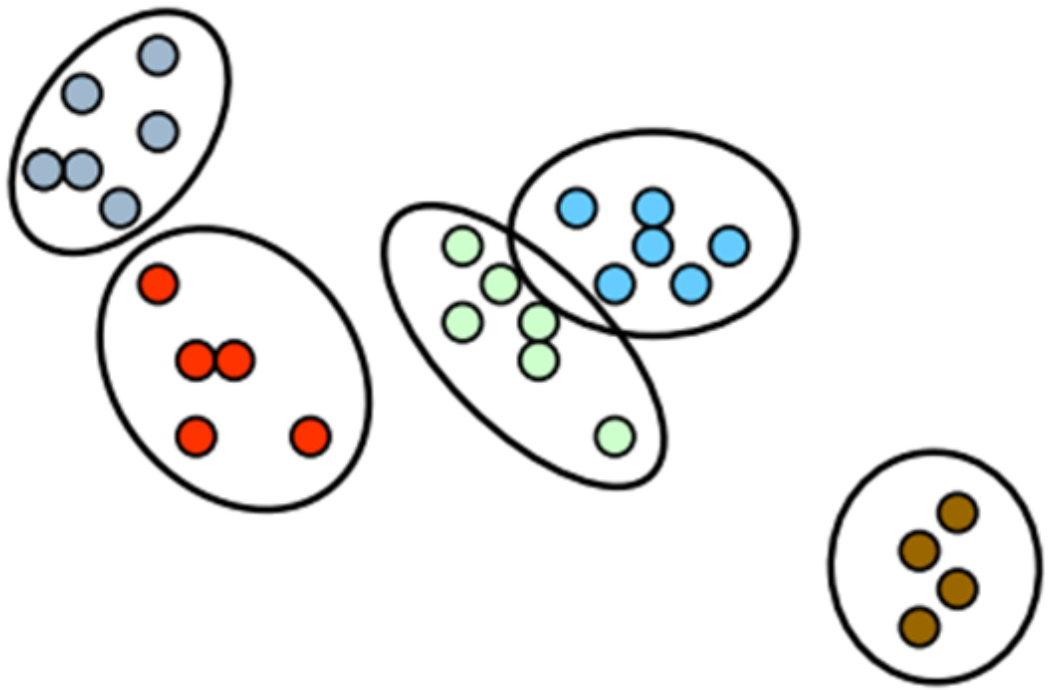
- Dataset: $\mathbb{X} = \{\boldsymbol{x}^{(i)}\}_i$, where $\boldsymbol{x}^{(i)}$'s are i.i.d. samples of $\mathbf{x}$
  - No supervision (e.g., labels)

Unsupervised Learning

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

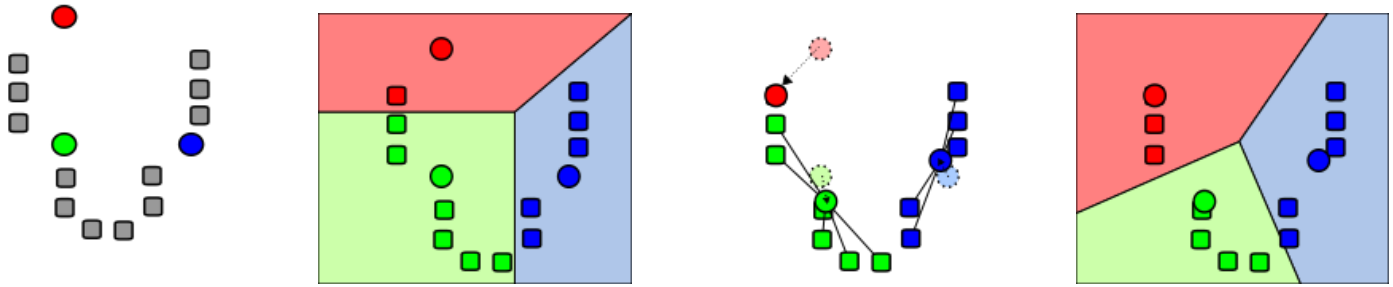
Clustering I

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



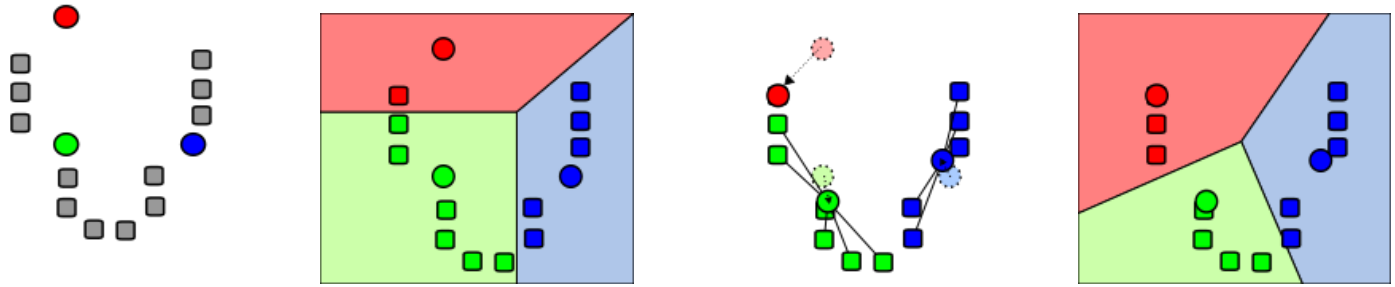
Clustering II

- K-means algorithm (K fixed):

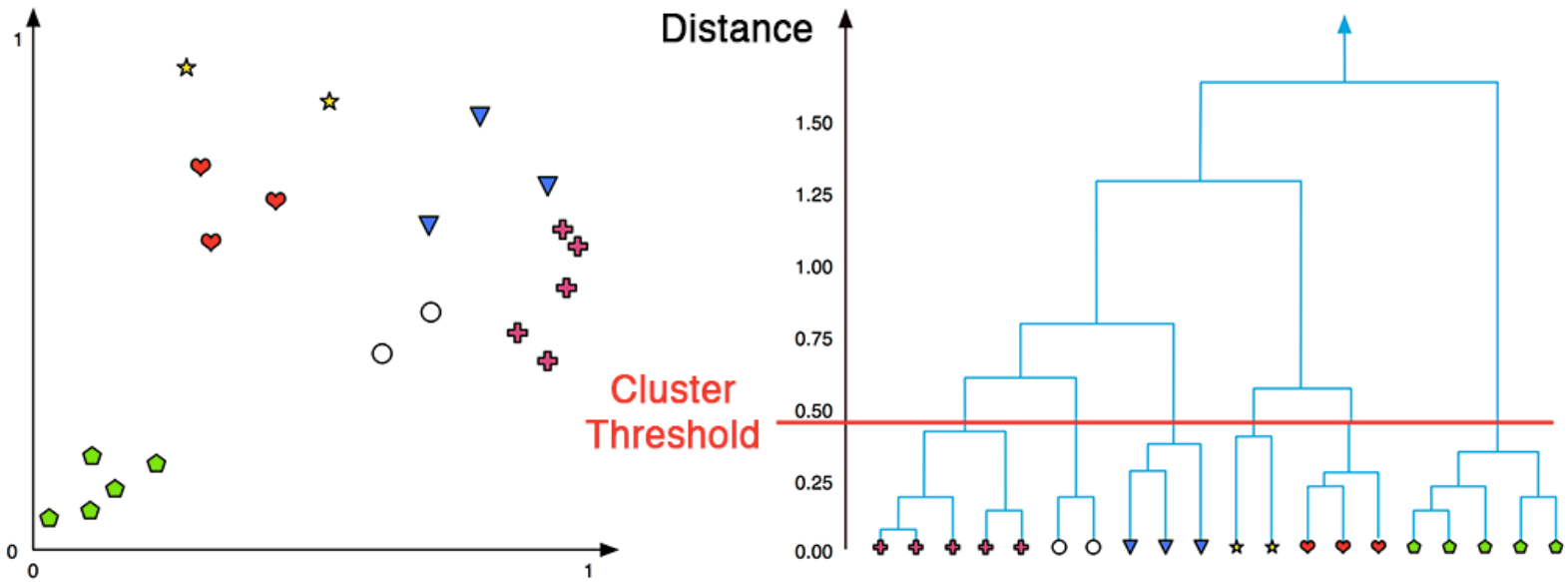


Clustering II

- K-means algorithm (K fixed):



- Hierarchical clustering (variable K):

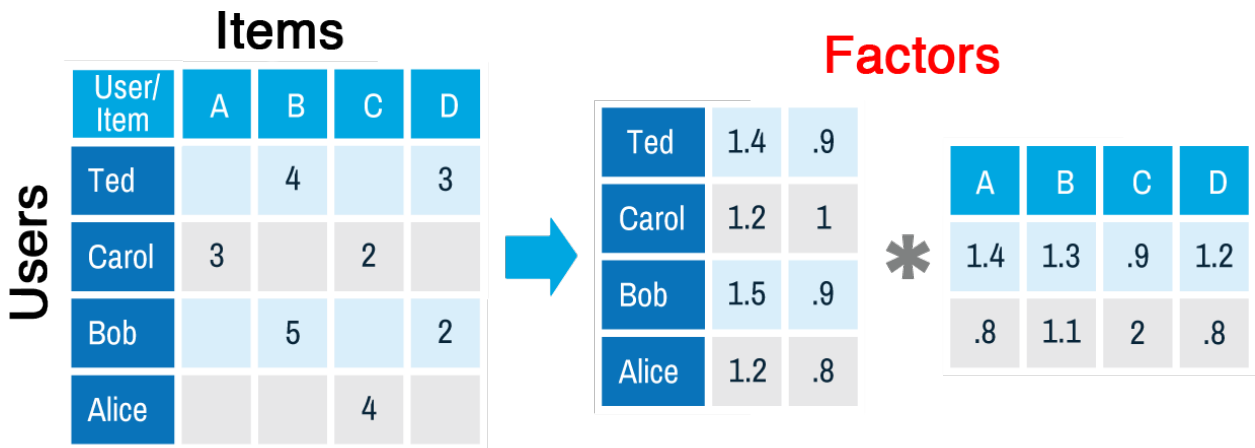


Factorization and Recommendation

- 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 $\boldsymbol{X}$, $\boldsymbol{X}_{i,:} = \boldsymbol{x}^{(i)}$, be a rating matrix

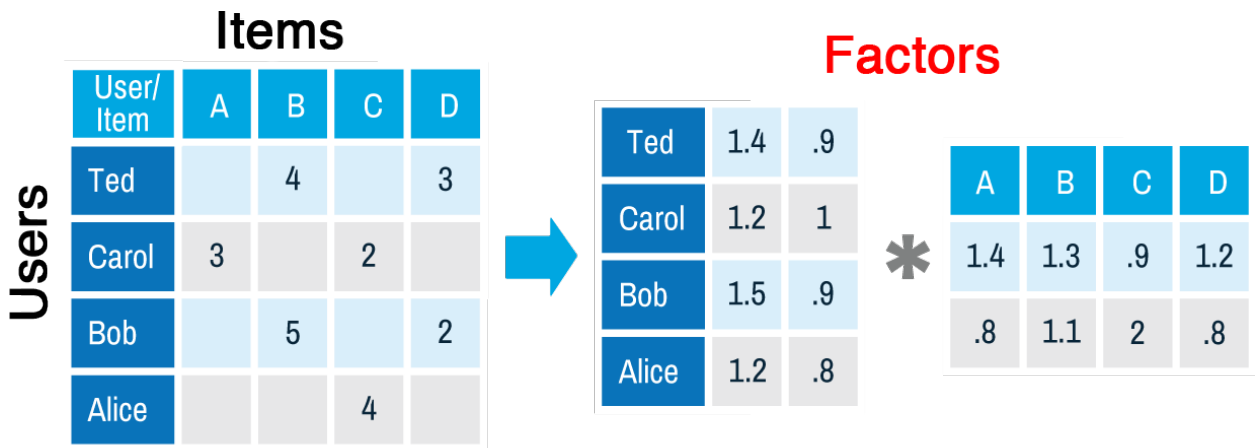


- Non-negative matrix factorization (NMF) [11, 12]:

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

Factorization and Recommendation

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



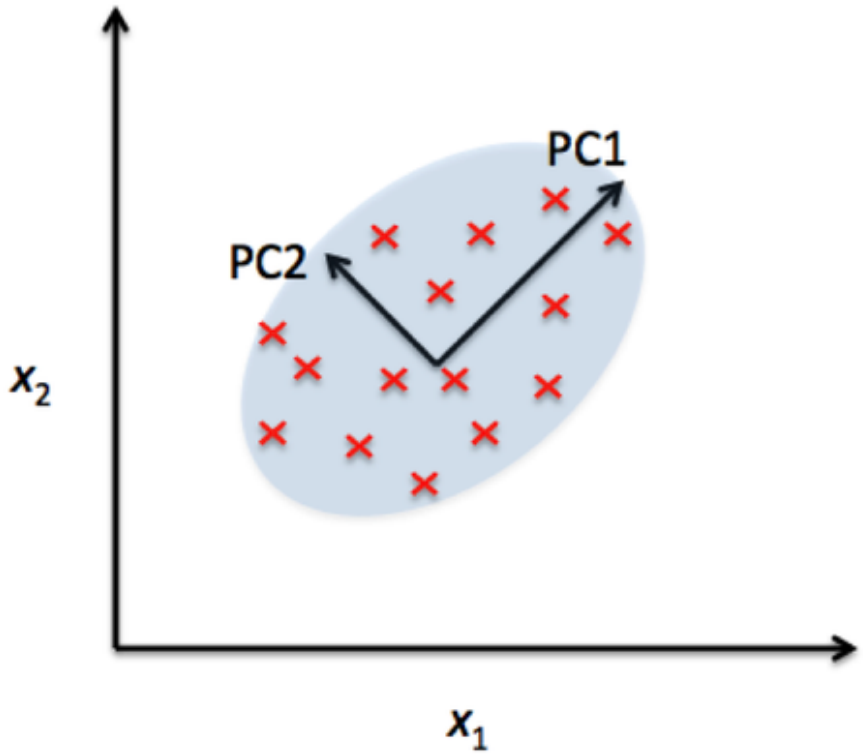
- Non-negative matrix factorization (NMF) [11, 12]:

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

- $\boldsymbol{X}^* = \boldsymbol{W}^* \boldsymbol{H}^*$ 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



Predictive Learning

- Goal: to learn a model that is able to "fill in the blanks"

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

Predictive Learning

- Goal: to learn a model that is able to "fill in the blanks"
- Links unsupervised tasks with supervised models

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

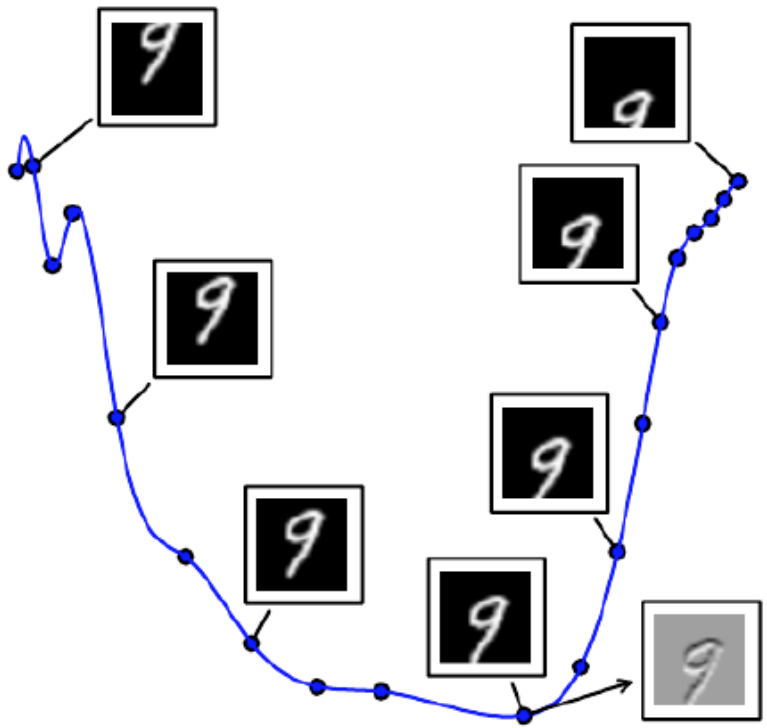
Predictive Learning

- Goal: to learn a model that is able to "fill in the blanks"
- 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

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

Manifold Learning

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

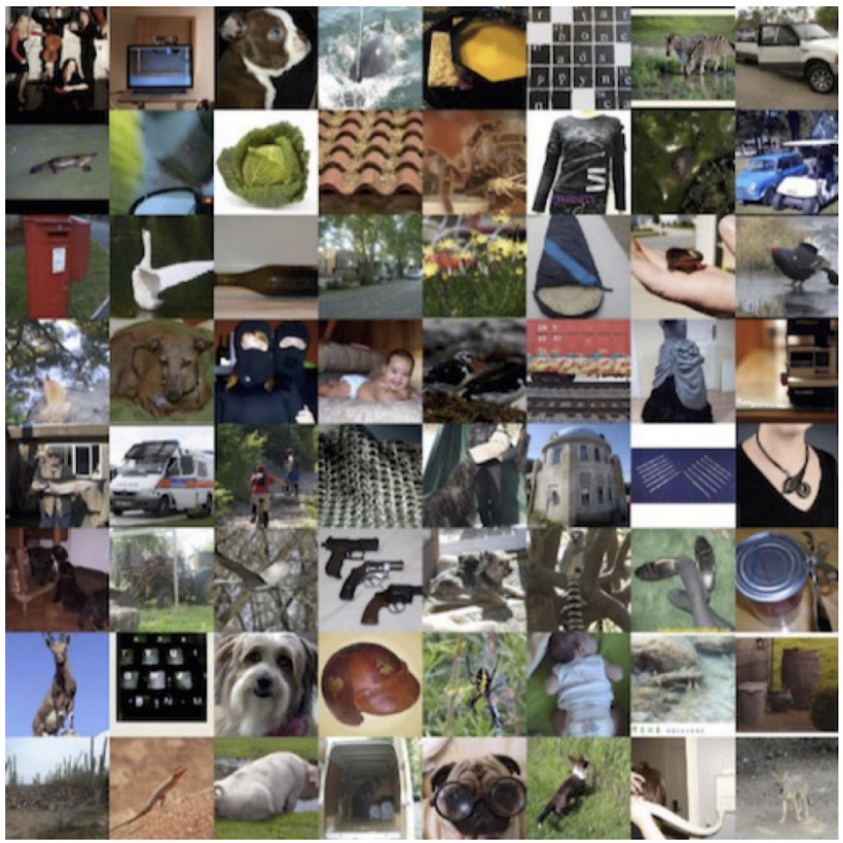


Data Synthesis/Generation I

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

Data Synthesis/Generation I

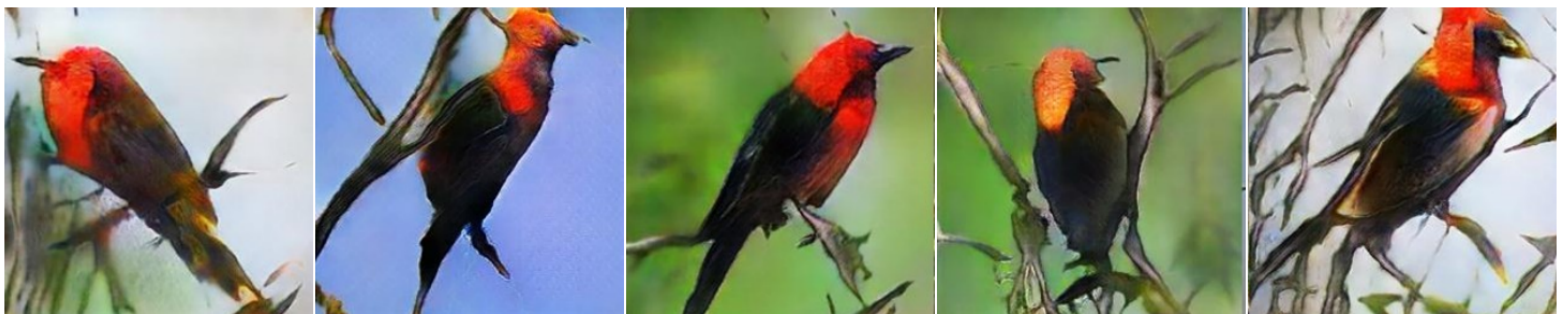
- Goal: to generate new samples of $\mathbf{x}$
- *Generative adversarial networks* (GANs)



Data Synthesis/Generation II

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

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



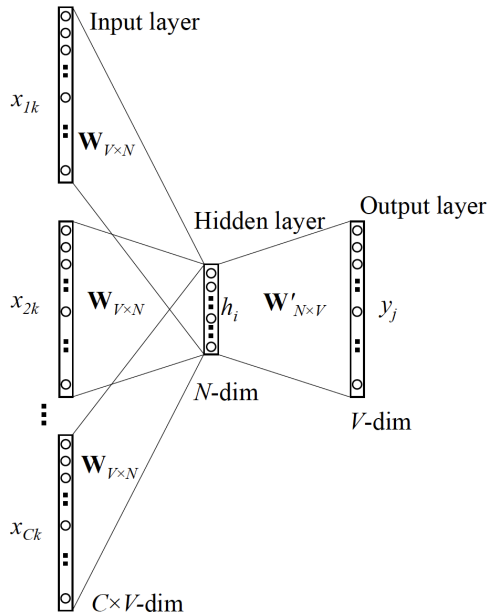
Outline

Self-Supervised Learning

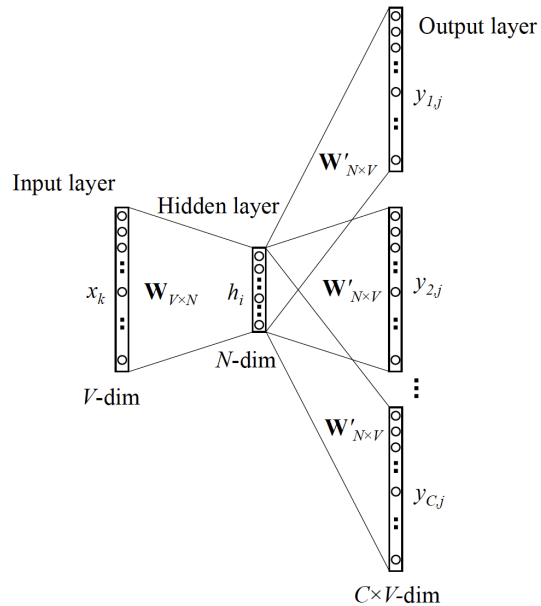
- Goal: to learn a model for blank filling

Self-Supervised Learning

- Goal: to learn a model for blank filling
- E.g., word2vec [16, 15]: "*... the cat sat* ***on****...*"



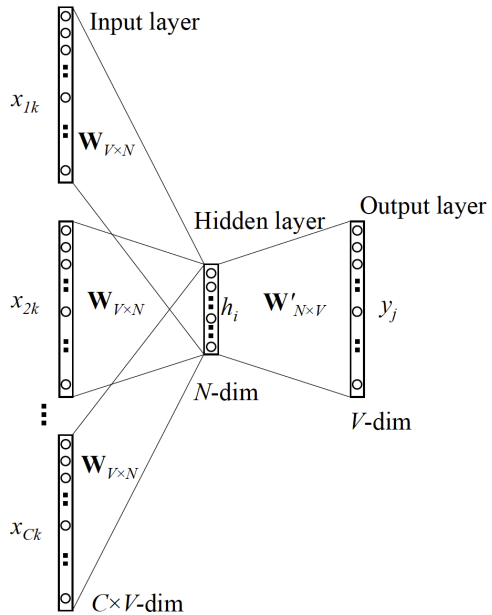
CBOW



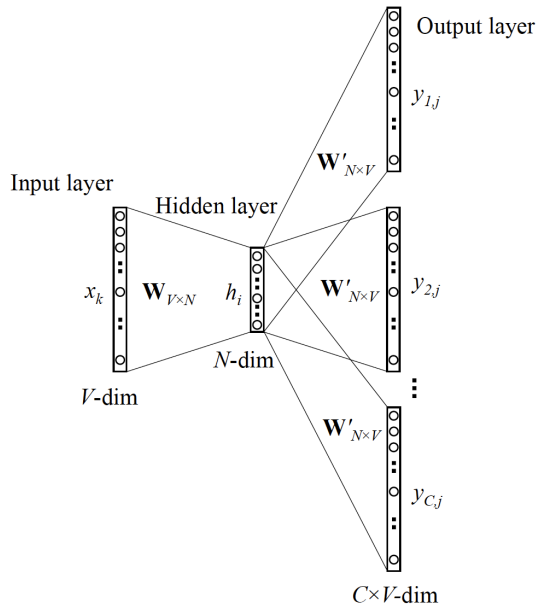
Skip Gram

Self-Supervised Learning

- Goal: to learn a model for blank filling
- E.g., word2vec [16, 15]: "*... 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

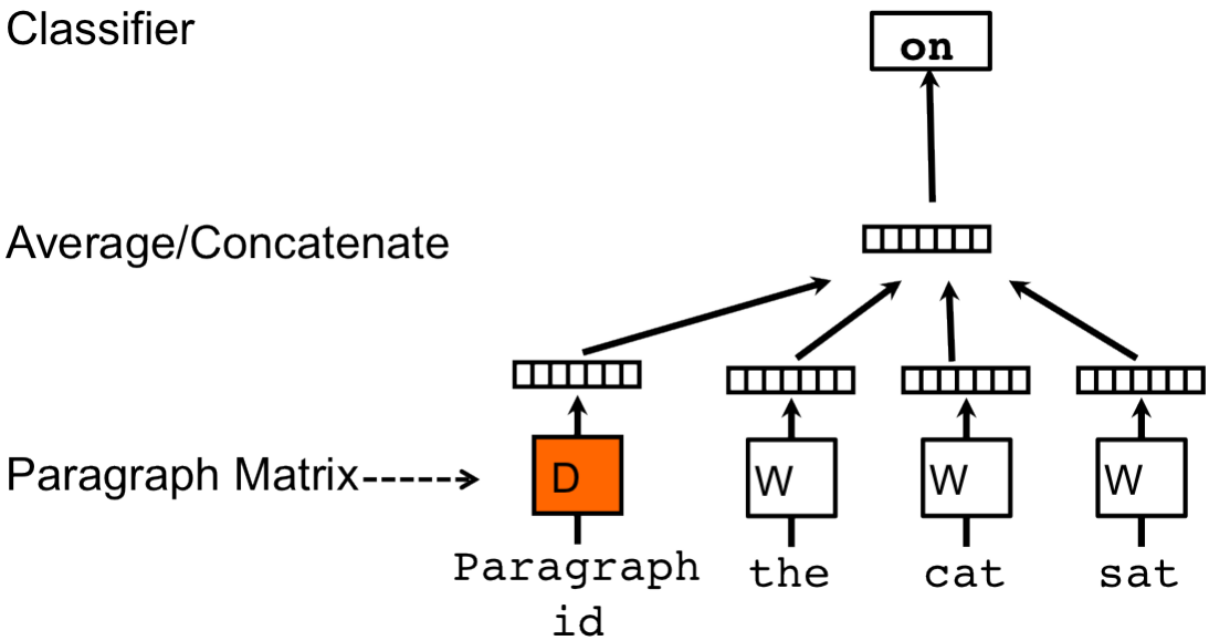
- How to encode a document?
  - Bag of words (TF-IDF), average word2vec, etc.

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"
- Predictive 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"
- Predictive learning for docs?
- Doc2vec [9]: to capture the ***context*** not explained by words



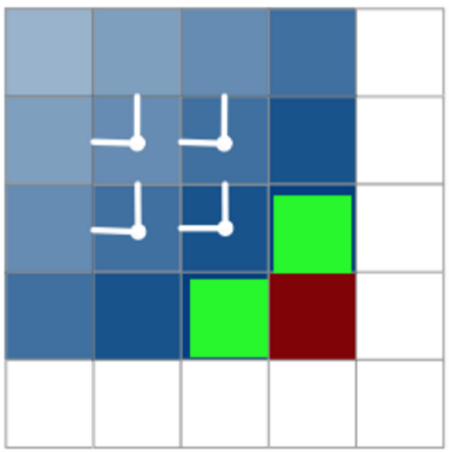
Filling Images

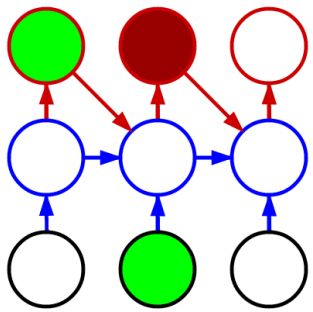
- How?

Filling Images

- How?

- PixelRNN [24], PixelCNN [23]





More

- Predicting the future by watching unlabeled videos [13, 6, 26]:

More

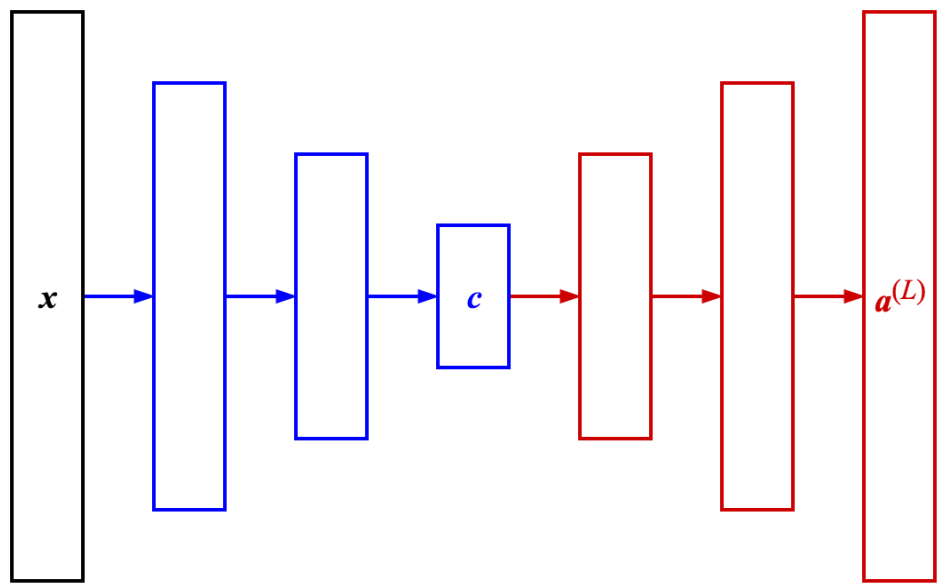
- Predicting the future by watching unlabeled videos [13, 6, 26]:

- Better representations/predictions without the need for labels

Outline

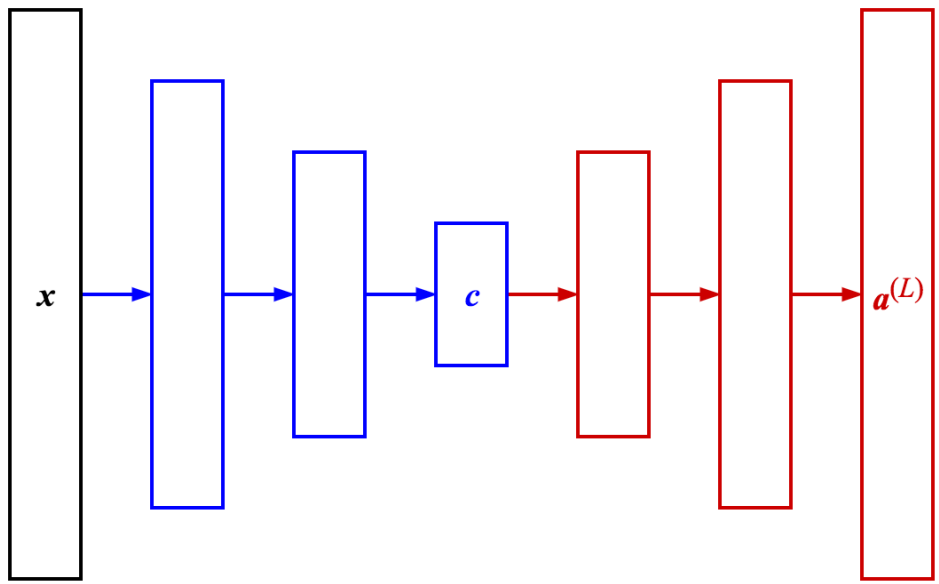
Autoencoders I

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



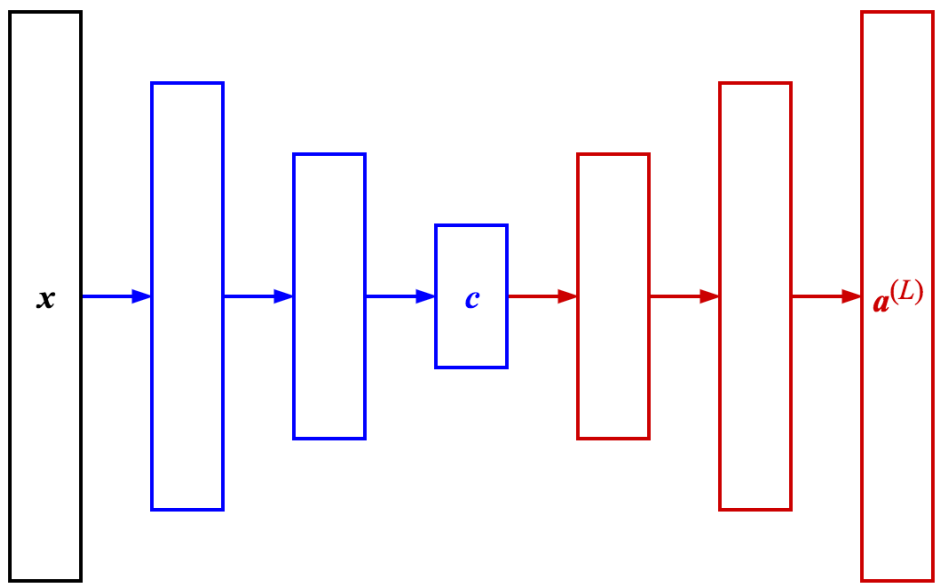
Autoencoders I

- ***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: $\arg\min_{\Theta} -\log \mathrm{P}(\mathbb{X} \mid \Theta) = \arg\min_{\Theta} -\sum_n \log \mathrm{P}(\boldsymbol{x}^{(n)} \mid \Theta)$



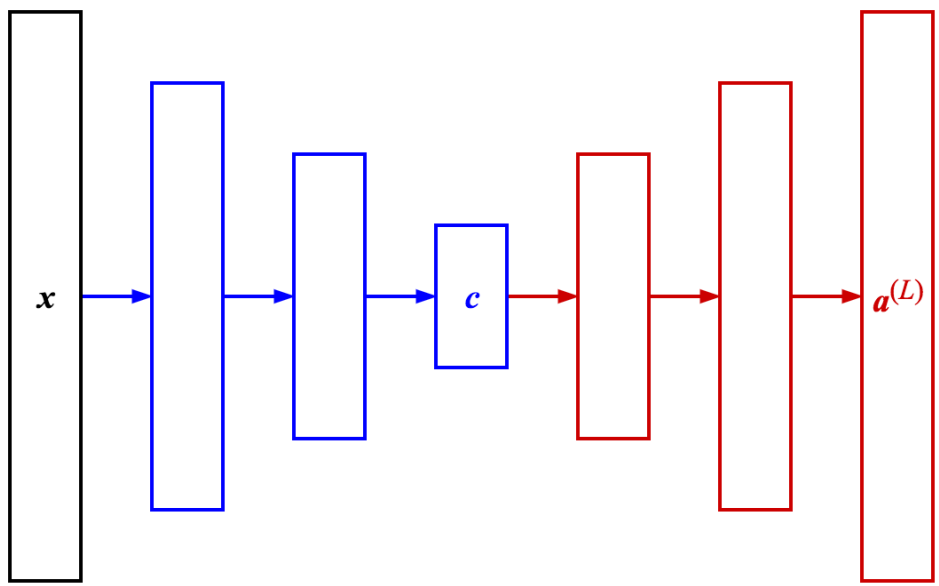
Autoencoders I

- ***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: $\arg\min_\Theta -\log \mathrm{P}(\mathbb{X} \mid \Theta) = \arg\min_\Theta -\sum_n \log \mathrm{P}(\boldsymbol{x}^{(n)} \mid \Theta)$
- Sigmoid output units $a_j^{(L)} = \hat{\rho}_j$ for $\mathrm{x}_j \sim \mathrm{Bernoulli}(\rho_j)$
  - $\mathrm{P}(x_j^{(n)} \mid \Theta) = (a_j^{(L)})^{x_j^{(n)}} (1 - a_j^{(L)})^{(1-x_j^{(n)})}$



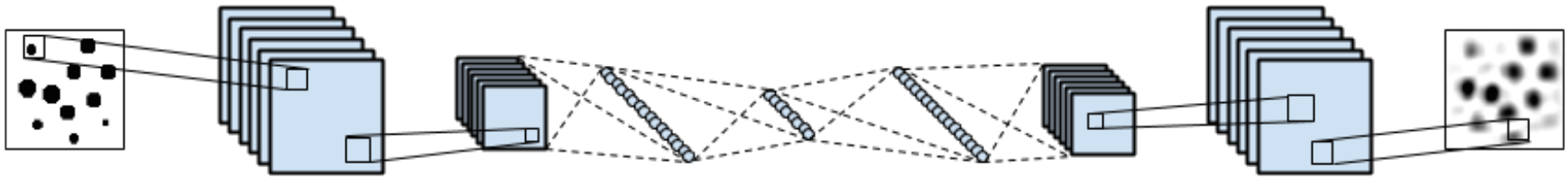
Autoencoders I

- ***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: $\arg\min_\Theta -\log \mathrm{P}(\mathbb{X} \mid \Theta) = \arg\min_\Theta -\sum_n \log \mathrm{P}(\boldsymbol{x}^{(n)} \mid \Theta)$
- Sigmoid output units $a_j^{(L)} = \hat{\rho}_j$ for $\mathrm{x}_j \sim \text{Bernoulli}(\rho_j)$
  - $\mathrm{P}(x_j^{(n)} \mid \Theta) = (a_j^{(L)})^{x_j^{(n)}} (1 - a_j^{(L)})^{(1-x_j^{(n)})}$
- Linear output units $\boldsymbol{a}^{(L)} = \boldsymbol{z}^{(L)} = \hat{\mu}$ for $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$
  - $-\log \mathrm{P}(\boldsymbol{x}^{(n)} \mid \Theta) = \|\boldsymbol{x}^{(n)} - \boldsymbol{a}^{(L)}\|^2$



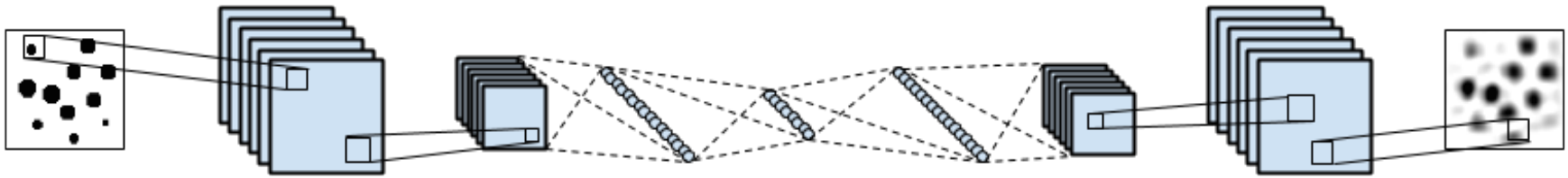
Convolutional Autoencoders

- Convolution + deconvolution layers:



Convolutional Autoencoders

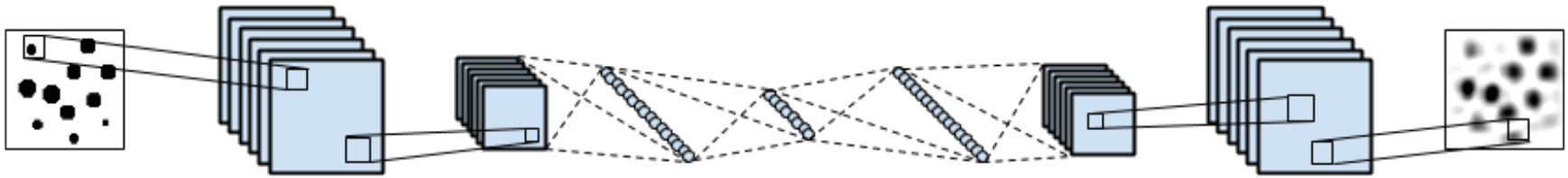
- Convolution + deconvolution layers:



- Decoder is a simplified DeconvNet [27] trained from scratch:

Convolutional Autoencoders

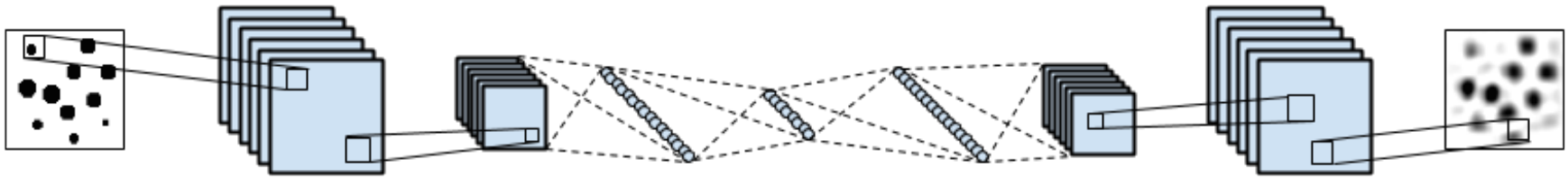
- Convolution + deconvolution layers:



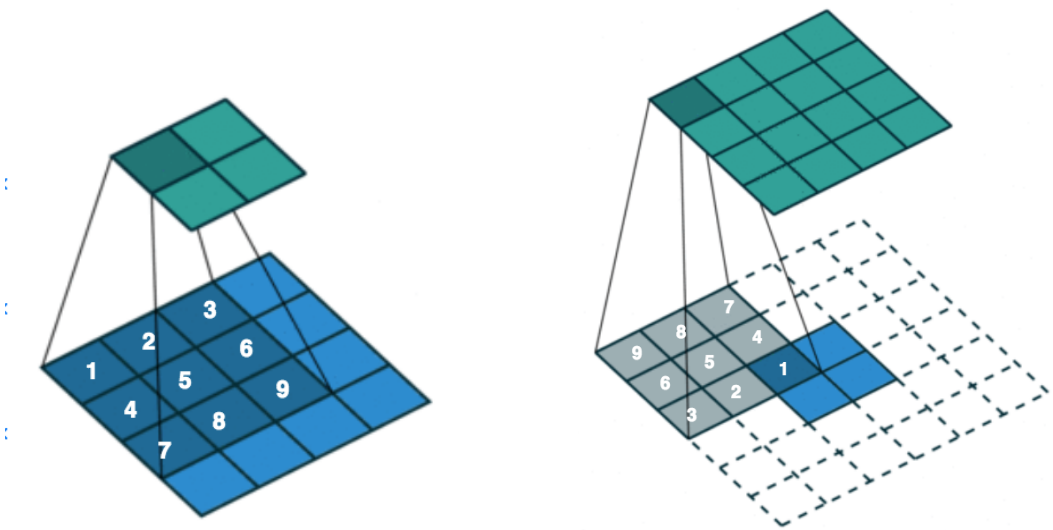
- Decoder is a simplified DeconvNet [27] trained from scratch:
  - Uppooling $\rightarrow$ upsampling (no need to remember max positions)

Convolutional Autoencoders

- Convolution + deconvolution layers:

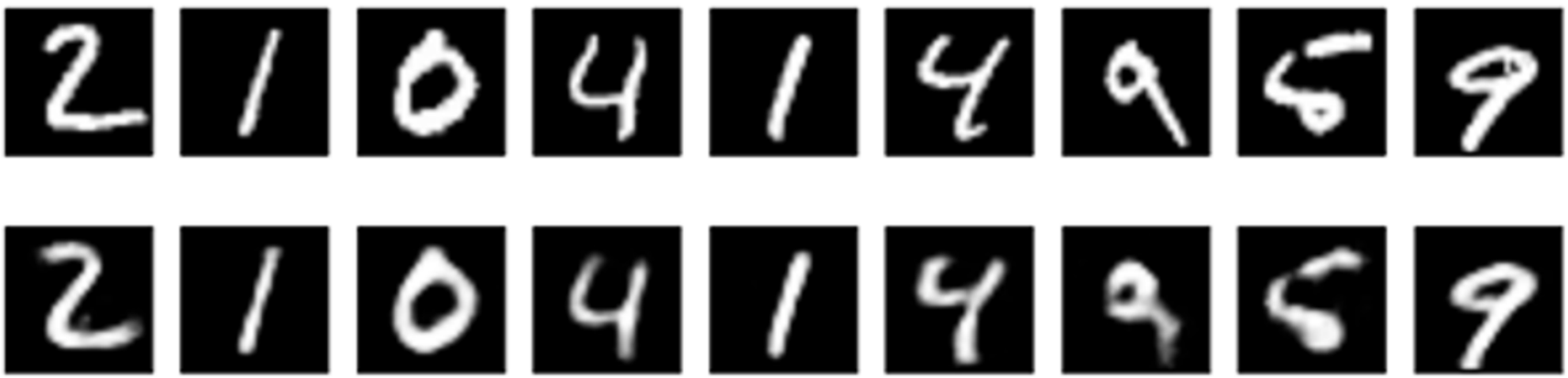


- Decoder is a simplified DeconvNet [27] trained from scratch:
  - Uppooling $\rightarrow$ upsampling (no need to remember max positions)
  - Deconvolution $\rightarrow$ convolution



Codes & Reconstructed $\mathbf{x}$

- A 32-bit code can roughly represents a 32×32 (1024 dimensional) MNIST image

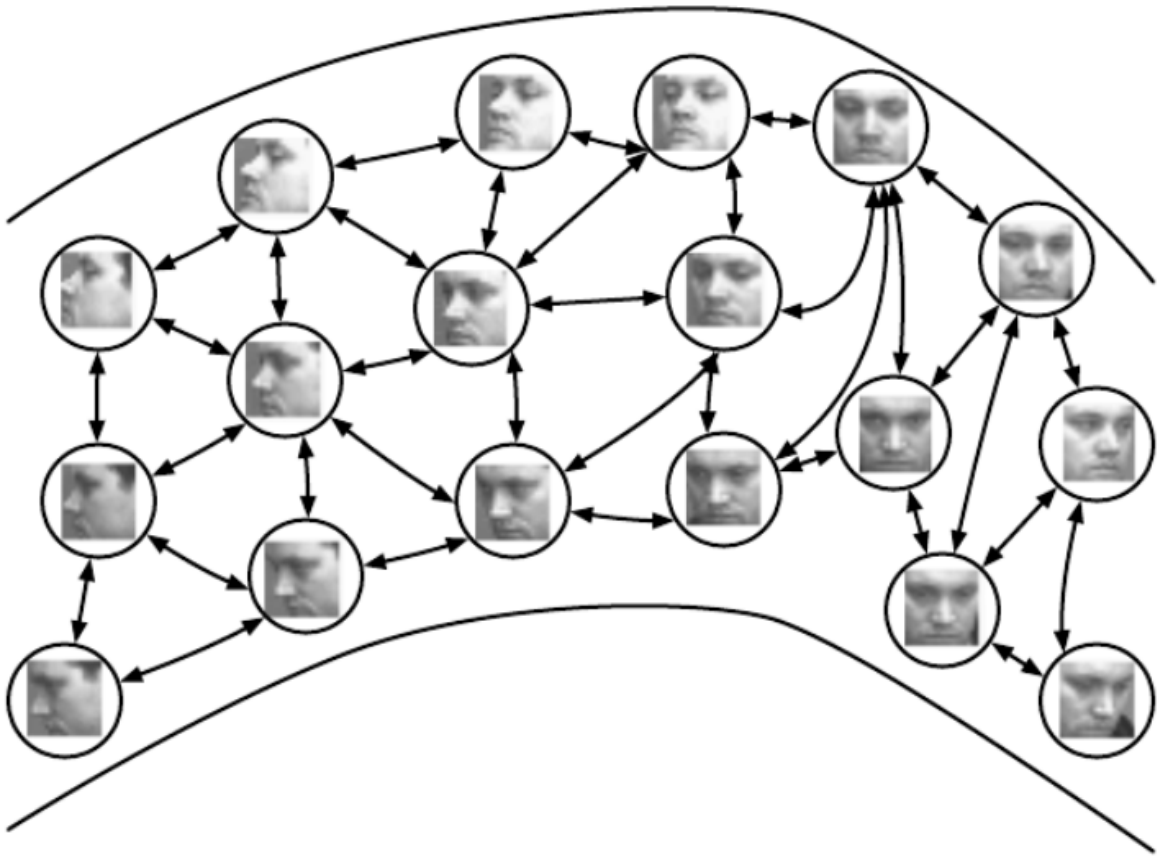


Manifolds I

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

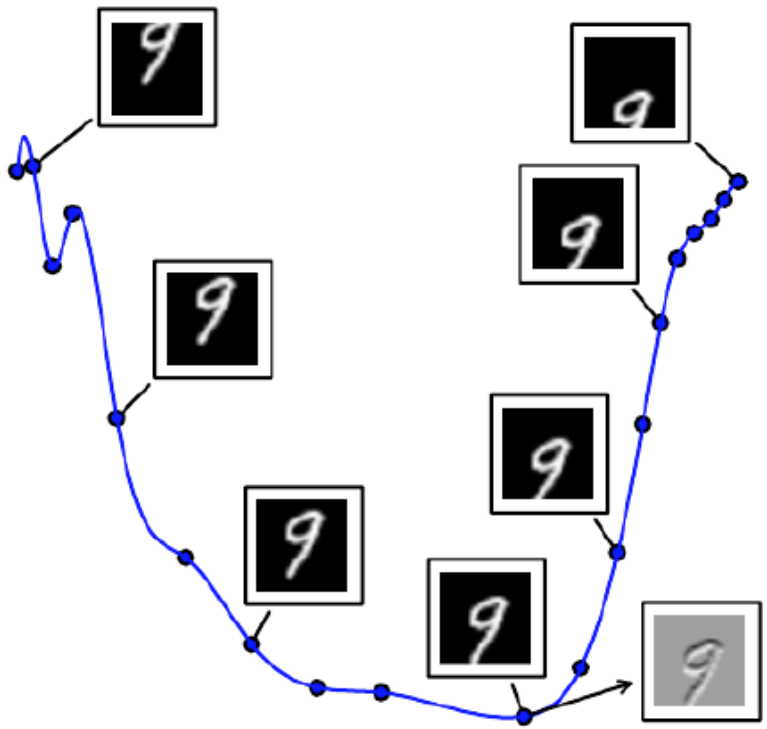
Manifolds I

- 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 $\boldsymbol{x}$ on a manifold, we have its ***tangent space*** spanned by ***tangent vectors***
  - Local directions specify how one can change $\boldsymbol{x}$ 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 [19]: regularizes the code $\mathbf{c}$ such that it is invariant to local changes of $\mathbf{x}$:

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

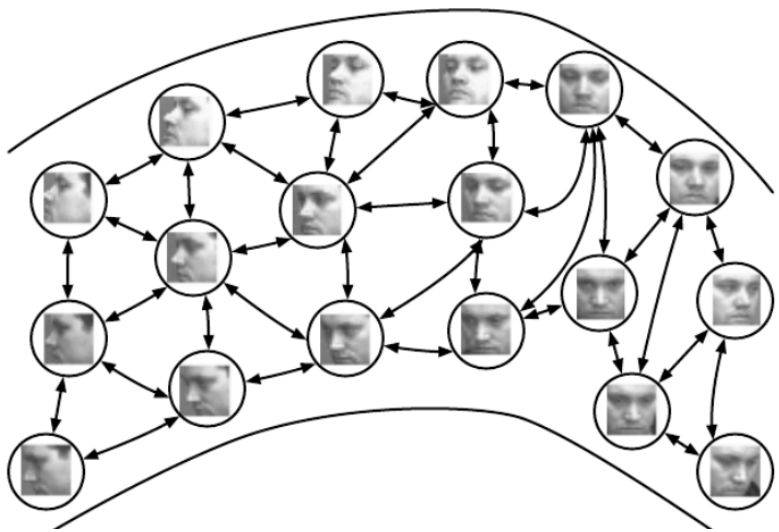
  - $\partial \boldsymbol{c}^{(n)} / \partial \boldsymbol{x}^{(n)}$ is a Jacobian matrix

Learning Manifolds I

- How to make $\mathbf{c}$ produced by autoencoders denote a ***coordinate*** of a dimensional manifold?
- Contractive autoencoder [19]: regularizes the code $\mathbf{c}$ such that it is invariant to local changes of $\mathbf{x}$:

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

  - $\partial \boldsymbol{c}^{(n)} / \partial \boldsymbol{x}^{(n)}$ is a Jacobian matrix
- Prevents the encoder from changing $\mathbf{c}$ easily for changes of $\mathbf{x}$
  - Except for the changes of $\mathbf{x}$ that follows tangent vectors



Learning Manifolds II

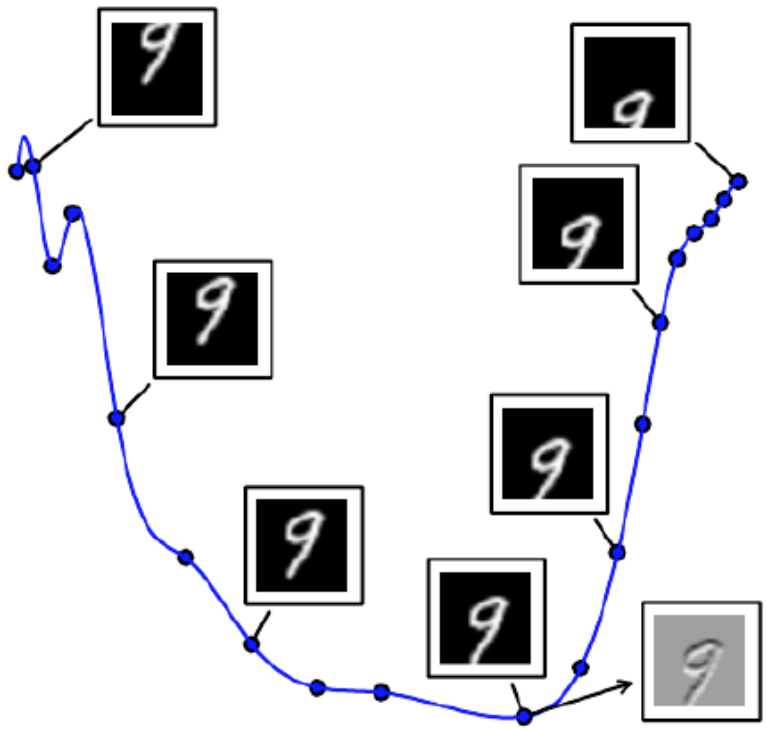
- In practice, it is easier to train a denoising autoencoder [25]:

Learning Manifolds II

- In practice, it is easier to train a denoising autoencoder [25]:
  - Encoder: to encode $\mathbf{x}$ ***with*** random noises
  - Decoder: to reconstruct $\mathbf{x}$ ***without*** noises

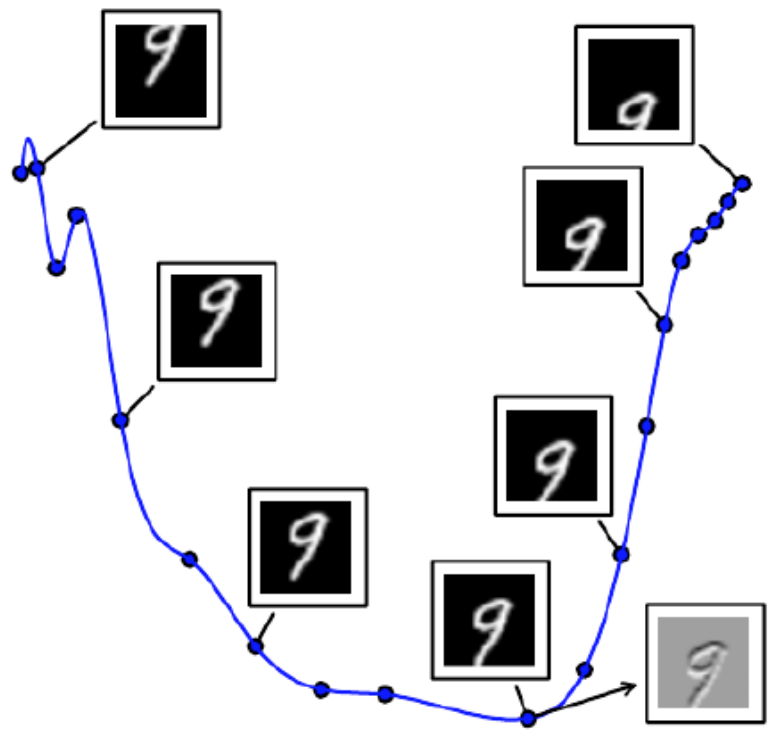
Getting Tangent Vectors I

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



Getting Tangent Vectors II

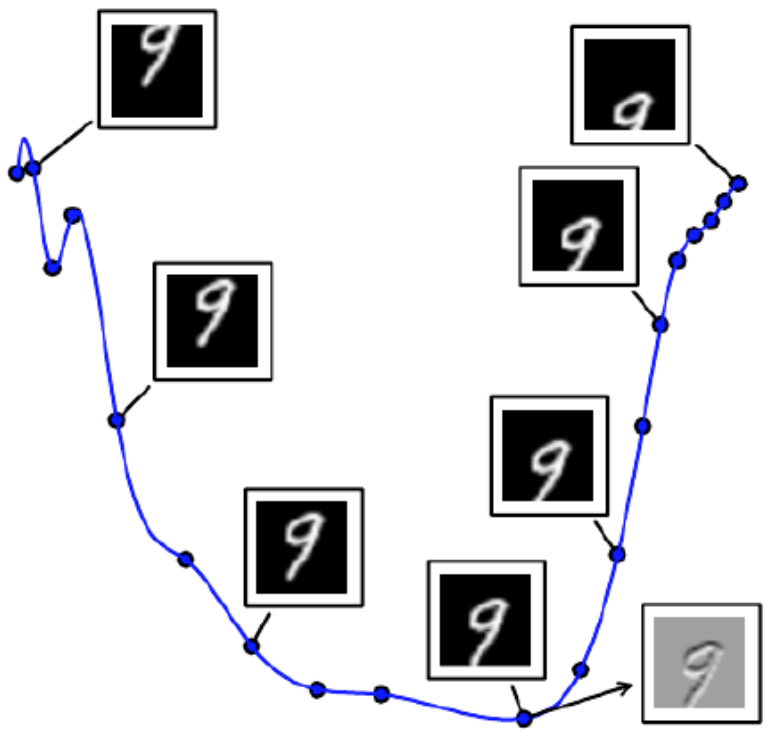
- Recall: directions in the input space that ***changes c most*** should be tangent vectors



Getting Tangent Vectors II

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

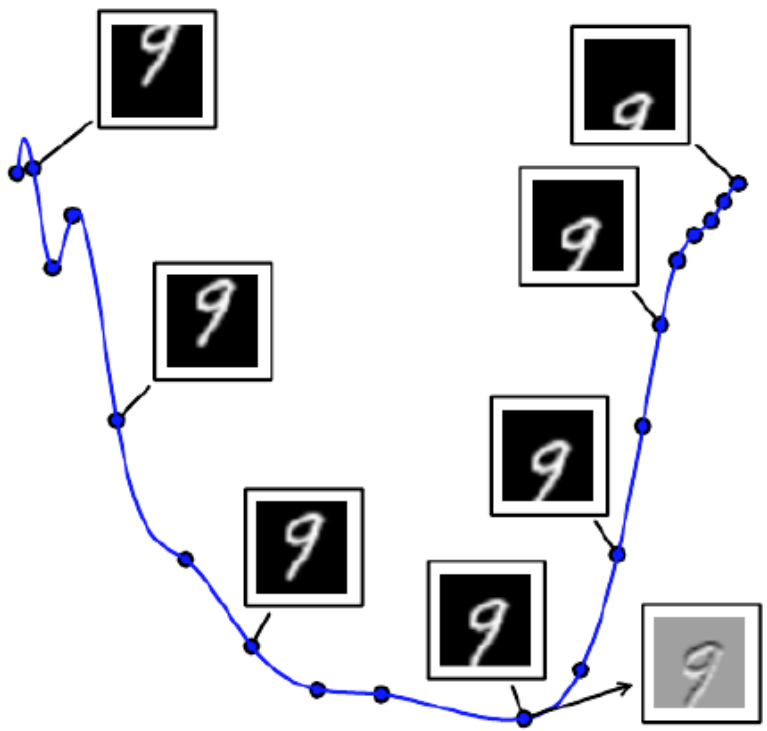
- Given a point $\boldsymbol{x}$, let $\boldsymbol{c}$ be the code of $\boldsymbol{x}$ and $\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{x}}$ be the Jacobian matrix of $\boldsymbol{c}$ at $\boldsymbol{x}$



Getting Tangent Vectors II

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

- Given a point $\boldsymbol{x}$, let $\boldsymbol{c}$ be the code of $\boldsymbol{x}$ and $\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{x}}$ be the Jacobian matrix of $\boldsymbol{c}$ at $\boldsymbol{x}$
  - $\boldsymbol{J}(\boldsymbol{x})$ summarizes how $\boldsymbol{c}$ changes in terms of $\boldsymbol{x}$

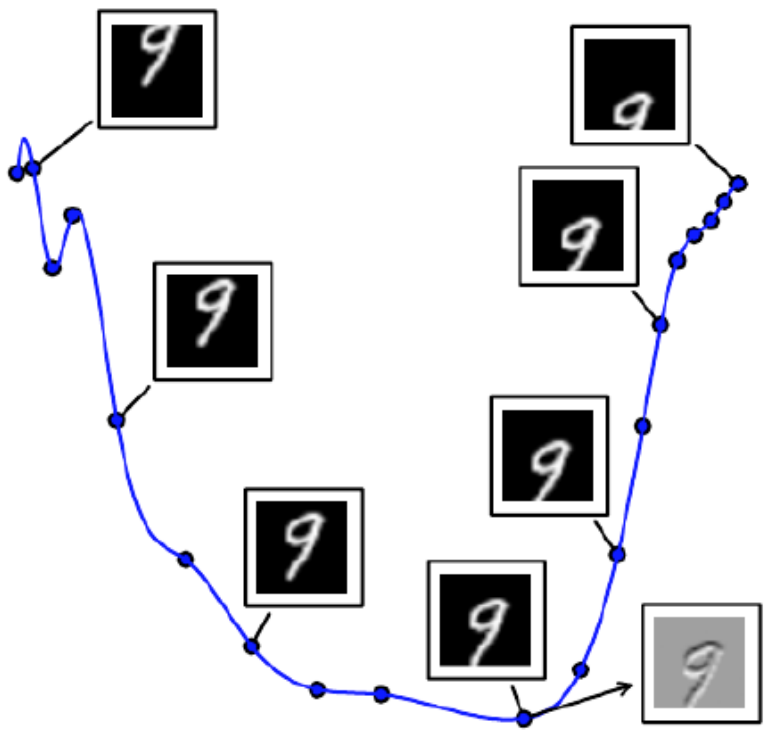


Getting Tangent Vectors II

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

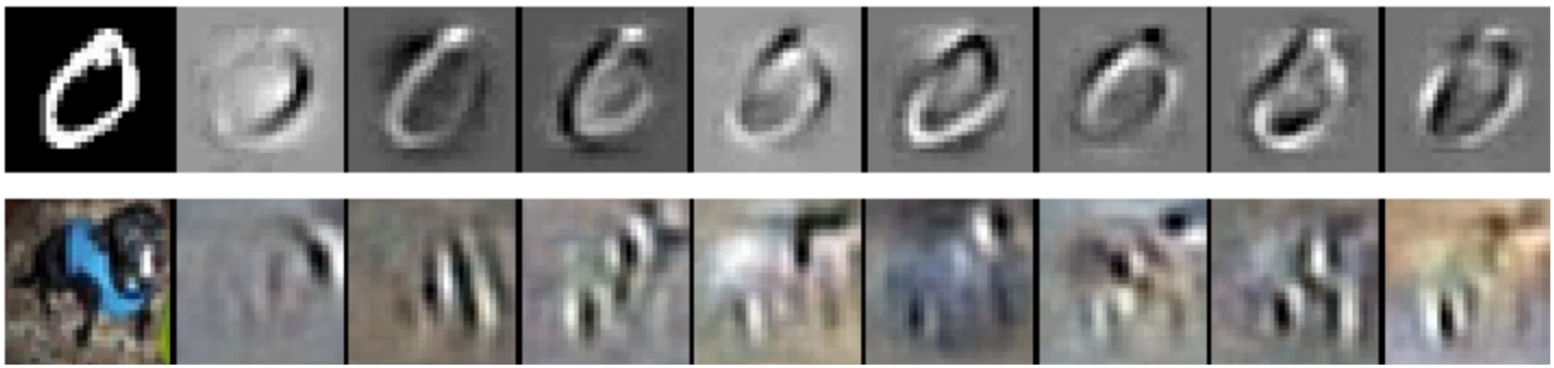
- Given a point $\boldsymbol{x}$, let $\boldsymbol{c}$ be the code of $\boldsymbol{x}$ and $\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{x}}$ be the Jacobian matrix of $\boldsymbol{c}$ at $\boldsymbol{x}$
  - $\boldsymbol{J}(\boldsymbol{x})$ summarizes how $\boldsymbol{c}$ changes in terms of $\boldsymbol{x}$

1. Decompose $\boldsymbol{J}(\boldsymbol{x})$ using SVD such that $\boldsymbol{J}(\boldsymbol{x}) = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$
2. Let tangent vectors be ***rows of V corresponding to the largest singular values in D***



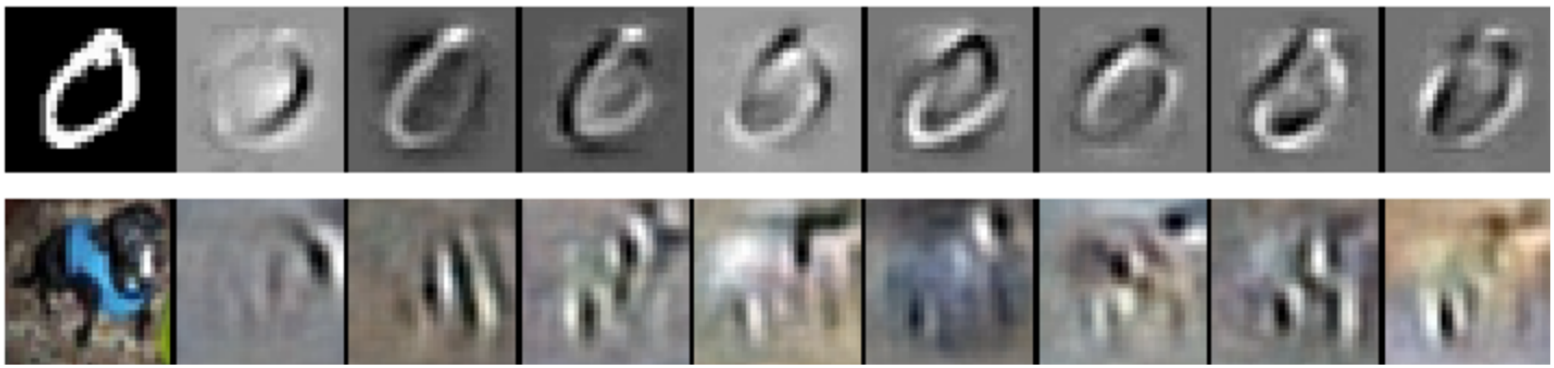
Getting Tangent Vectors III

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



Getting Tangent Vectors III

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

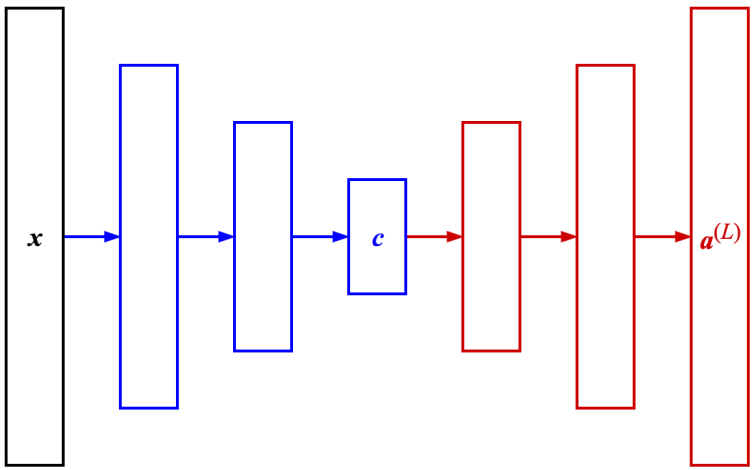


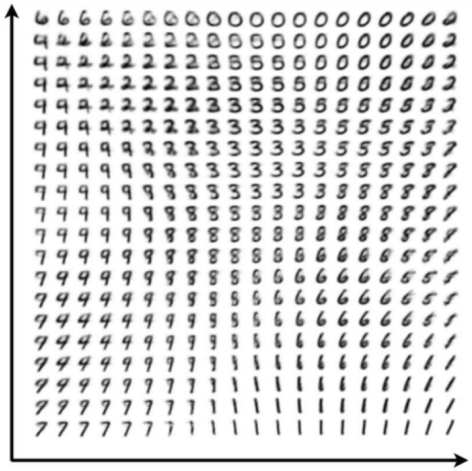
- Can be used by Tangent Prop [22]:
- Let $\{\boldsymbol{v}^{(i,j)}\}_j$ be tangent vectors of each example $\boldsymbol{x}^{(i)}$
- Trains an NN classifier f with cost penalty: $\Omega[f] = \sum_{i,j} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(i)})^\top \boldsymbol{v}^{(i,j)}$
  - Points in the same manifold share the same label

Outline

Decoder as Data Generator

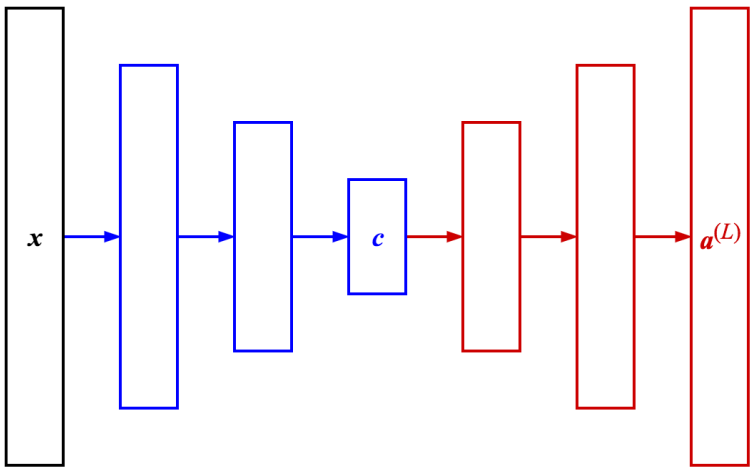
- Decoder of an autoencoder can be used to generate data points even with ***synthetic codes***

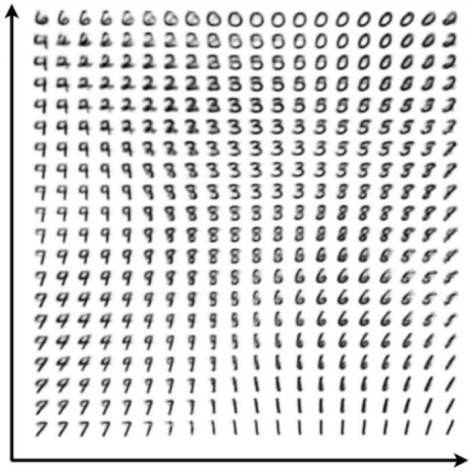




Decoder as Data Generator

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

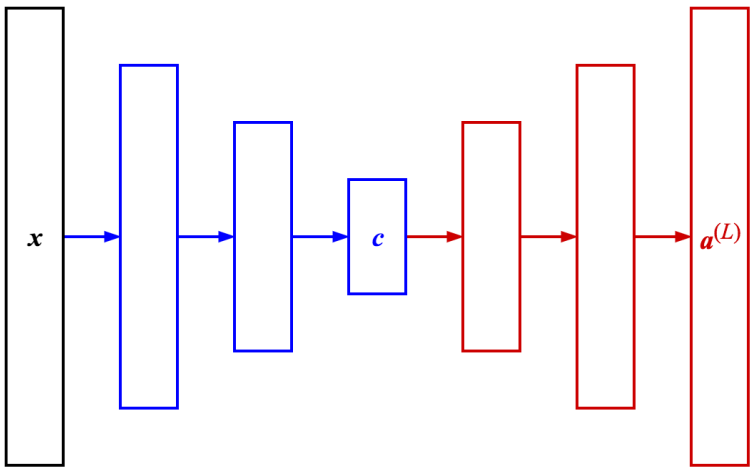


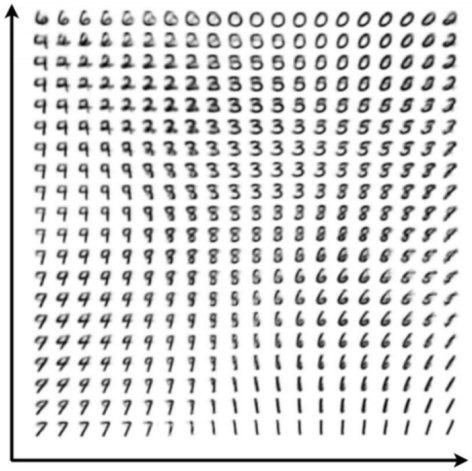


- Problems:
  - Same $\boldsymbol{c}$, same output

Decoder as Data Generator

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

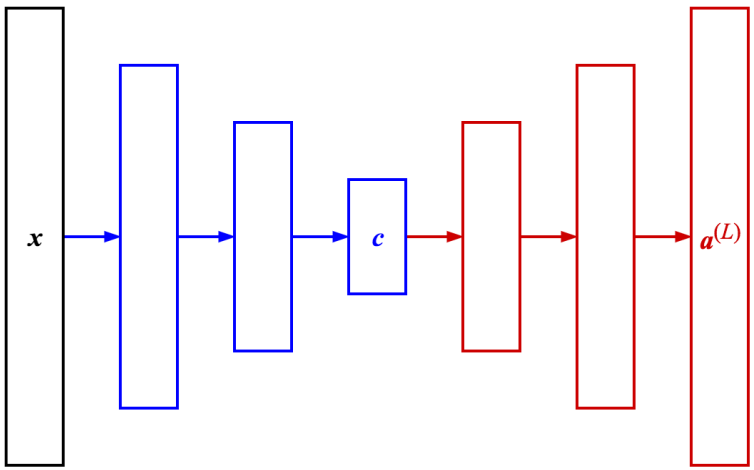


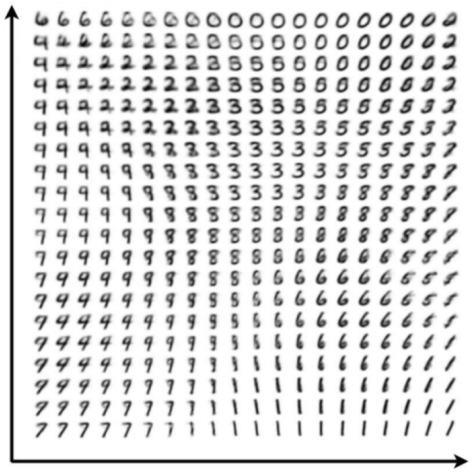


- Problems:
  - Same $\boldsymbol{c}$, same output$\rightarrow$ dropout layers, variational autoencoders [8]

Decoder as Data Generator

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

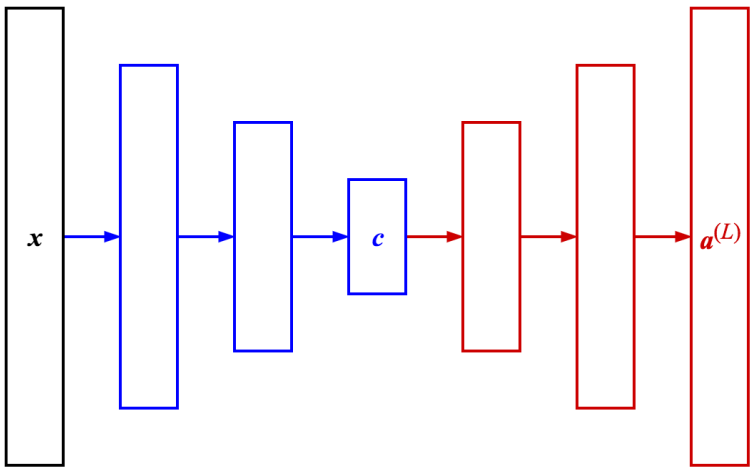


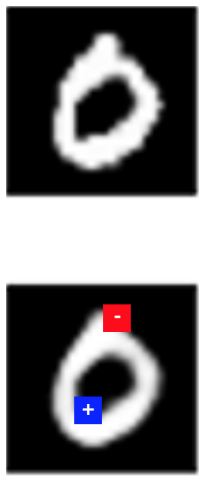


- Problems:
  - Same $\boldsymbol{c}$, same output$\rightarrow$ dropout layers, variational autoencoders [8]
  - ***Blurry images***

Why Blurry Images?

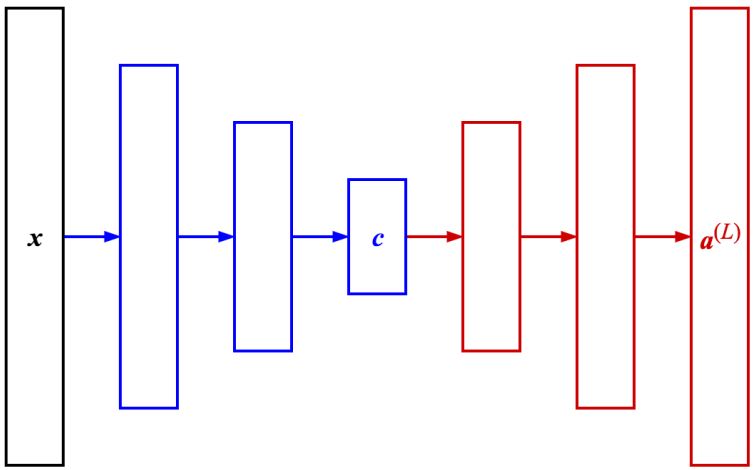
- Cost function: $\arg\min_\Theta -\log \mathrm{P}(\mathbb{X}\,|\,\Theta) = \arg\min_\Theta -\sum_n \log \mathrm{P}(\boldsymbol{x}^{(n)}\,|\,\Theta)$
- Image generation: linear output units $\boldsymbol{a}^{(L)} = \boldsymbol{z}^{(L)} = \hat{\mu}$ for $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$
  - $-\log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

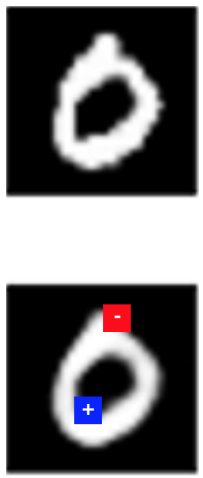




Why Blurry Images?

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

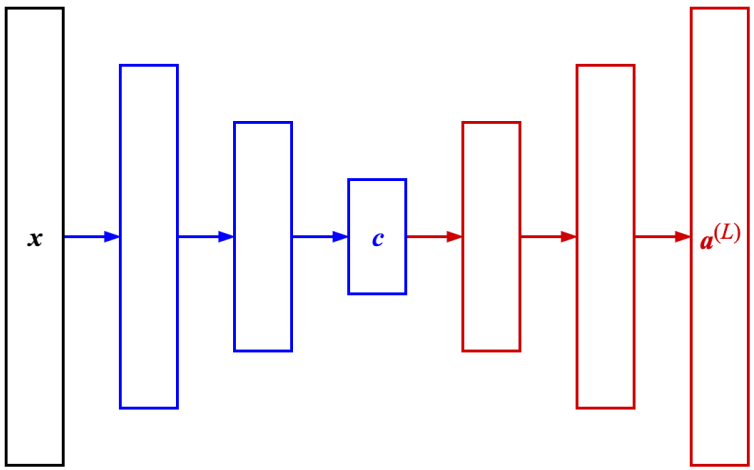


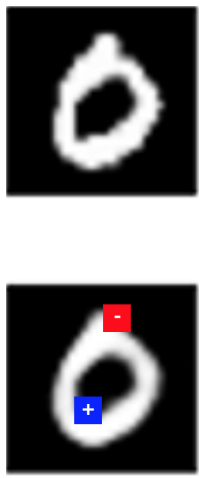


- Better assuming distribution for $\mathbf{x}$?

Why Blurry Images?

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

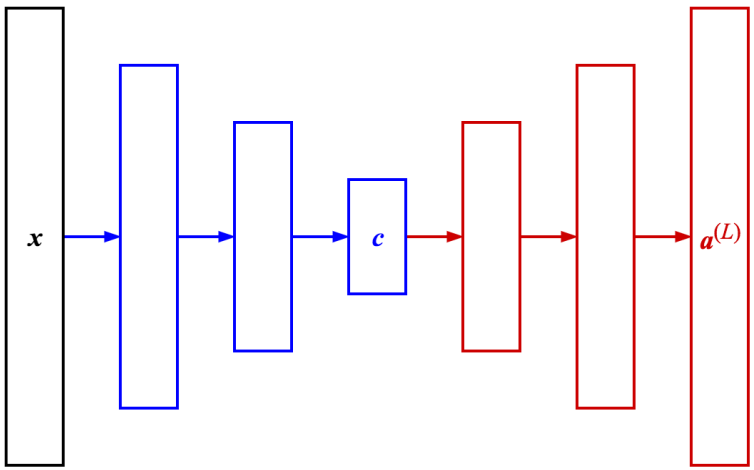


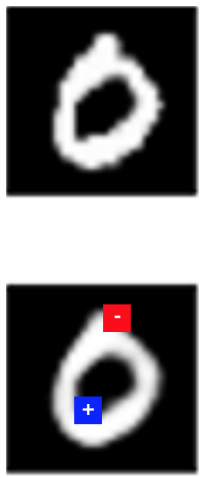


- Better assuming distribution for $\mathbf{x}$? $\mathrm{P}(\mathbf{x})$ may be very complex
- Better "goodness" measure?

Why Blurry Images?

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



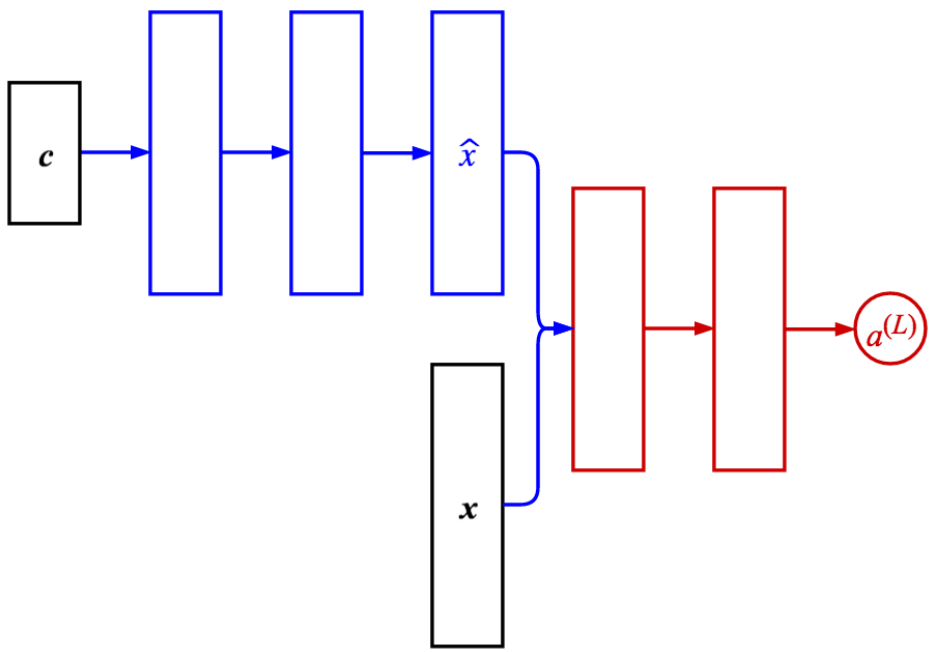


- Better assuming distribution for $\mathbf{x}$? $\mathrm{P}(\mathbf{x})$ may be very complex
- Better "goodness" measure? Why not use an NN to tell if a generated image is of good quality?

Outline

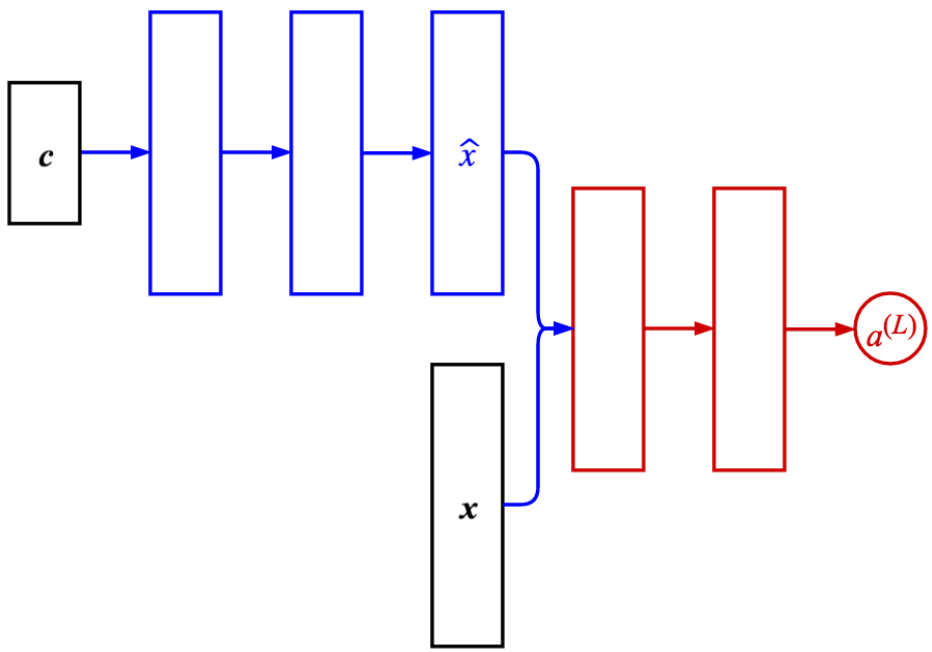
Generative Adversarial Networks (GANs)

- *Generative adversarial network* (*GAN*) [3]:



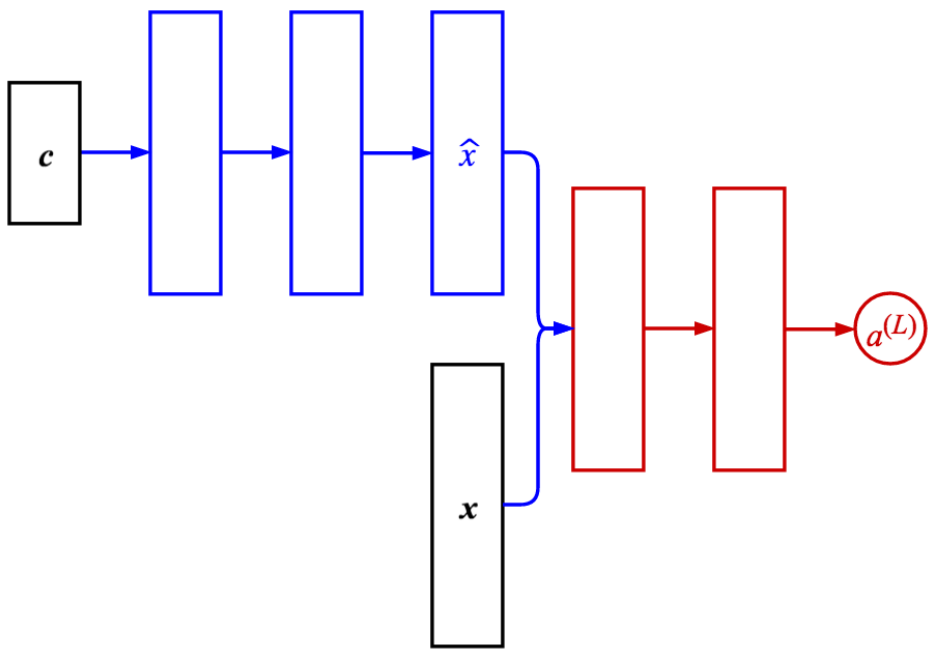
Generative Adversarial Networks (GANs)

- ***Generative adversarial network*** (***GAN***) [3]:
- ***Generator*** g: to generate data points from random codes
  - No need for "encoder" since the task is data synthesis



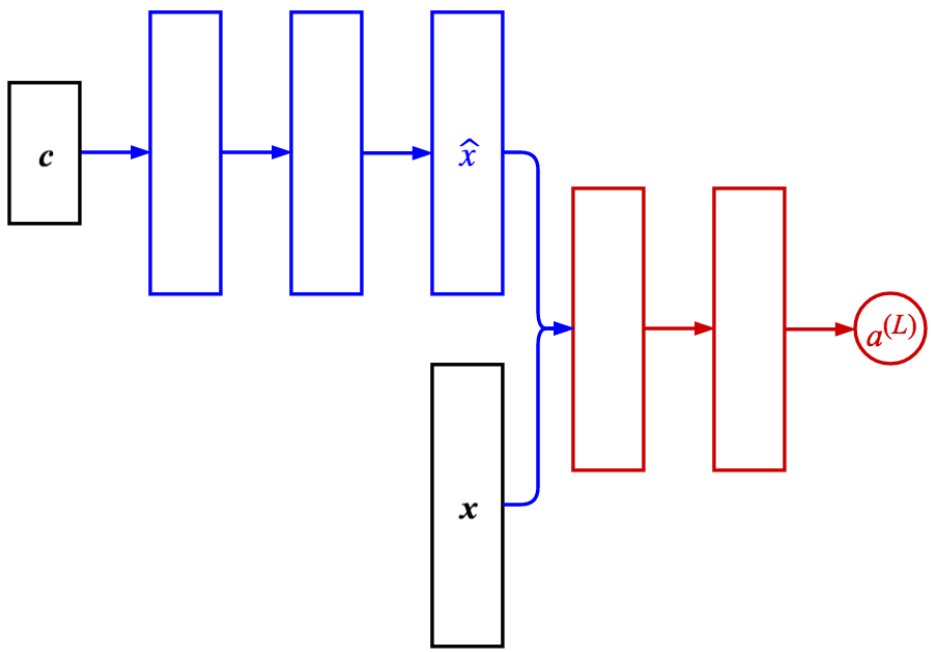
Generative Adversarial Networks (GANs)

- ***Generative adversarial network*** (***GAN***) [3]:
- ***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 $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ are tied
  - A binary classifier with Sigmoid output unit $a^{(L)} = \hat{\rho}$ for $\mathrm{P}(\mathrm{y} = \text{true point} \,|\, \mathbf{x}) \sim \mathrm{Bernoulli}(\rho)$



Generative Adversarial Networks (GANs)

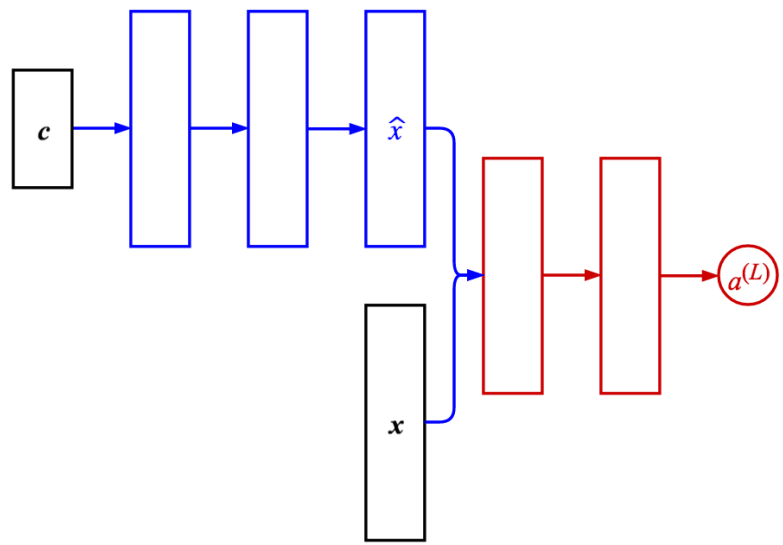
- ***Generative adversarial network*** (***GAN***) [3]:
- ***Generator*** g: to generate data points from random codes
  - No need for "encoder" since the task is data synthesis
- ***Discriminator*** f: to separate generated points from real ones
  - Weights for $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ are tied
  - A binary classifier with Sigmoid output unit $a^{(L)} = \hat{\rho}$ for $\mathrm{P}(\mathrm{y} = \text{true point} \,|\, \mathbf{x}) \sim \mathrm{Bernoulli}(\boldsymbol{\rho})$
- Goal: to train a g that tricks d into believing $g(\boldsymbol{c})$ is real



Cost Function

- Given N real training points and N generated points:

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



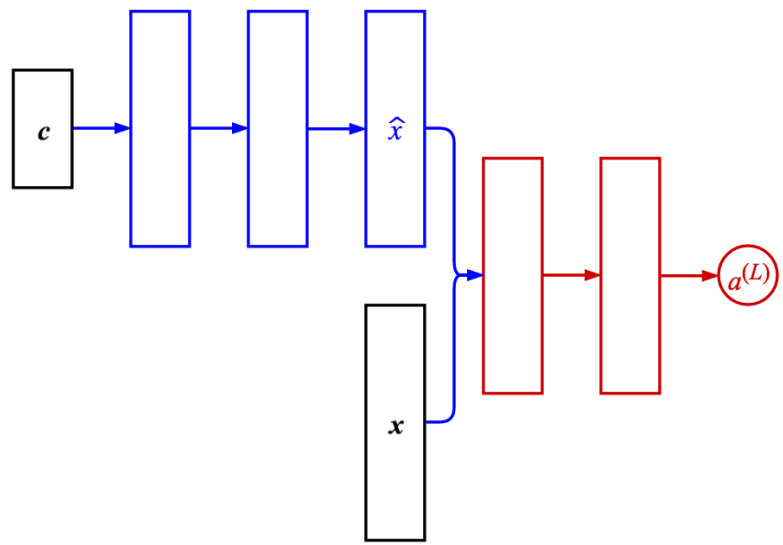
Cost Function

- Given N real training points and N generated points:

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

- Recall that f maximizes the log likelihood

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



Cost Function

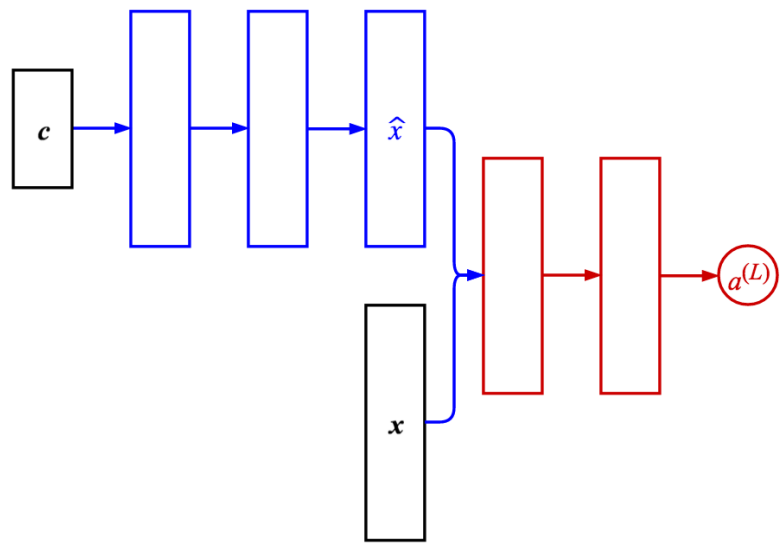
- Given N real training points and N generated points:

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

- Recall that f maximizes the log likelihood

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

- Inner max first, then outer min



Cost Function

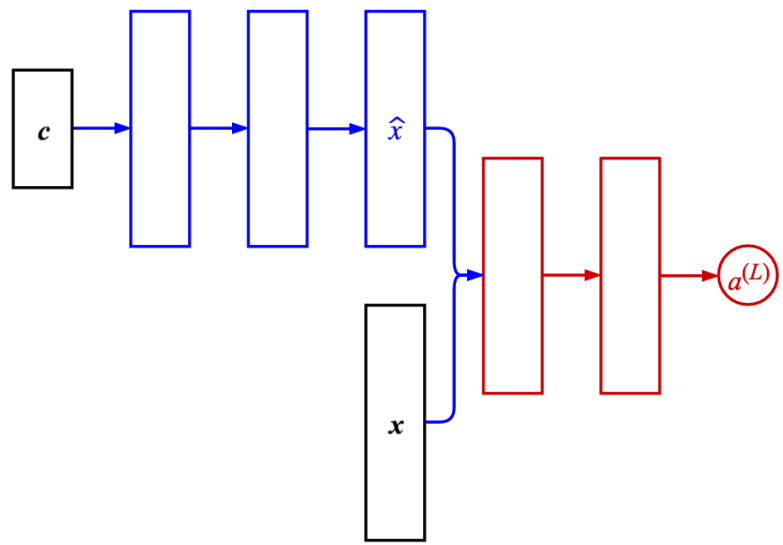
- Given N real training points and N generated points:

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

- Recall that f maximizes the log likelihood

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

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



Training: Alternative SGD

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

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

Training: Alternative SGD

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

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

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

Training: Alternative SGD

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

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. ***Repeat K times (with fixed Θ_g):***
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$
2. ***Execute once (with fixed Θ_f):***
   1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$

Training: Alternative SGD

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

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. ***Repeat K times (with fixed Θ_g):***
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$
2. ***Execute once (with fixed Θ_f):***
   1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$

- Why limiting the steps (K) when updating Θ_f?

Training: Alternative SGD

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

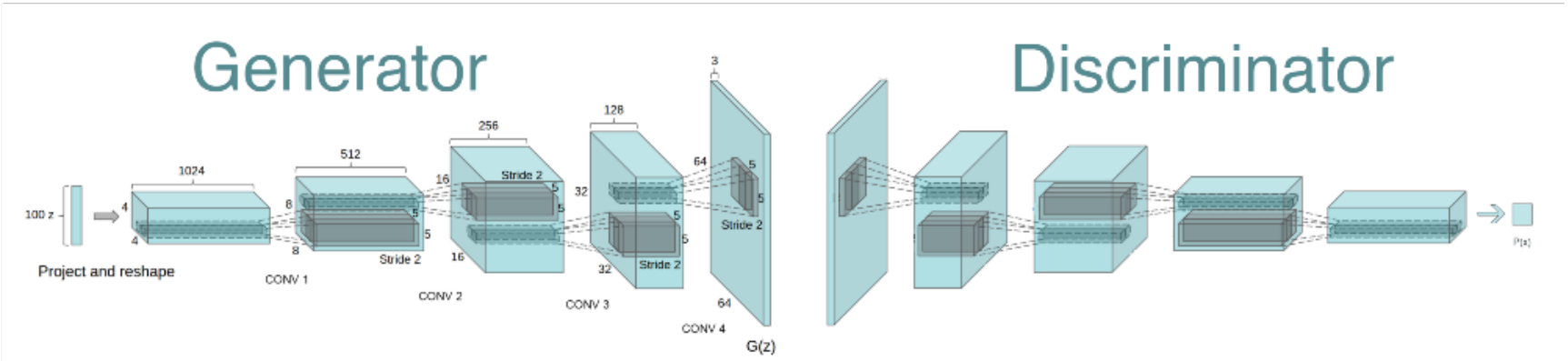
- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. ***Repeat K times (with fixed Θ_g):***
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f} [\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$
2. ***Execute once (with fixed Θ_f):***
   1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g} [\sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$

- Why limiting the steps (K) when updating Θ_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

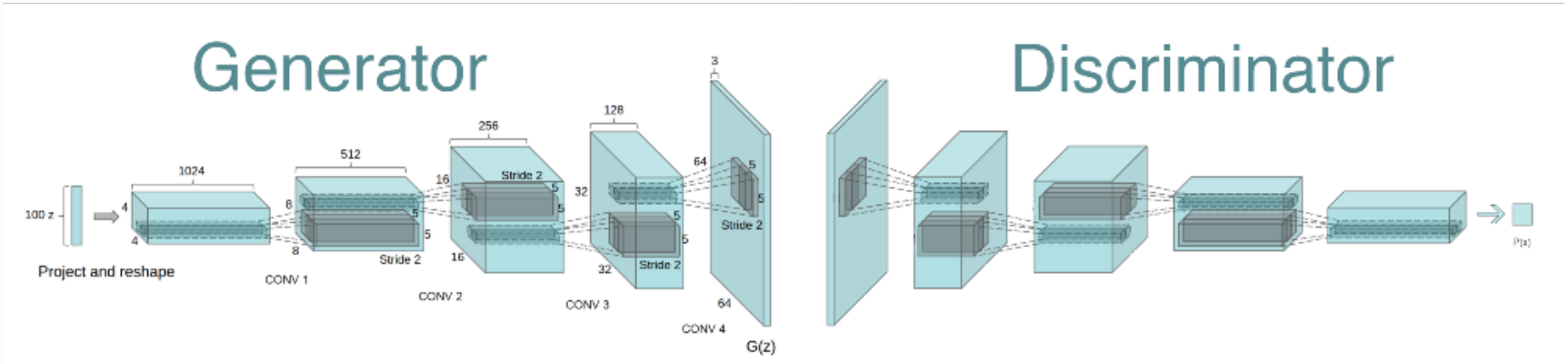
Results

- Domain-specific architecture, e.g., DC-GAN [17]



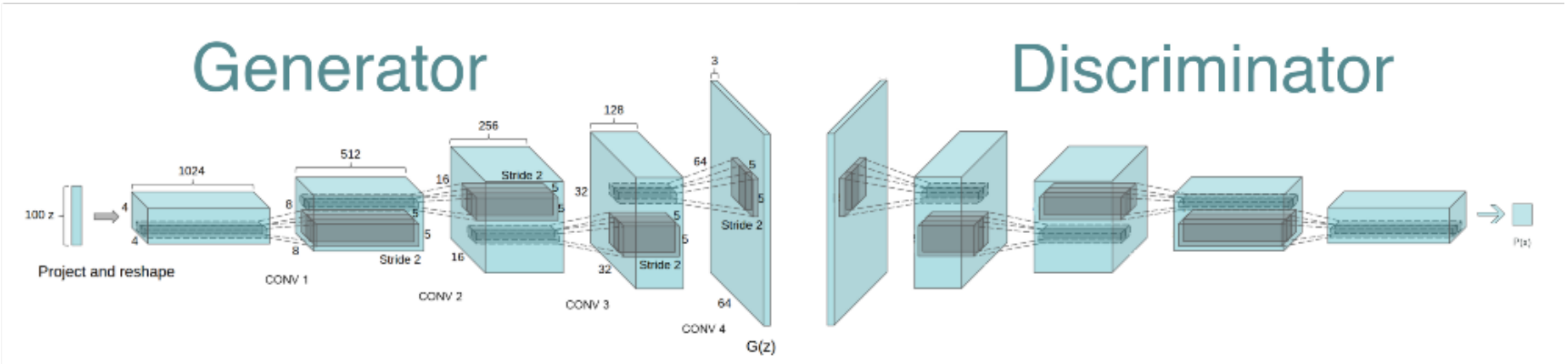
Results

- Domain-specific architecture, e.g., DC-GAN [17]



Results

- Domain-specific architecture, e.g., DC-GAN [17]



- GANs are well-known to be hard to train
  - GAN hacks on GitHub

Outline

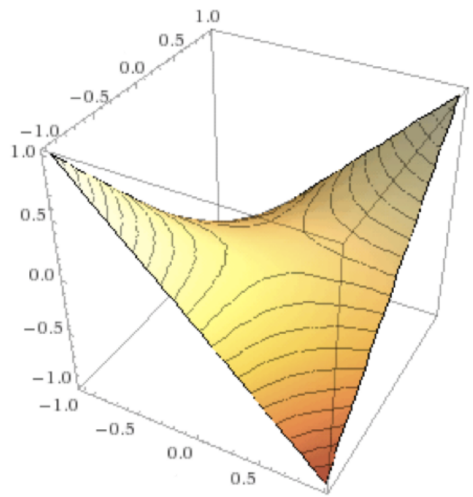
Challenge: Non-Convergence

- 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

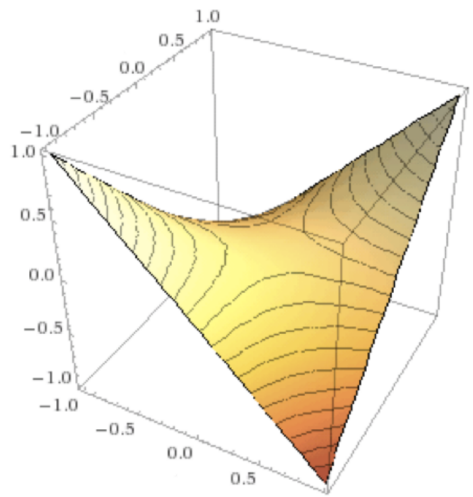
$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$



Challenge: Non-Convergence

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

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

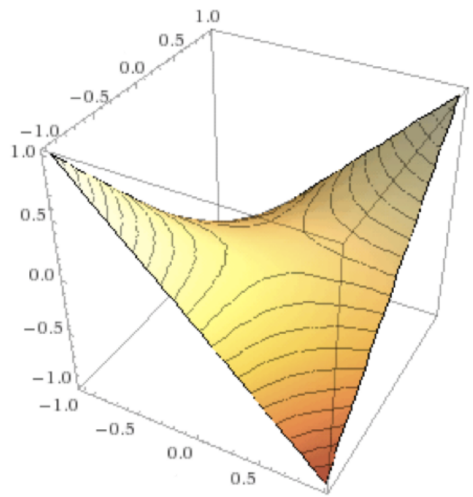


- The updated Θ_f and Θ_g may cancel each other's progress

Challenge: Non-Convergence

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

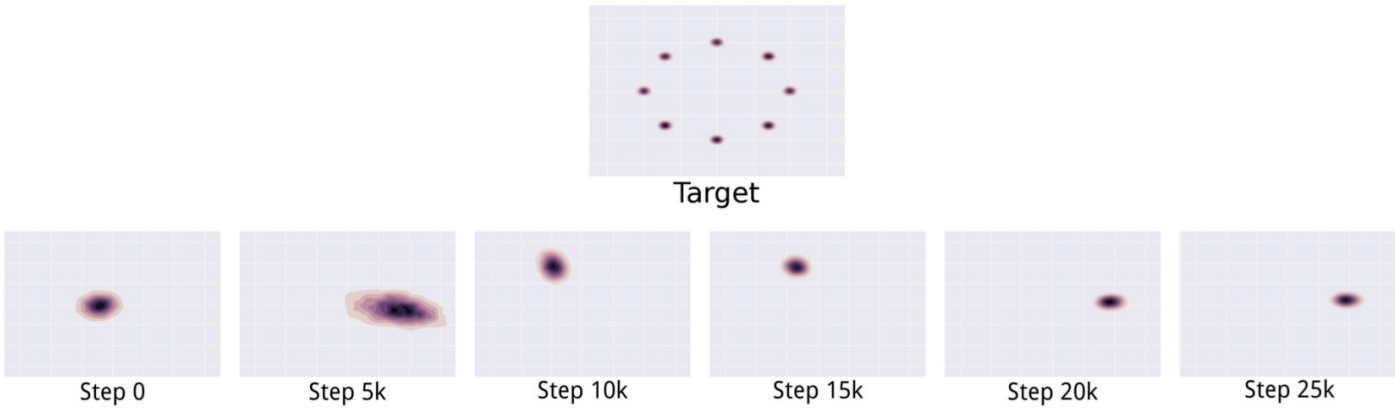
$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$



- The updated Θ_f and Θ_g may cancel each other's progress
- Requires human monitoring and termination

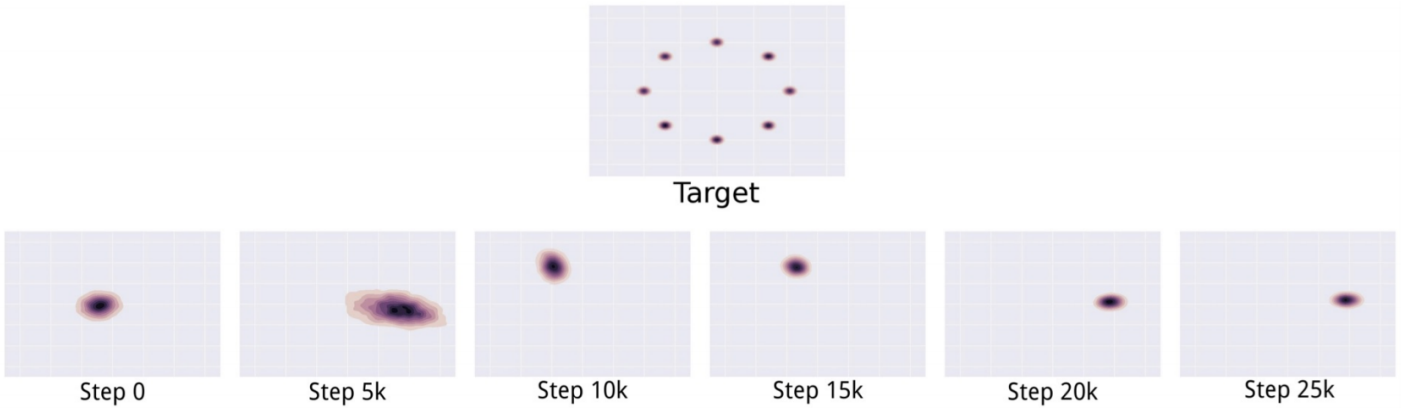
Mode Collapsing

- 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



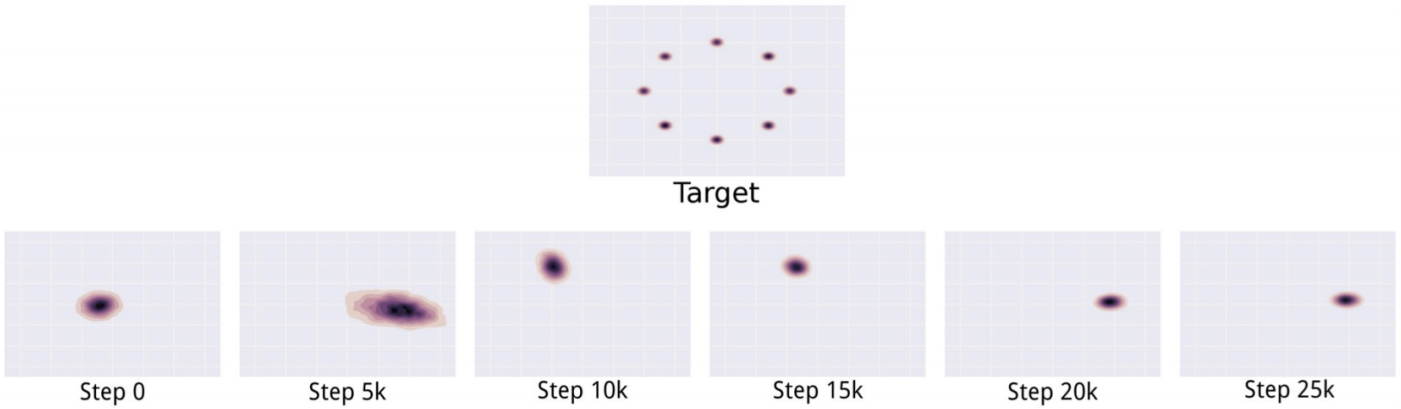
- When K is small, alternate SGD does not distinguish between $\min_{\Theta_g} \max_{\Theta_f}$ and $\max_{\Theta_f} \min_{\Theta_g}$

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

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

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

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

- $\max_{\Theta_f}\min_{\Theta_g}$? g is encouraged to map every code to the "mode" that f believes is most likely to be real

Solutions

- Minibatch discrimination [21]
  - 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 $\boldsymbol{x}^{(n)}/\hat{\boldsymbol{x}}^{(n)}$ with ***batch features***?

Solutions

- Minibatch discrimination [21]
  - 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 $\boldsymbol{x}^{(n)}/\hat{\boldsymbol{x}}^{(n)}$ 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 [21]
  - 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 $\boldsymbol{x}^{(n)}/\hat{\boldsymbol{x}}^{(n)}$ 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

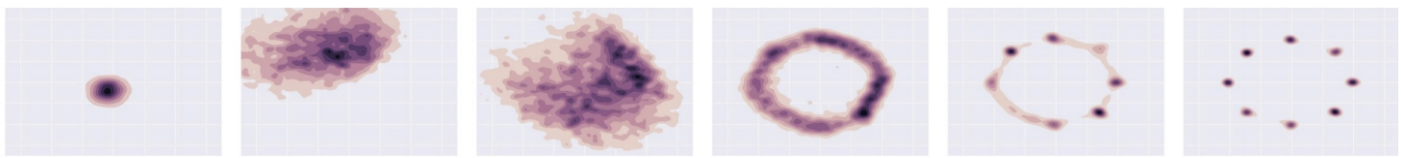
without   with

Solutions

- Minibatch discrimination [21]
  - 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 $\boldsymbol{x}^{(n)}/\hat{\boldsymbol{x}}^{(n)}$ 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

without   with

- Unrolled GANs [14]: to back-propagate through ***several*** $\max$ ***steps*** when computing $\nabla_{\Theta_g} C$



Challenge: Balance between g and f

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

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

Challenge: Balance between g and f

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

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

- Too small K:

Challenge: Balance between g and f

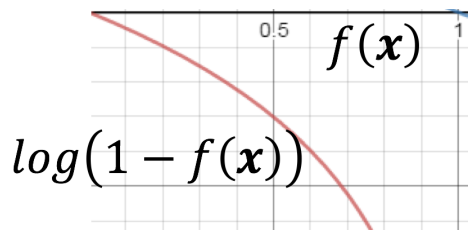
$$\arg \min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

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

Challenge: Balance between g and f

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

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

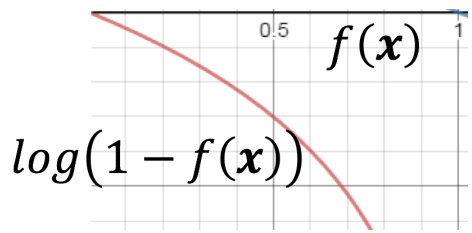


- Too small K:

Challenge: Balance between g and f

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

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

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

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

Solution: Wasserstein GAN [1]

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

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

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. Repeat K times (with fixed Θ_g):
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta\, clip(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))])$

Solution: Wasserstein GAN [1]

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

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

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. Repeat K times (with fixed Θ_g):
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta\, clip(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))])$
2. Execute once (with fixed Θ_f):
   1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g}[-\sum_m f(g(\boldsymbol{c}^{(m)}))]$

GANs from Information Theory Perspective

- Review the Information Theory first!

GANs from Information Theory Perspective

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

GANs from Information Theory Perspective

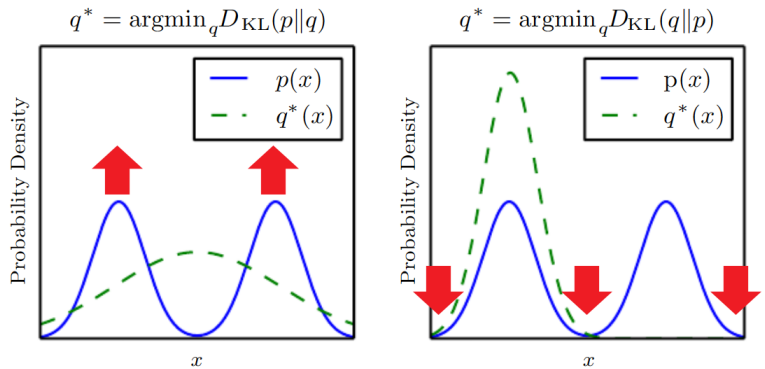
- Review the Information Theory first!
- Let P_{data} / P_g be distribution of $\mathbf{x}$ / $\hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find g: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{KL}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)$

GANs from Information Theory Perspective

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

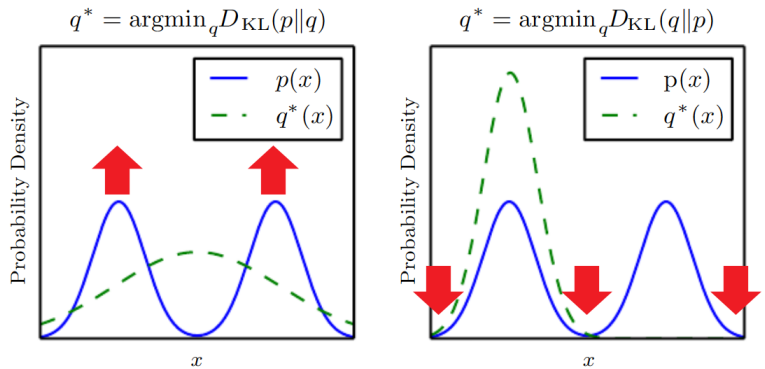
GANs from Information Theory Perspective

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



GANs from Information Theory Perspective

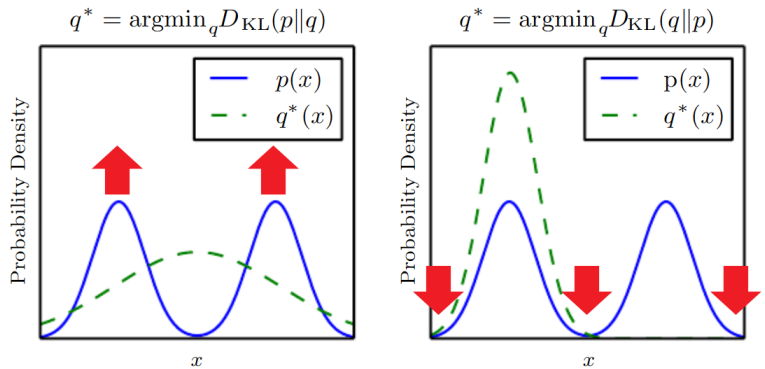
- Review the Information Theory first!
- Let P_{data} / P_g be distribution of $\mathbf{x}$ / $\hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find g: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{KL}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)$
  - Why not $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{KL}}(\mathrm{P}_g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



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

GANs from Information Theory Perspective

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



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

$$\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g) = \frac{1}{2}\mathrm{D}_{\mathrm{KL}}(\mathrm{P}_{\text{data}} \| \mathrm{Q}) + \frac{1}{2}\mathrm{D}_{\mathrm{KL}}(\mathrm{P}_g \| \mathrm{Q}), \text{ where } \mathrm{Q} = \frac{1}{2}(\mathrm{P}_g + \mathrm{P}_{\text{data}})$$

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

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

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

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

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

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

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

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

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

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

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

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

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

$$\mathrm{P}_{\mathsf{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x}))$$

  for each $\boldsymbol{x}$
  - Assuming that f has infinite capacity

Why Jensen-Shannon Divergence? II

Why Jensen-Shannon Divergence? II

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

$$\mathrm{P}_{\text{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x}))?$$

- $f^*(\boldsymbol{x}) = \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})} \in [0, 1]$ [Proof]
- That is,

$$C^* = \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\text{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x}))] d\boldsymbol{x}$$

Why Jensen-Shannon Divergence? II

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

$$\mathrm{P}_{\text{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x}))?$$

- $f^*(\boldsymbol{x}) = \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})} \in [0, 1]$ [Proof]
- That is,

$$\begin{aligned} C^* &= \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\text{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x}))] d\boldsymbol{x} \\ &= \int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x}) \log \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})} d\boldsymbol{x} \\ &\quad + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log(1 - \frac{\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})}) d\boldsymbol{x} \end{aligned}$$

Why Jensen-Shannon Divergence? II

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

$$\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))?$$

- $f^*(\boldsymbol{x}) = \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})} \in [0,1]$ [Proof]
- That is,

$$\begin{aligned} C^* &= \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))]d\boldsymbol{x} \\ &= \int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x}) \log \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})} d\boldsymbol{x} \\ &\qquad + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log(1 - \frac{\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})}) d\boldsymbol{x} \\ &= -2\log 2 + \int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x}) \log \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{(\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x}))/2} d\boldsymbol{x} \\ &\qquad + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log(1 - \frac{\mathrm{P}_g(\boldsymbol{x})}{(\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x}))/2}) d\boldsymbol{x} \end{aligned}$$

Why Jensen-Shannon Divergence? II

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

$$\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))?$$

- $f^*(\boldsymbol{x}) = \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})} \in [0,1]$ [Proof]
- That is,

$$\begin{aligned}
C^* &= \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))]d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x}) \log \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})} d\boldsymbol{x} \\
&\qquad + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log\left(1 - \frac{\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})}\right) d\boldsymbol{x} \\
&= -2\log 2 + \int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x}) \log \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{(\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x}))/2} d\boldsymbol{x} \\
&\qquad + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log\left(1 - \frac{\mathrm{P}_g(\boldsymbol{x})}{(\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x}))/2}\right) d\boldsymbol{x} \\
&= -2\log 2 + 2\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)
\end{aligned}$$

Balance between g and f, Revisited I

- Cost function of GAN:

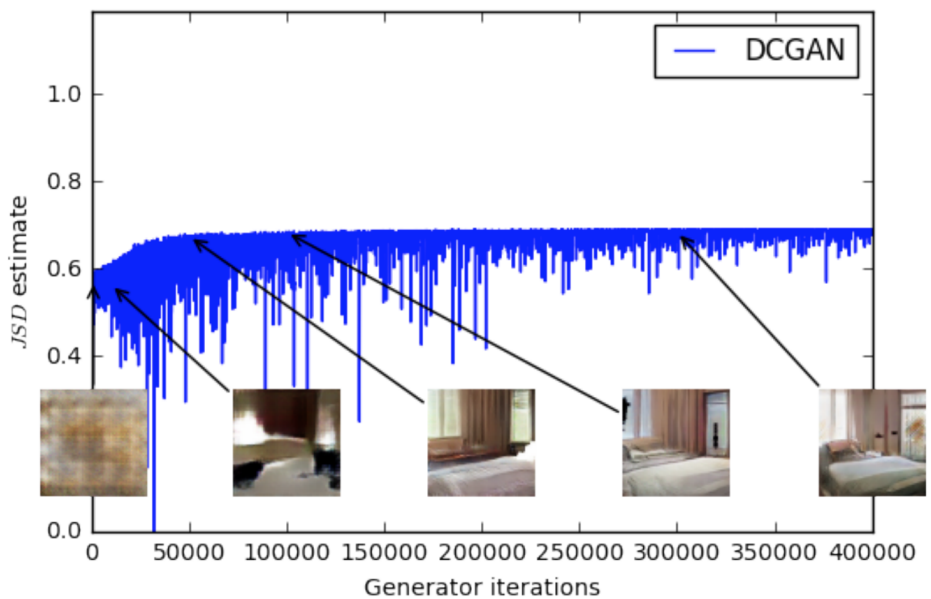
$$\begin{aligned}&\arg\min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)}))) \\ &= \arg\min_{\Theta_g} -2\log 2 + 2\mathrm{D}_{\mathsf{JS}}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)\end{aligned}$$

Balance between g and f, Revisited I

- Cost function of GAN:

$$\arg\min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$
$$= \arg\min_{\Theta_g} -2\log 2 + 2\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$$

- However, no mater how g changes, $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ remains high during the GAN training process

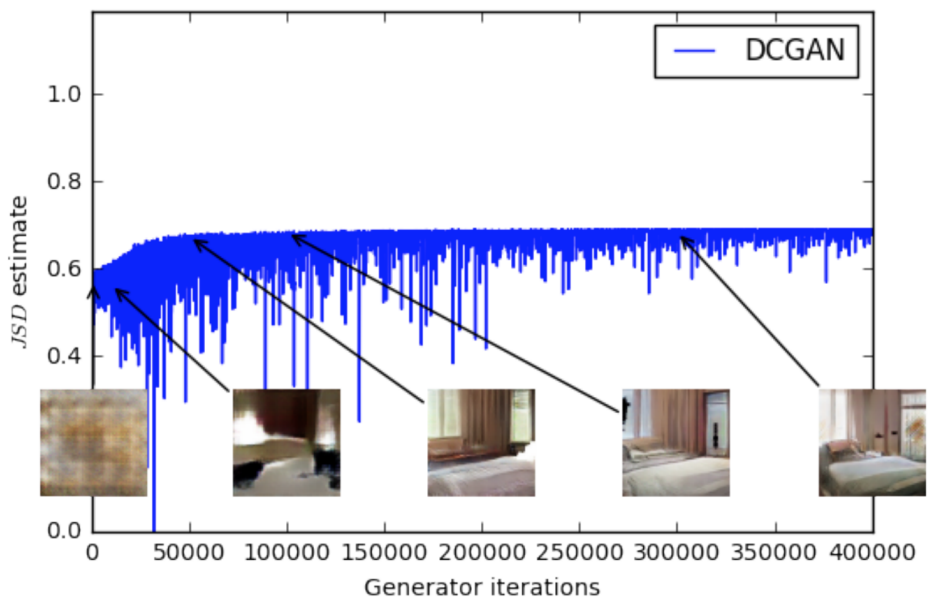


Balance between g and f, Revisited I

- Cost function of GAN:

$$\arg\min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$
$$= \arg\min_{\Theta_g} -2\log 2 + 2\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$$

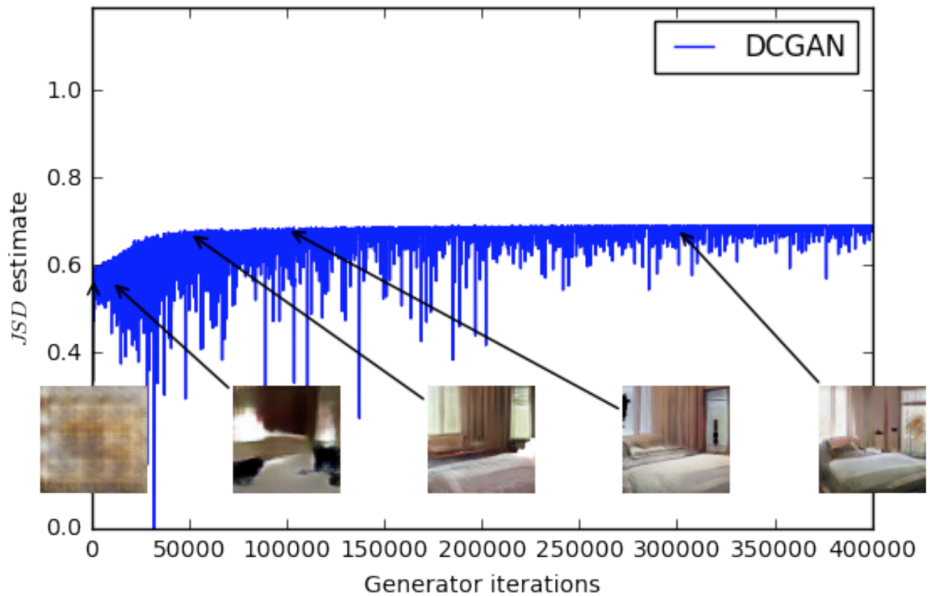
- However, no mater how g changes, $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ remains high during the GAN training process



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

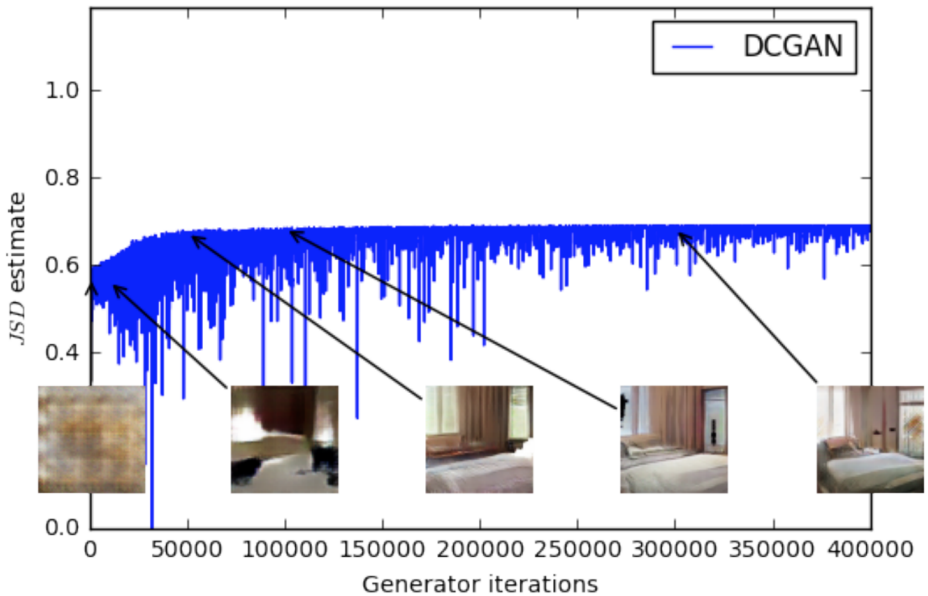
Balance between g and f, Revisited II

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



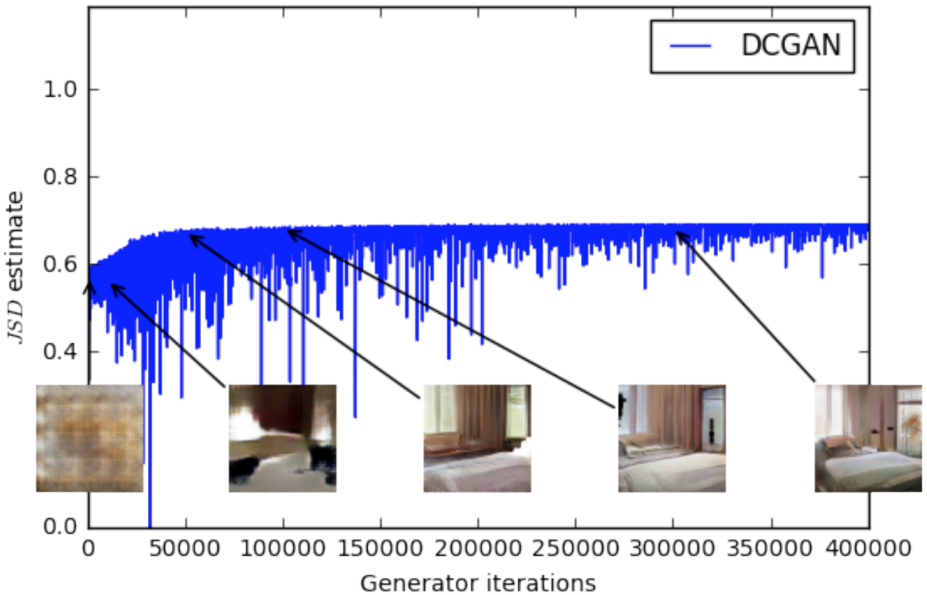
Balance between g and f, Revisited II

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



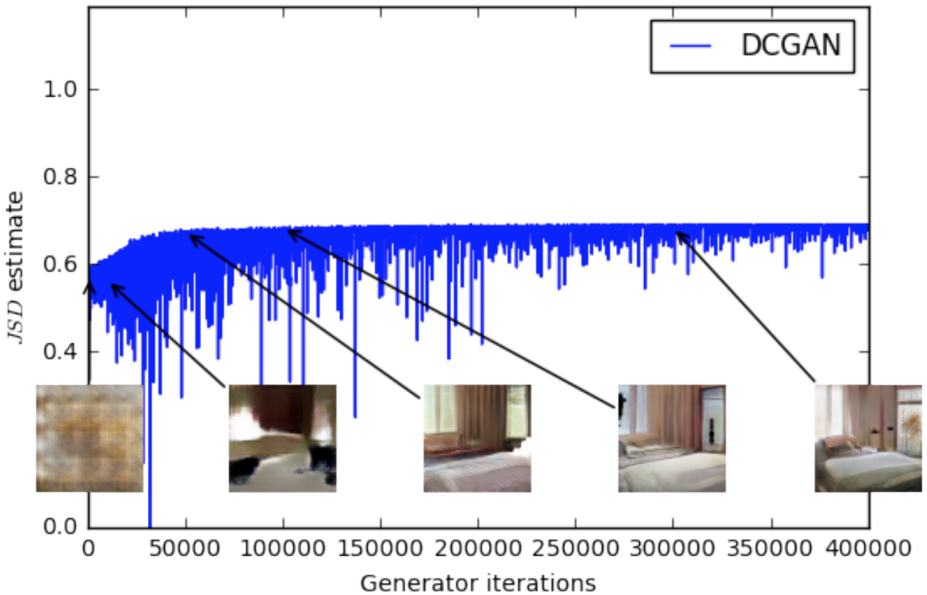
Balance between g and f, Revisited II

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



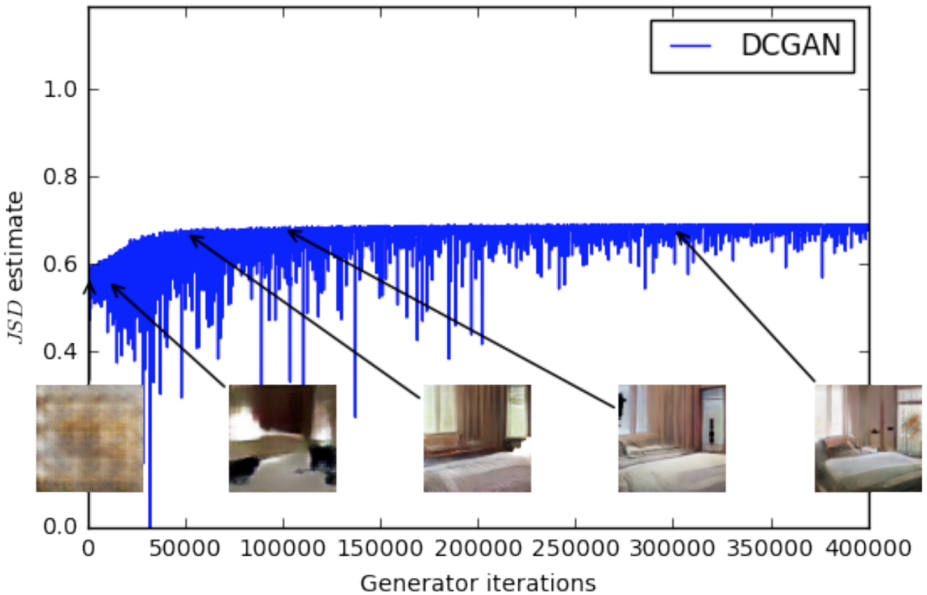
Balance between g and f, Revisited II

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



Balance between g and f, Revisited II

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

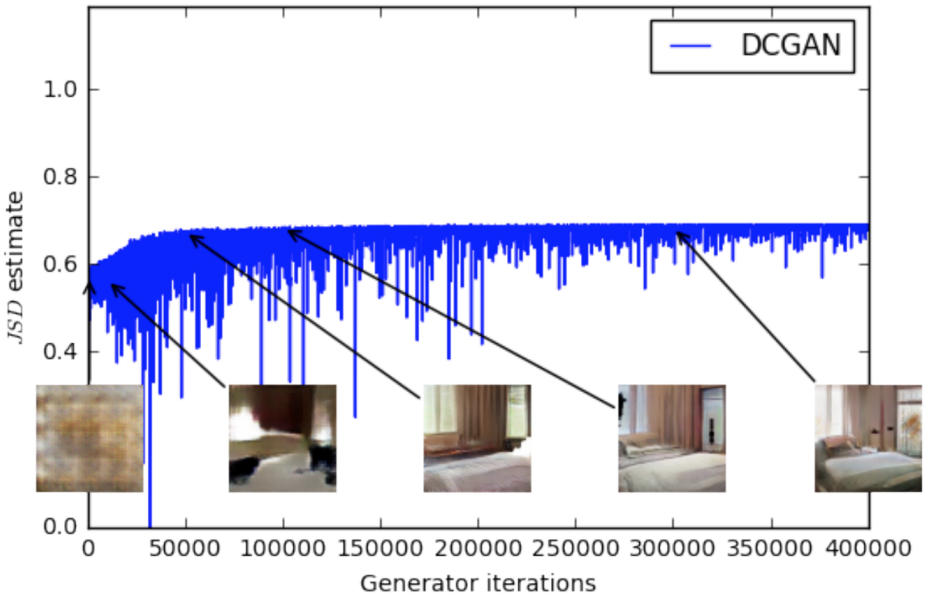


- Suppose $P_g(\boldsymbol{x}) \neq 0 \Leftrightarrow P_{data}(\boldsymbol{x}) = 0$ and $P_g(\boldsymbol{x}) = 0 \Leftrightarrow P_{data}(\boldsymbol{x}) \neq 0$, we have

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

Balance between g and f, Revisited II

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

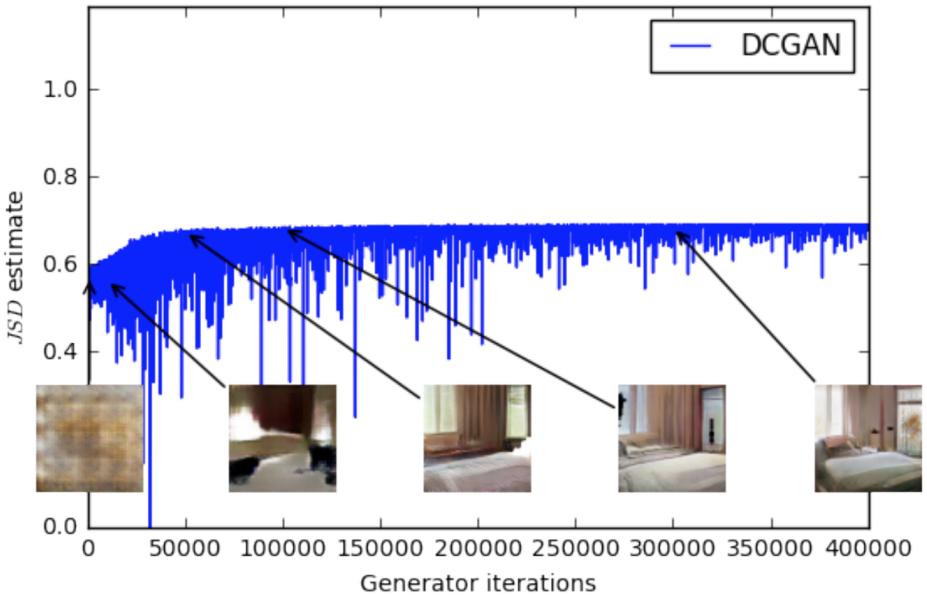


- Suppose $P_g(x) \neq 0 \Leftrightarrow P_{data}(x) = 0$ and $P_g(x) = 0 \Leftrightarrow P_{data}(x) \neq 0$, we have

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

Balance between g and f, Revisited II

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

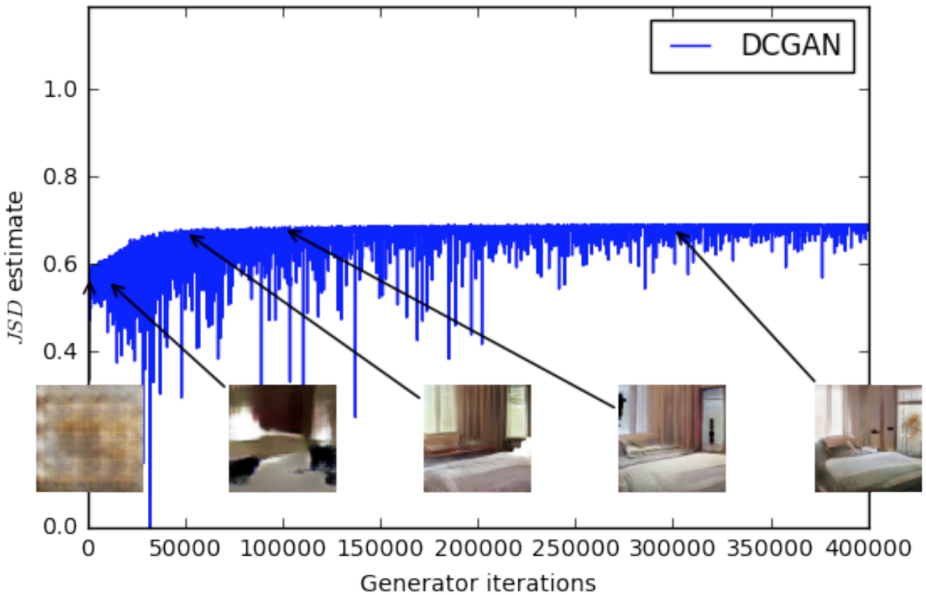


- Suppose $P_g(x) \neq 0 \Leftrightarrow P_{data}(x) = 0$ and $P_g(x) = 0 \Leftrightarrow P_{data}(x) \neq 0$, we have

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

Balance between g and f, Revisited II

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



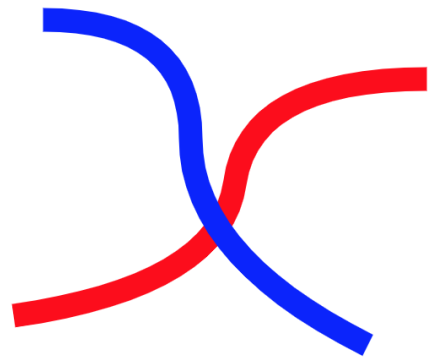
- Suppose $\mathrm{P}_g(\boldsymbol{x}) \neq 0 \Leftrightarrow \mathrm{P_{data}}(\boldsymbol{x}) = 0$ and $\mathrm{P}_g(\boldsymbol{x}) = 0 \Leftrightarrow \mathrm{P_{data}}(\boldsymbol{x}) \neq 0$, we have

$$\begin{aligned}
\mathrm{D_{JS}}(\mathrm{P}_g \| \mathrm{P_{data}}) &= \tfrac{1}{2}\mathrm{D_{KL}}(\mathrm{P}_g \| \tfrac{\mathrm{P}_g + \mathrm{P_{data}}}{2}) + \tfrac{1}{2}\mathrm{D_{KL}}(\mathrm{P_{data}} \| \tfrac{\mathrm{P}_g + \mathrm{P_{data}}}{2}) \\
&= \tfrac{1}{2}\int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log \frac{2\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_g(\boldsymbol{x}) + \mathrm{P_{data}}(\boldsymbol{x})} d\boldsymbol{x} \\
&\qquad + \tfrac{1}{2}\int_{\boldsymbol{x}} \mathrm{P_{data}}(\boldsymbol{x}) \log \frac{2\mathrm{P_{data}}(\boldsymbol{x})}{\mathrm{P}_g(\boldsymbol{x}) + \mathrm{P_{data}}(\boldsymbol{x})} d\boldsymbol{x} \\
&= \tfrac{1}{2}\log 2 + \tfrac{1}{2}\log 2 = \log 2
\end{aligned}$$

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

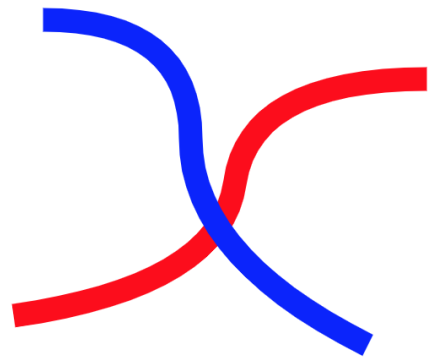
Disjoining $\mathbf{P}_g$ and $\mathbf{P}_{\text{data}}$

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



Disjoining $\mathbf{P}_g$ and $\mathbf{P}_{\text{data}}$

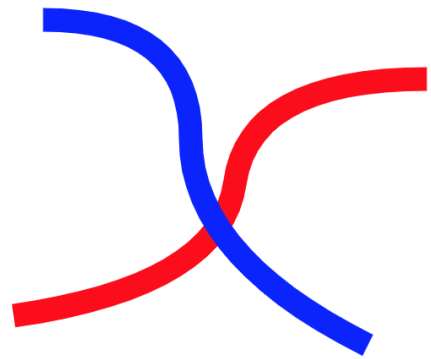
- In a high dimensional space, $\mathbf{x}$ and $g(\mathbf{z})$ may resides in low dimensional manifolds
  - P_g and P_{data} may have values only on the manifolds



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

Disjoining P_g and P_{data}

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



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

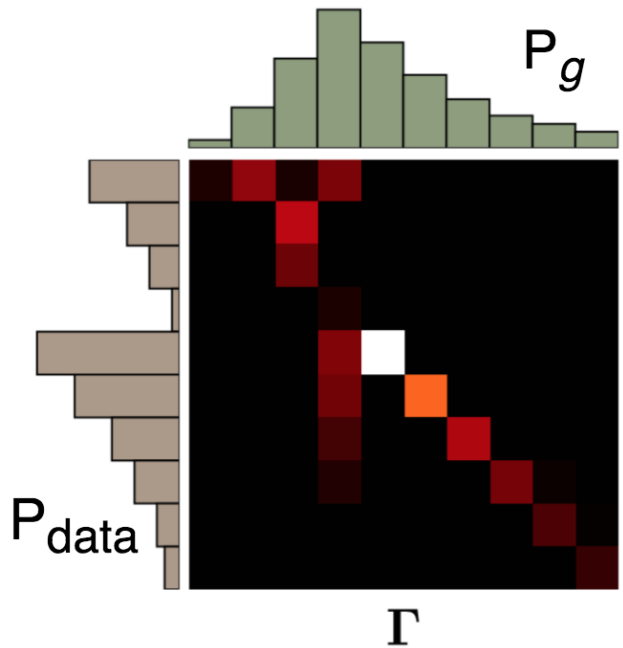
Better Divergence Measure?

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

Better Divergence Measure?

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

$$\begin{aligned} \mathrm{W}(\mathrm{P_{data}}, \mathrm{P}_g) &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P_{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|] \\ &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P_{data}}, \mathrm{P}_g)} \int_{(\boldsymbol{x}, \hat{\boldsymbol{x}})} \mathrm{Q}(\boldsymbol{x}, \hat{\boldsymbol{x}}) \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| d(\boldsymbol{x}, \hat{\boldsymbol{x}}) \end{aligned}$$

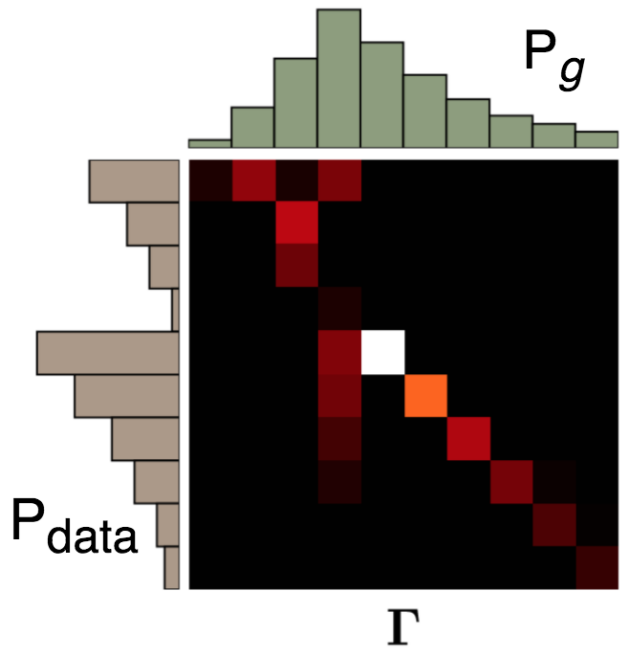


Better Divergence Measure?

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

$$\begin{aligned} \mathrm{W}(\mathrm{P_{data}}, \mathrm{P}_g) &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P_{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|] \\ &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P_{data}}, \mathrm{P}_g)} \int_{(\boldsymbol{x}, \hat{\boldsymbol{x}})} \mathrm{Q}(\boldsymbol{x}, \hat{\boldsymbol{x}}) \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| d(\boldsymbol{x}, \hat{\boldsymbol{x}}) \end{aligned}$$

- Intuitively, the minimal "cost" to change $\mathrm{P_{data}}$ into P_g

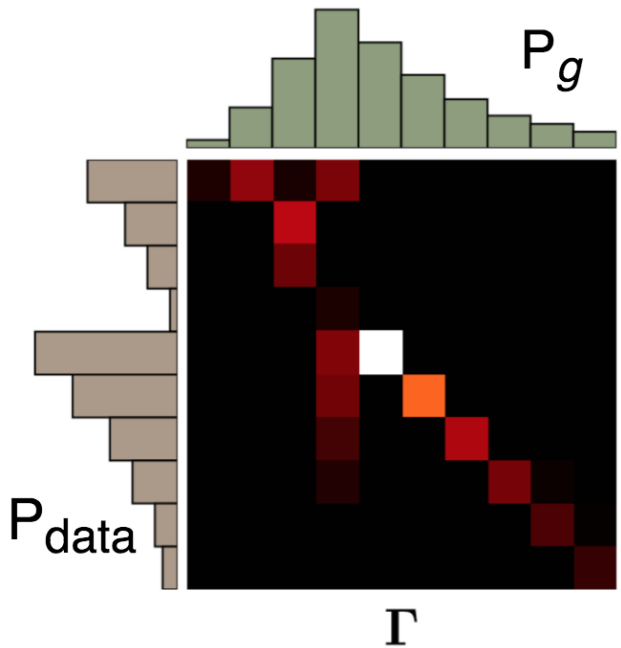


Better Divergence Measure?

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

$$\begin{aligned} \mathrm{W}(\mathrm{P_{data}}, \mathrm{P}_g) &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P_{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|] \\ &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P_{data}}, \mathrm{P}_g)} \int_{(\boldsymbol{x}, \hat{\boldsymbol{x}})} \mathrm{Q}(\boldsymbol{x}, \hat{\boldsymbol{x}}) \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| d(\boldsymbol{x}, \hat{\boldsymbol{x}}) \end{aligned}$$

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



Wasserstein GAN I

- W-GAN: $\arg\min_{\Theta_g} \mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$
  - $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g) = \inf_{\mathrm{Q} \in \Gamma(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|]$

Wasserstein GAN I

- W-GAN: $\arg\min_{\Theta_g} \mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$
  - $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g) = \inf_{\mathrm{Q} \in \Gamma(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|]$
- Unfortunately, $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$ is hard to solve directly

Wasserstein GAN II

Theorem

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

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

we have [a]

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

Wasserstein GAN II

Theorem

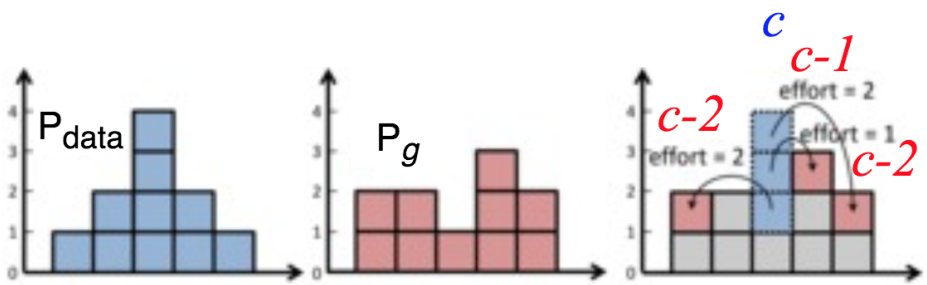
Consider f's that are Lipschitz continuous with constant 1, i.e.,

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

we have [a]

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

[a] https://vincentherrmann.github.io/blog/wasserstein/



Wasserstein GAN II

Theorem

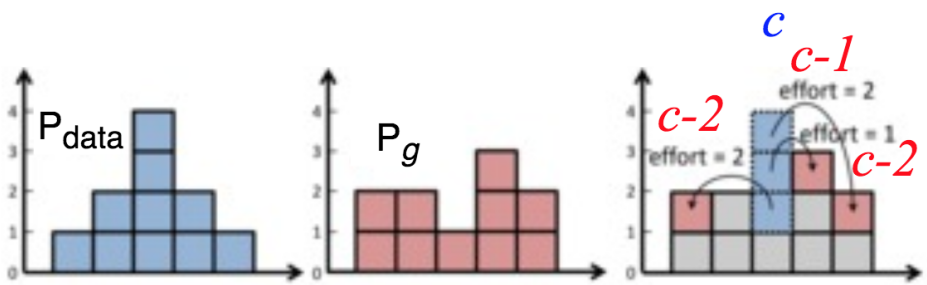
Consider f's that are Lipschitz continuous with constant 1, i.e.,

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

we have [a]

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

[a] https://vincentherrmann.github.io/blog/wasserstein/



- W-GAN [1]: $\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\text{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_g}[f(\mathbf{x})]$

Alternate SGD for W-GAN

$$\begin{aligned}&\arg\min_{\Theta_g}\max_{\Theta_f}\mathrm{E}_{\mathbf{x}\sim\mathrm{P}_{\mathbf{data}}}[f(\mathbf{x})]-\mathrm{E}_{\mathbf{x}\sim\mathrm{P}_g}[f(\mathbf{x})]\\ &=\arg\min_{\Theta_g}\max_{\Theta_f}\textstyle\sum_n f(\boldsymbol{x}^{(n)})-\sum_m f(g(\boldsymbol{c}^{(m)}))\end{aligned}$$

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

Alternate SGD for W-GAN

$$\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\text{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})]$$
$$= \arg\min_{\Theta_g} \max_{\Theta_f} \sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))$$

- f a regressor ***without*** the sigmoid output layer
- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. Repeat K times (with fixed Θ_g):
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta\, clip(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))])$
2. Execute once (with fixed Θ_f):
   1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g}[-\sum_m f(g(\boldsymbol{c}^{(m)}))]$

Why Gradient Clipping?

- Update rule for Θ_f:

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

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

Why Gradient Clipping?

- Update rule for Θ_f:

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

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

$$\begin{aligned} &\arg\min_{\Theta_g} \mathrm{W}(\mathrm{P}_{\text{data}}, \mathrm{P}_g) \\ &= \arg\min_{\Theta_g} \max_{\Theta_f} \textstyle\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)})) \end{aligned}$$

only if f is 1-Lipschitz continuous

Why Gradient Clipping?

- Update rule for Θ_f:

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

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

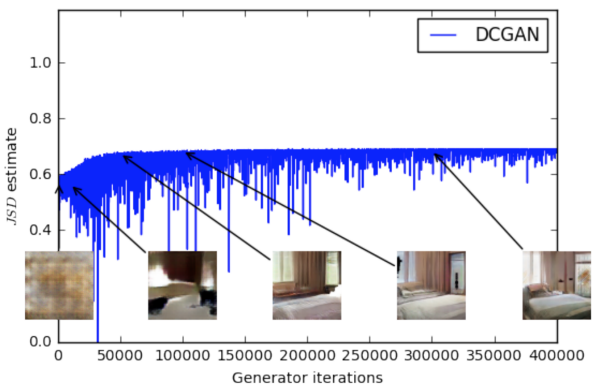
$$\begin{aligned} &\arg\min_{\Theta_g} \mathrm{W}(\mathrm{P}_{\text{data}}, \mathrm{P}_g) \\ &= \arg\min_{\Theta_g} \max_{\Theta_f} \textstyle\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)})) \end{aligned}$$

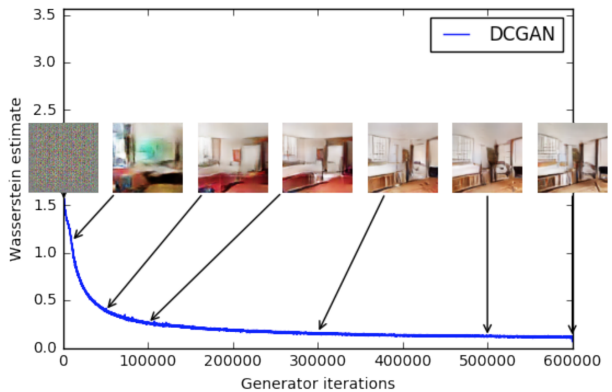
only if f is 1-Lipschitz continuous

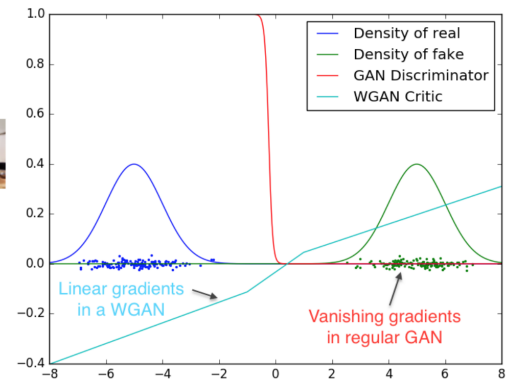
- Heuristics for making f 1-Lipchitz

Advantages of W-GAN

- "Corrects" the inner $\max$ problem
  - Wasserstein distance guides g even when P_g and P_{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 τ

Improved W-GAN I

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

$$\max_{\Theta_f} \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\mathsf{data}}} [f(\mathbf{x})] - \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_g} [f(\mathbf{x})]$$

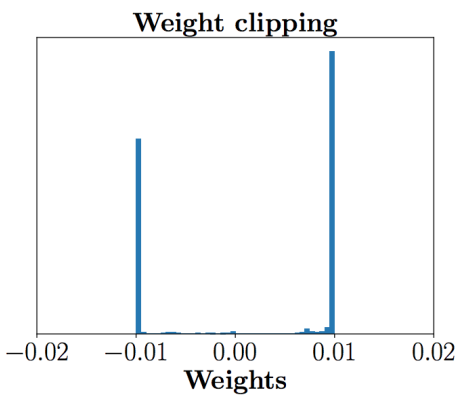
- g is not updated for minimizing $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$

Improved W-GAN I

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

$$\max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})]$$

- g is not updated for minimizing $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$
- Distribution of weight values of f ($\tau = 0.01$):



- Exploding and vanishing gradients

Improved W-GAN II

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

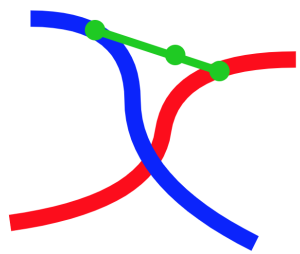
$$\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\text{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})] - \lambda \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\text{penalty}}}[\max(0, \|\nabla f(\mathbf{x})\| - 1)]$$

Improved W-GAN II

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

$$\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})] - \lambda \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{penalty}}}[\max(0, \|\nabla f(\mathbf{x})\| - 1)]$$

- $\mathrm{P}_{\mathsf{penalty}}$?

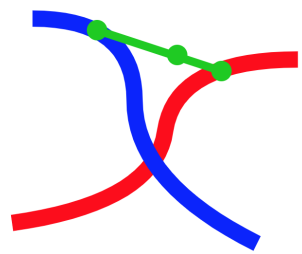


Improved W-GAN II

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

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

- $\mathrm{P}_{\text{penalty}}$?

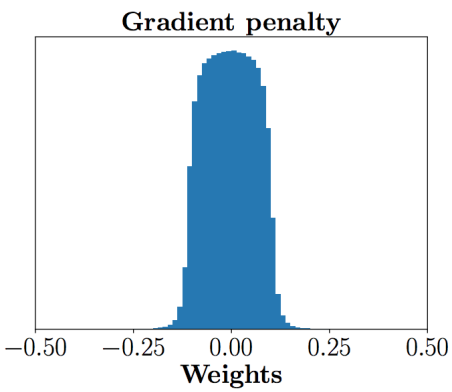


- W-GAN-GP [4]:

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

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

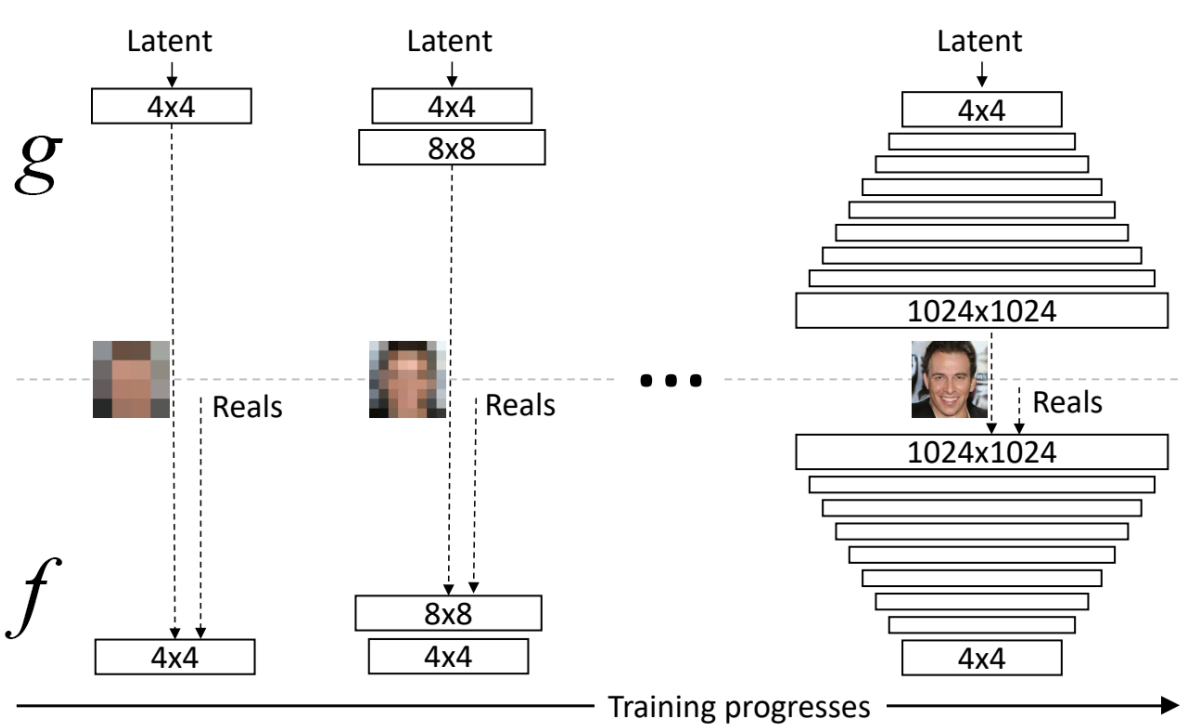
Counting

Perspective

Shape

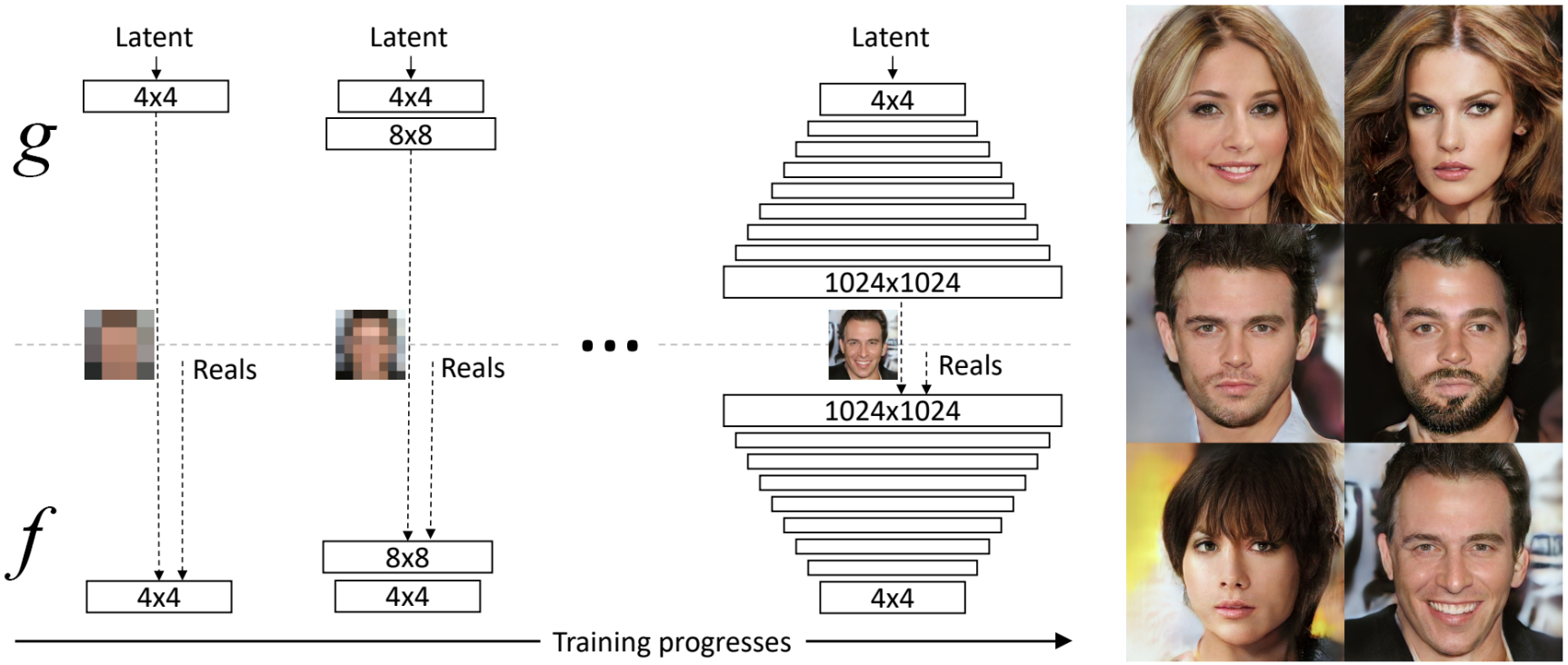
- A CNN, when used as f, detects ***existence*** of patterns more than their ***relative positions***
  - f loses track of the position of a pattern after several pooling layers
  - Relative position of patterns in $\mathbb{X}$ may change due to different view angels
- Solutions? A better f, such as the CapsuleNet [20]?

Progressive Growing of GANs [7]



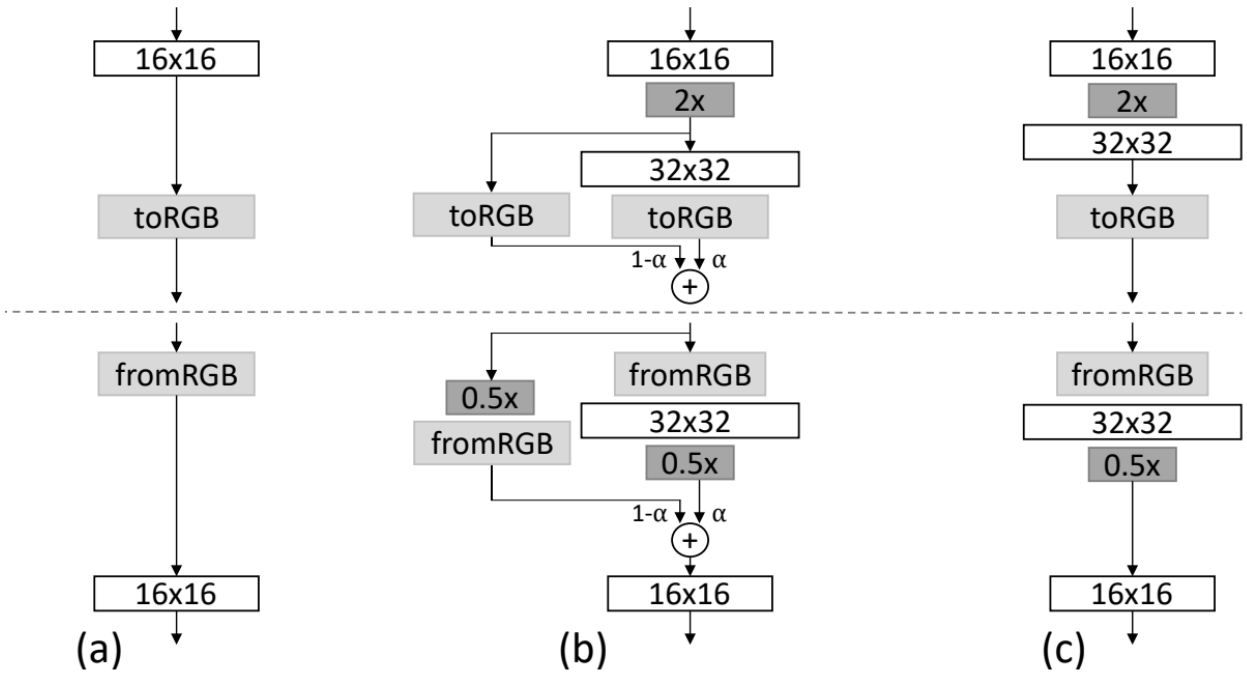
- Incrementally adds new layers in sequential GAN trainings
  - Convolution + upsampling for g each time
  - Convolution + downpooling for f each time
- Real images are downscaled to match the current resolution of g

Progressive Growing of GANs [7]



- 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 α
  - New convolution layer in g/f learns to generate/detect details first
  - Then learns the "context"

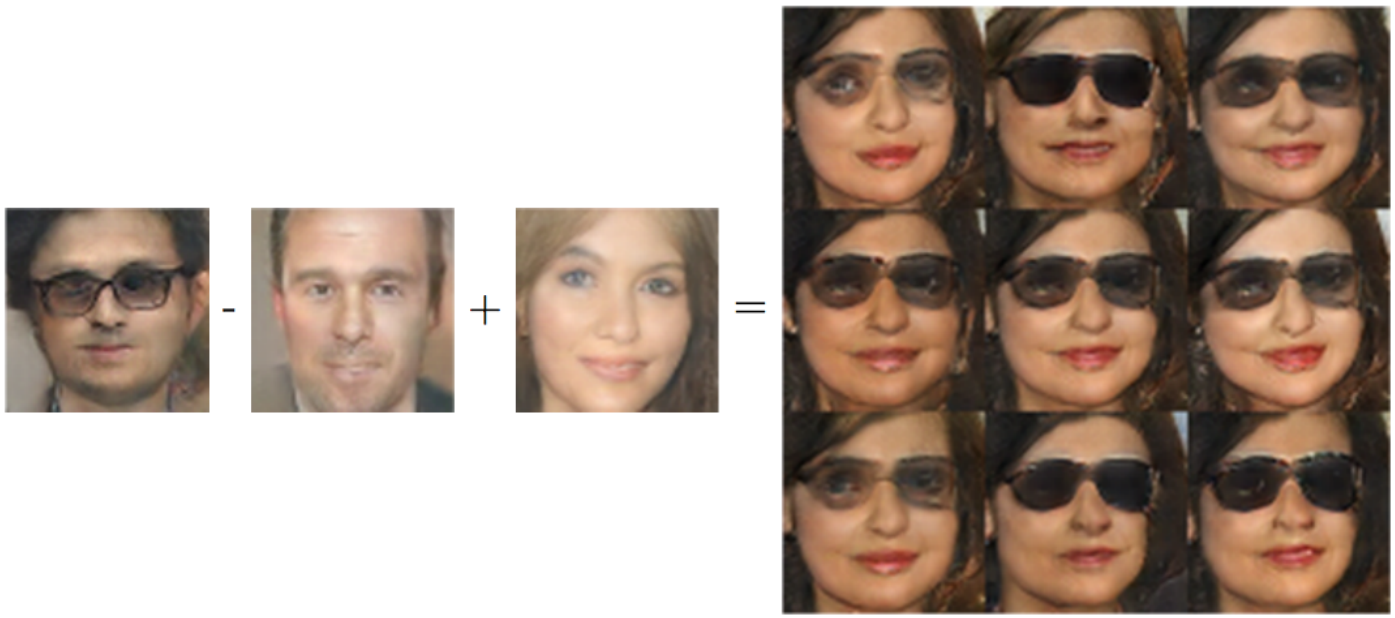
Results

- Minibatch discrimination [21] + W-GAN-GP [4] + progressive growing [7] + other tricks
- $2 \sim 6$ times faster

Outline

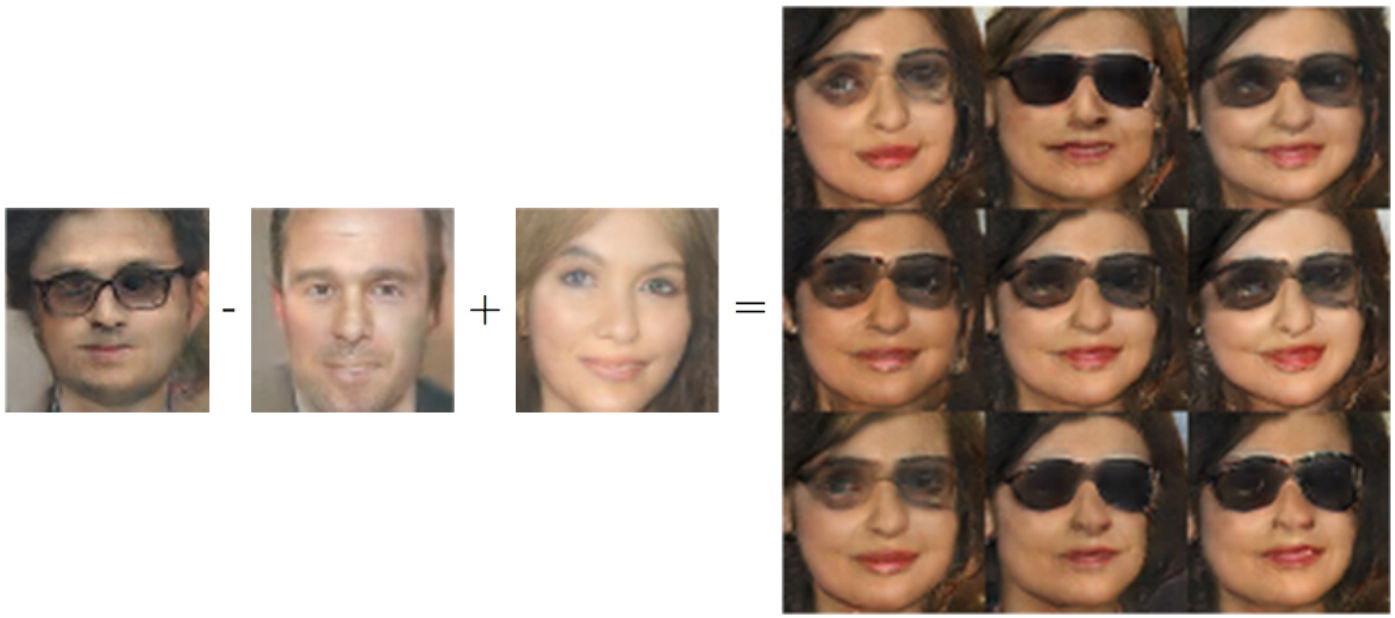
Code Space Arithmetics

- DC-GAN [17] can learn to use codes in meaningful ways:



Code Space Arithmetics

- DC-GAN [17] can learn to use codes in meaningful ways:



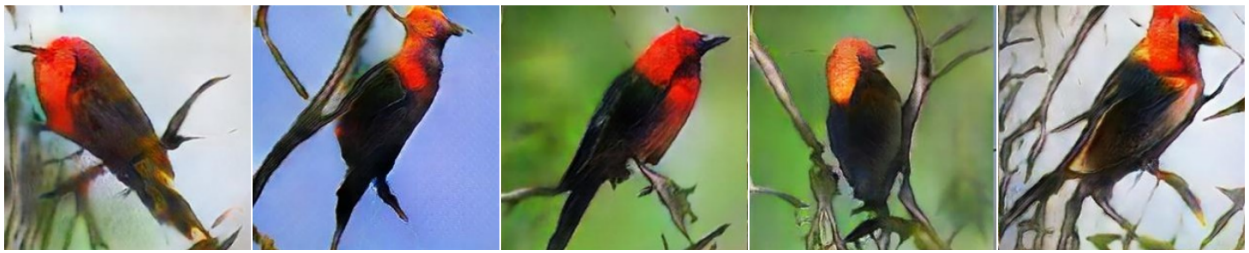
- Finding codes for images with constraints [28, 2] ▸ Demo 1 ▸ Demo 2

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

Conditional GAN I

- Text to image synthesis [18]: $\mathbb{X} = \{(\boldsymbol{x}^{(n)}, \boldsymbol{\phi}^{(n)})\}_n$

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

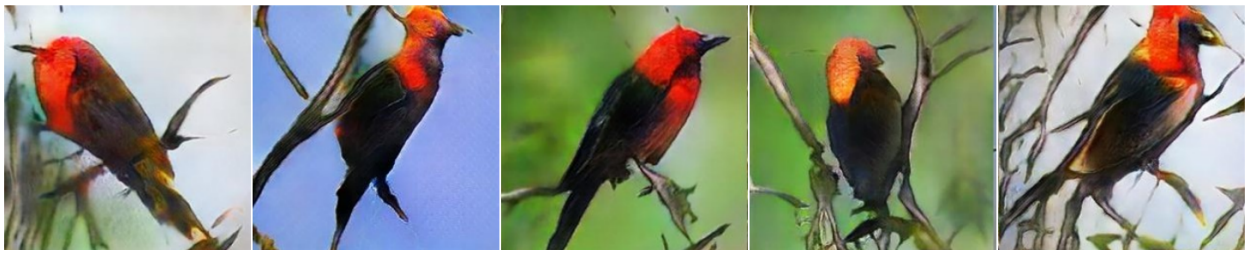


- How?

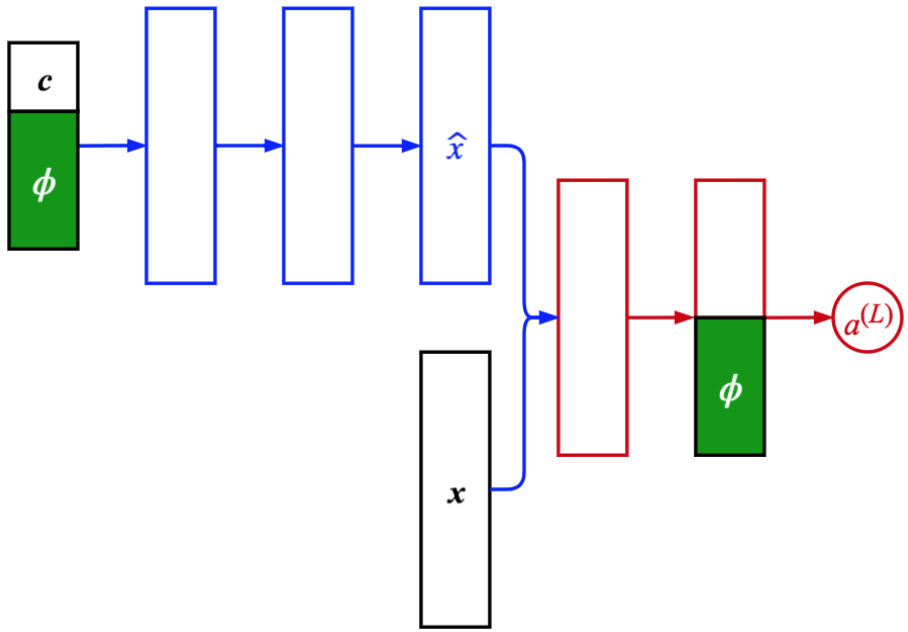
Conditional GAN I

- Text to image synthesis [18]: $\mathbb{X} = \{(\boldsymbol{x}^{(n)}, \boldsymbol{\phi}^{(n)})\}_n$

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

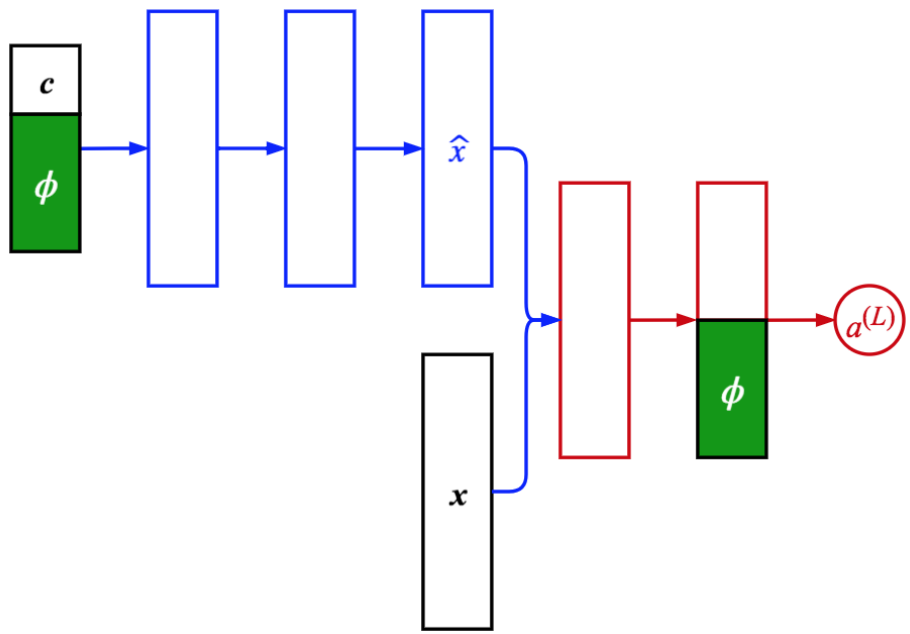


- How?



Conditional GAN II

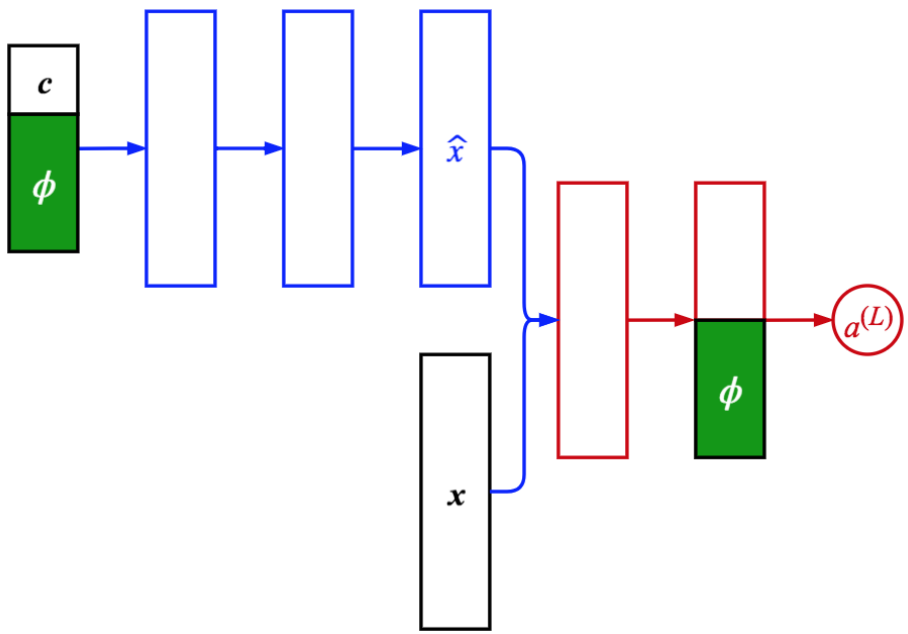
- Pitfall: g and f can choose to ignore the condition ϕ altogether
- Solution?



Conditional GAN II

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

$$(\boldsymbol{x}^{(n)}, \boldsymbol{\phi}^{(n)}) \Rightarrow \text{true}$$
$$(\boldsymbol{x}^{(n)}, \boldsymbol{\phi}') \Rightarrow \text{false}, \forall \boldsymbol{\phi}' \neq \boldsymbol{\phi}^{(n)}$$
$$(\hat{\boldsymbol{x}}^{(m)}, \boldsymbol{\phi}^{(m)}) \Rightarrow \text{false}$$



Super Resolution [10]

bicubic
(21.59dB/0.6423)

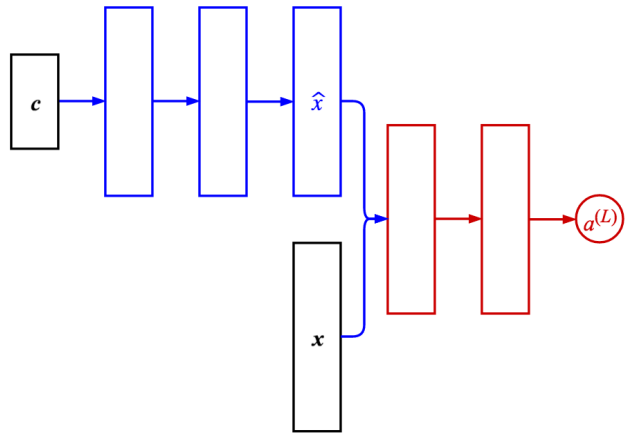
SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

Super Resolution [10]

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



bicubic
(21.59dB/0.6423)

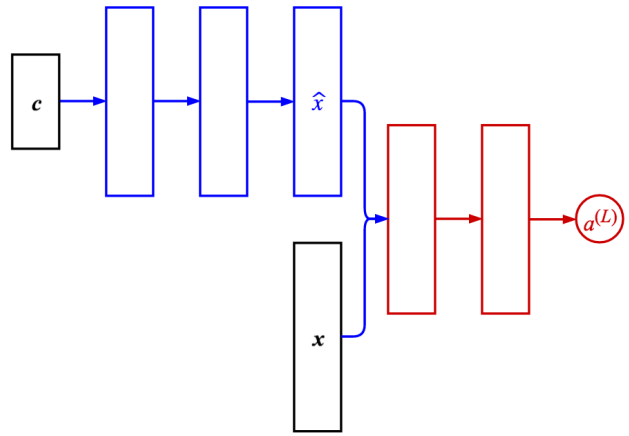
SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

Super Resolution [10]

- g : low res img $\rightarrow$ high res img
  - Training: $\boldsymbol{c}$'s are downscaled images
- No "creativity," f acts as a better lose metric



bicubic
(21.59dB/0.6423)

SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

Image-to-Image Translation [5] I

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

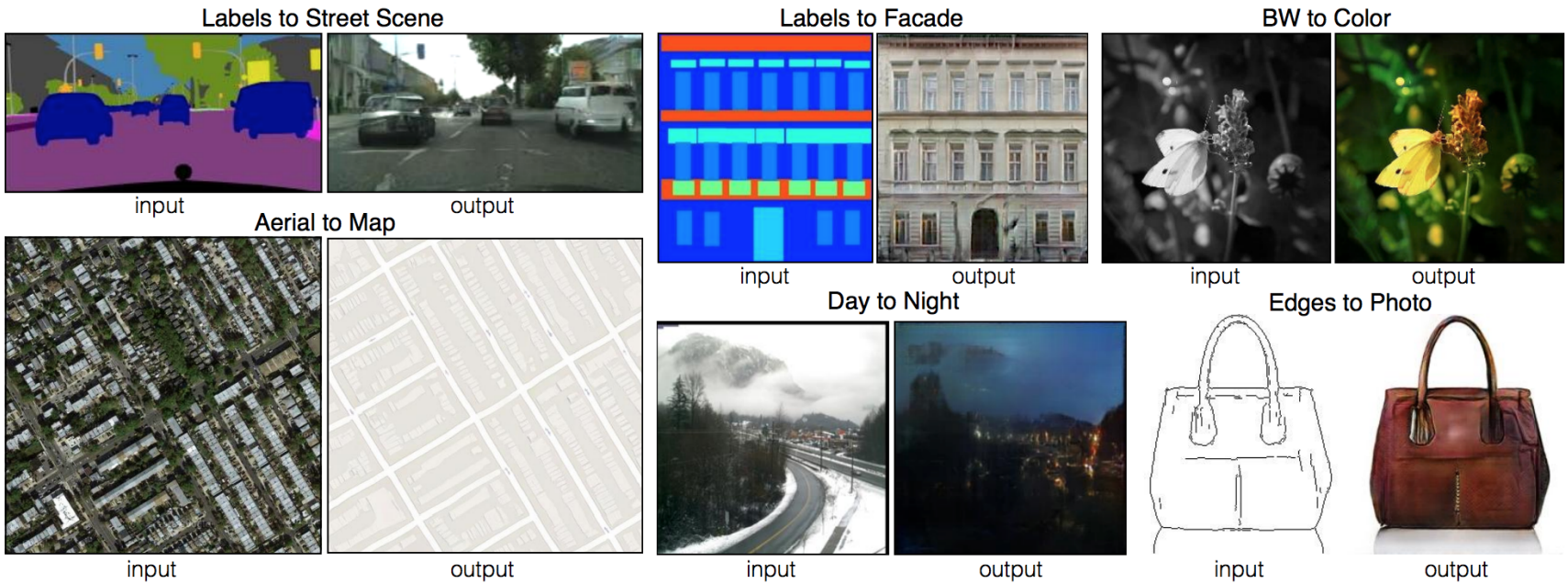


Image-to-Image Translation [5] II

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

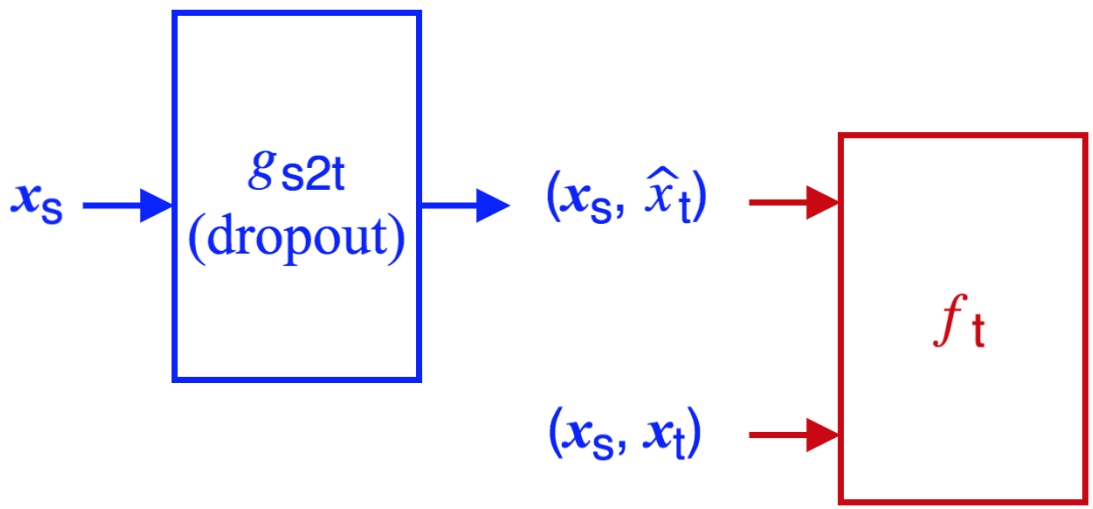
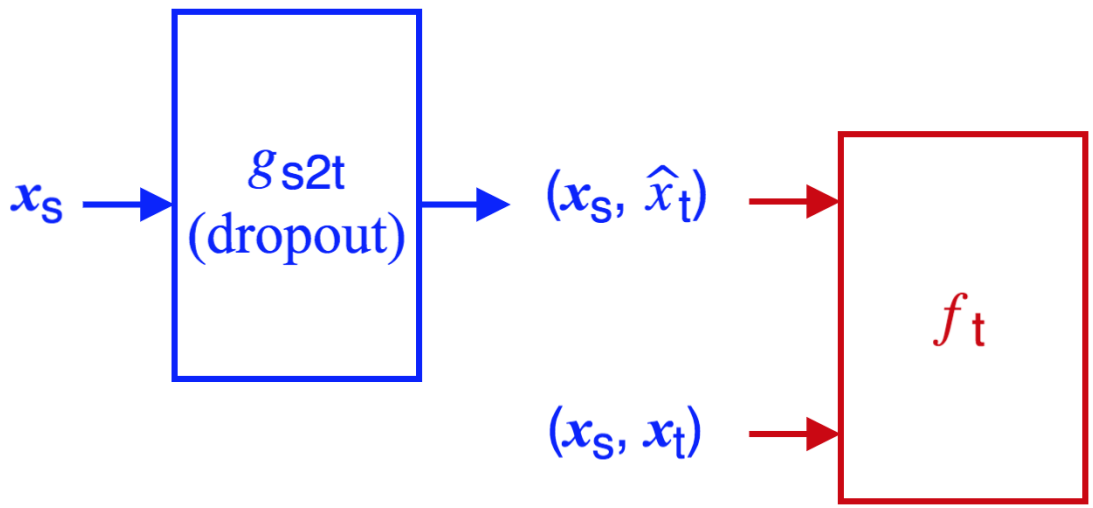


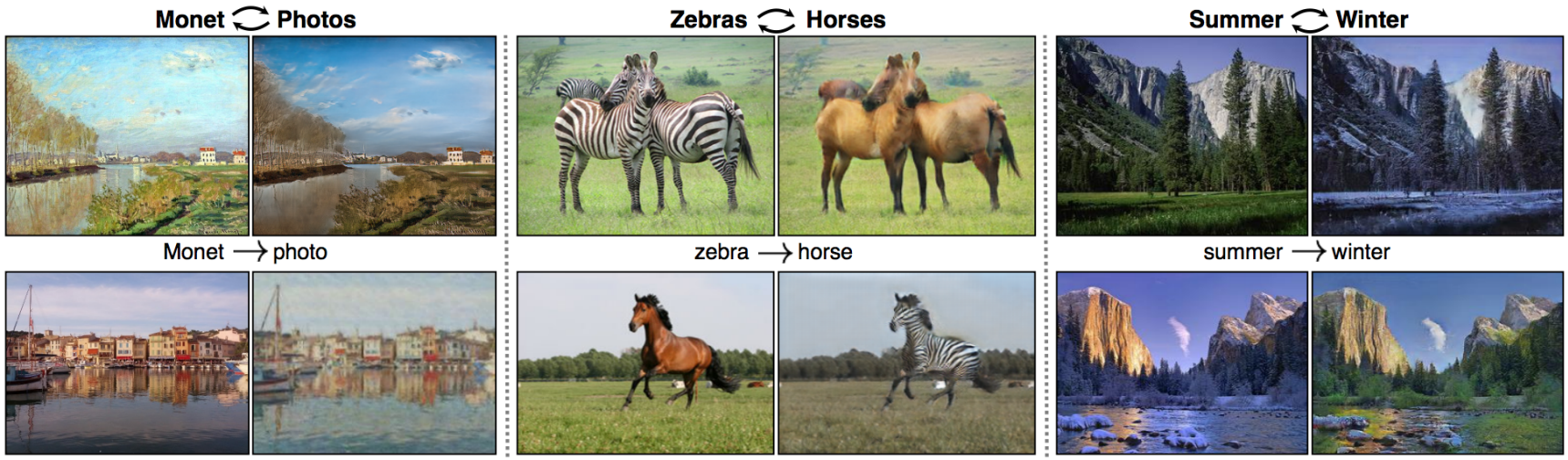
Image-to-Image Translation [5] II

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



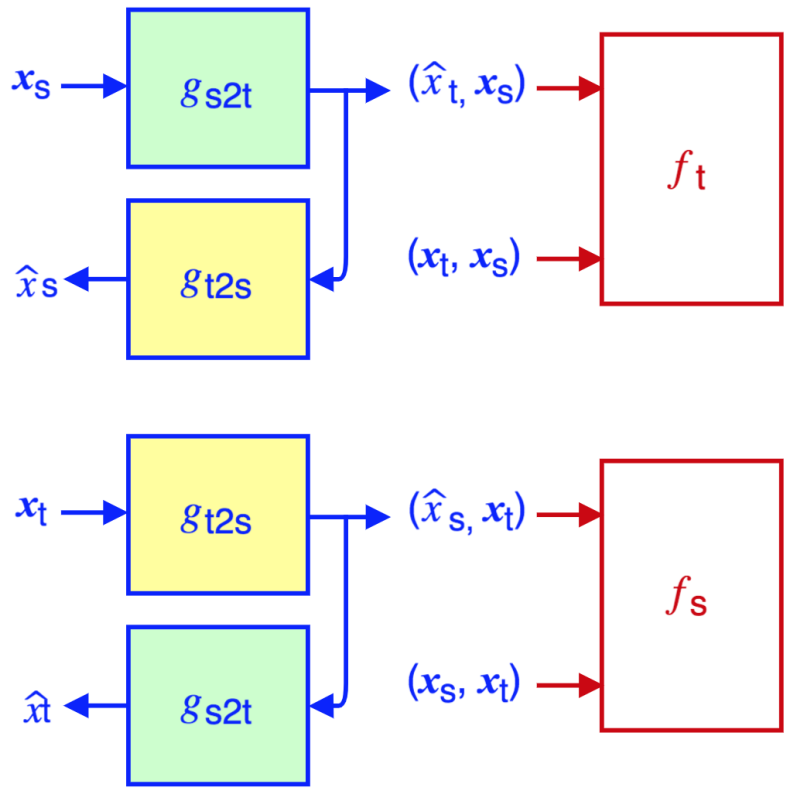
Unpaired Image-to-Image Translation I

- What if the images in different domains are ***unpaired***?
  - $\mathbb{X} = \{\boldsymbol{x}_{\text{src}}^{(n)}\}_n \cup \{\boldsymbol{x}_{\text{target}}^{(n)}\}_n$



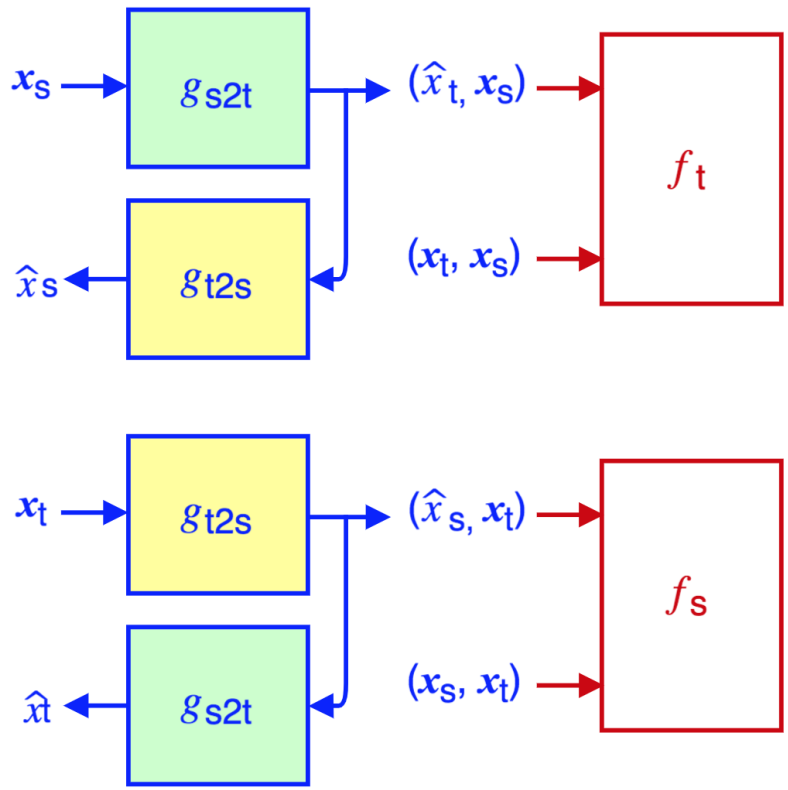
Unpaired Image-to-Image Translation II

- Cycle GAN [29]: to train two generators $g_{\text{src2target}}$ and $g_{\text{target2src}}$ simultaneously in two GANs



Unpaired Image-to-Image Translation II

- Cycle GAN [29]: to train two generators $g_{\text{src2target}}$ and $g_{\text{target2src}}$ simultaneously in two GANs
- Add a loss term $\sum_n \|\boldsymbol{x}_{\text{src}}^{(n)} - g_{\text{target2src}}(g_{\text{src2target}}(\boldsymbol{x}_{\text{src}}^{(n)}))\|_F$ and $\sum_n \|\boldsymbol{x}_{\text{target}}^{(n)} - g_{\text{src2target}}(g_{\text{target2src}}(\boldsymbol{x}_{\text{target}}^{(n)}))\|_F$ for $g_{\text{src2target}}$ and $g_{\text{target2src}}$



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] 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

[4] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville.
Improved training of wasserstein gans.
arXiv preprint arXiv:1704.00028, 2017.

[5] 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.

[6] 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.

[7] 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

[8] Diederik P Kingma and Max Welling.
Auto-encoding variational bayes.
arXiv preprint arXiv:1312.6114, 2013.

[9] Quoc V Le and Tomas Mikolov.
Distributed representations of sentences and documents.
In *ICML*, volume 14, pages 1188–1196, 2014.

[10] 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.

[11] 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

[12] Daniel D Lee and H Sebastian Seung.
Algorithms for non-negative matrix factorization.
In *Advances in neural information processing systems*, pages 556–562, 2001.

[13] William Lotter, Gabriel Kreiman, and David Cox.
Deep predictive coding networks for video prediction and unsupervised learning.
arXiv preprint arXiv:1605.08104, 2016.

[14] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein.
Unrolled generative adversarial networks.
arXiv preprint arXiv:1611.02163, 2016.

[15] 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

[16] 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.

[17] Alec Radford, Luke Metz, and Soumith Chintala.
Unsupervised representation learning with deep convolutional generative adversarial networks.
arXiv preprint arXiv:1511.06434, 2015.

[18] 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

[19] 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.

[20] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton.
Dynamic routing between capsules.
In *Advances in Neural Information Processing Systems*, pages 3857–3867, 2017.

[21] 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

[22] 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.

[23] 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.

[24] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu.
Pixel recurrent neural networks.
In *International Conference on Machine Learning*, pages 1747–1756, 2016.

Reference VIII

[25] 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.

[26] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba.
Anticipating the future by watching unlabeled video.
arXiv preprint arXiv:1504.08023, 2015.

[27] Matthew D Zeiler and Rob Fergus.
Visualizing and understanding convolutional networks.
In *European conference on computer vision*, pages 818–833. Springer, 2014.

Reference IX

[28] 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.

[29] 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.