Theory of Generative Adversarial Nets

Jianping Fan
Department of Computer Science
UNC-Charlotte

Course Website:
http://webpages.uncc.edu/jfan/itcs5152.html

Probabilistic Generative Models

Real world → observations

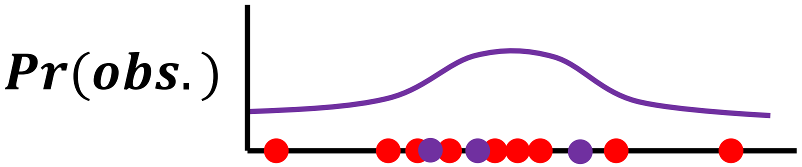
Model → synthetic observations

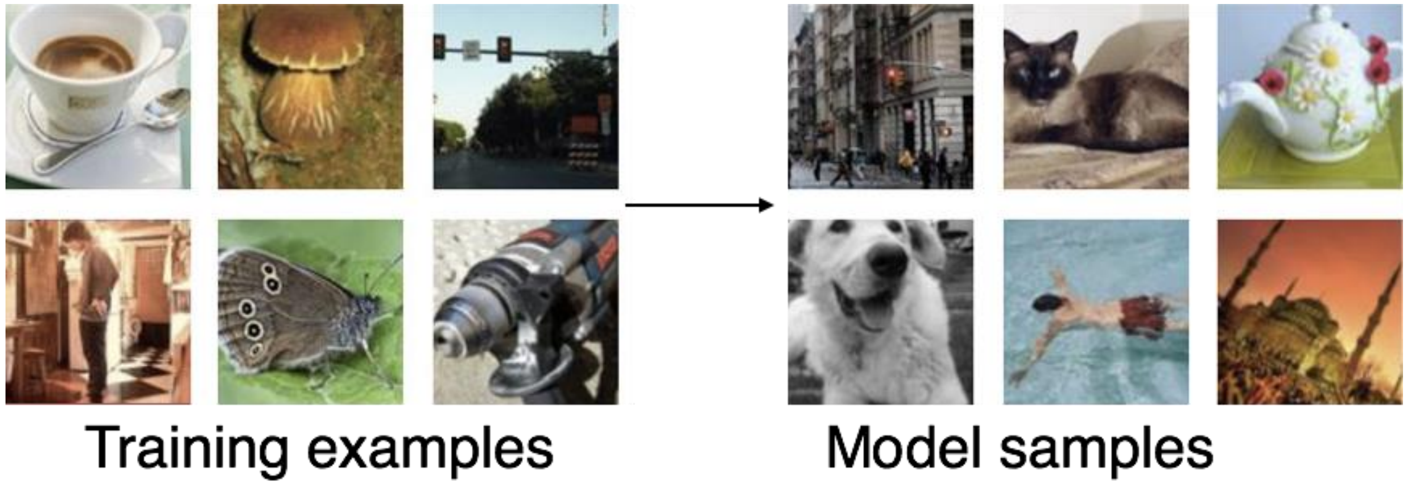
= goal of modeling

Density Estimation

$$Pr(observation) = Pr(synthetic\ obs.)$$

Synthesizing Examples
From Probabilistic Generative Model

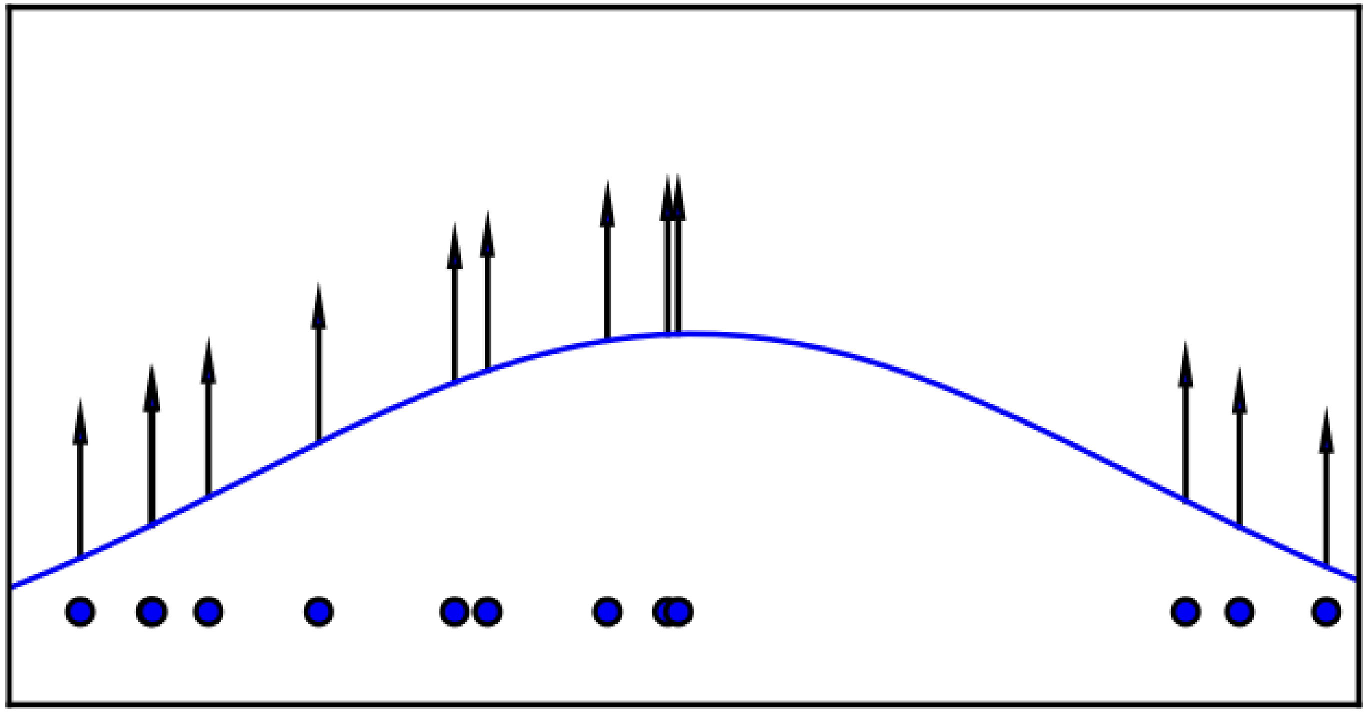




Goodfellow (2017)

Goodfellow NIPS tutorial and accompanying paper (arXiv:1701.00160v4 [cs.LG]) provided some figures

Maximum Likelihood Estimation



$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x\sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

Goodfellow (2017)

Density function $Pr_{\mathrm{model}}(x|\theta)$

Explicit and analytical

- e.g., Gaussian
- can sample directly from model

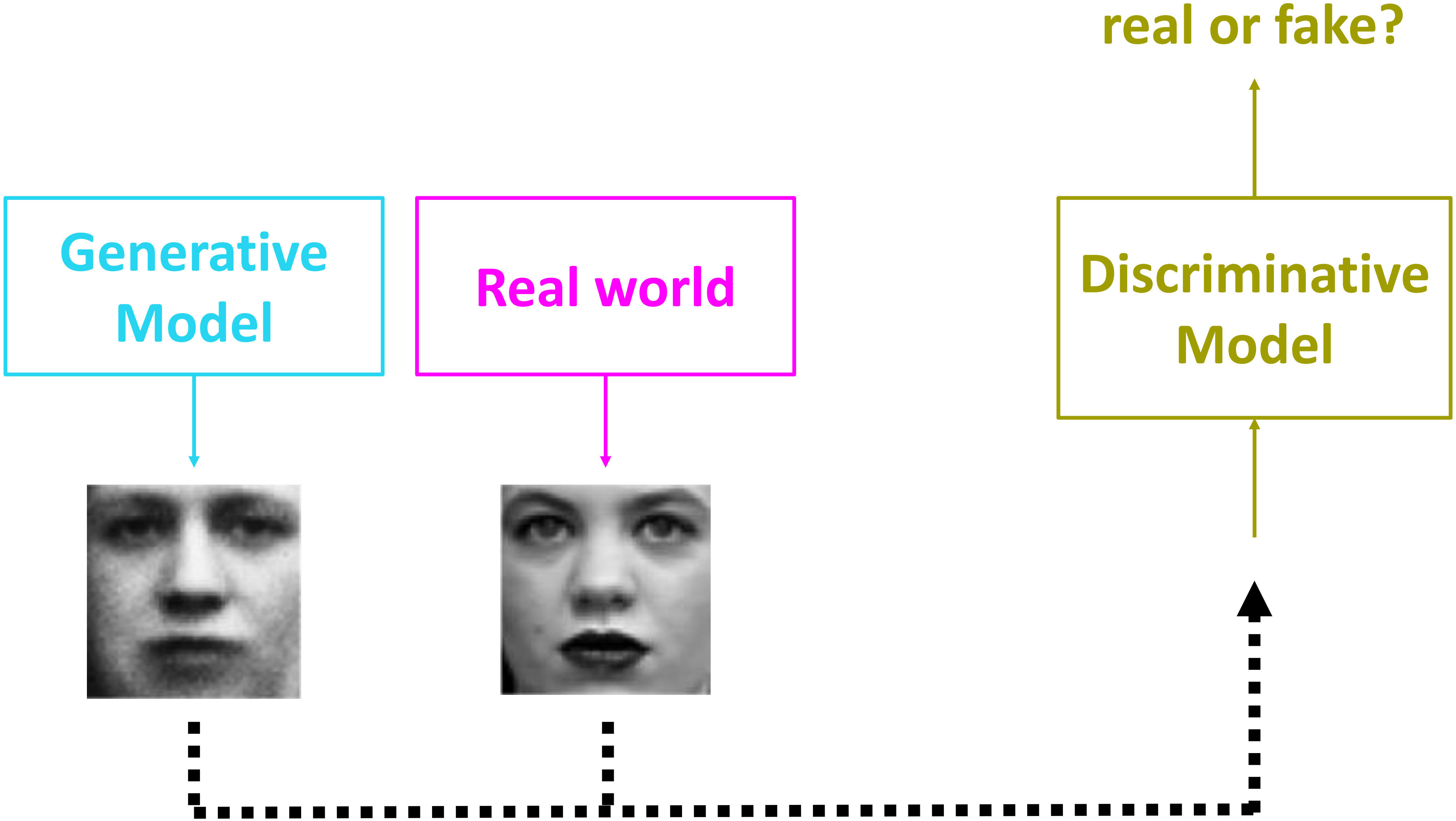
Explicit and approximate

- e.g., Boltzmann machine
- can estimate probability by running Markov chain monte carlo

Implicit

- GAN
- can’t estimate probability but can draw from distribution with given probability

Adversarial Networks



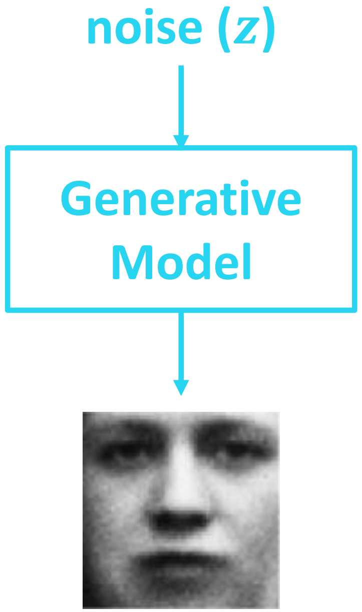
Generative Model

How to make it generate different samples each time it is run?

- input to model is noise

Generative model as a neural network

- computes $x = G(z|\theta)$
- differentiable
- does not have to be invertible
- z typically has very high dimensionality (higher than x)



Discriminative Model

Think of it as a critic

- a good critic can tell real from fake

Discriminative model as a neural net

- differentiable
- computes $D(x)$, with value 1 if real, 0 if fake

Training Procedure: Basic Idea

G tries to fool D

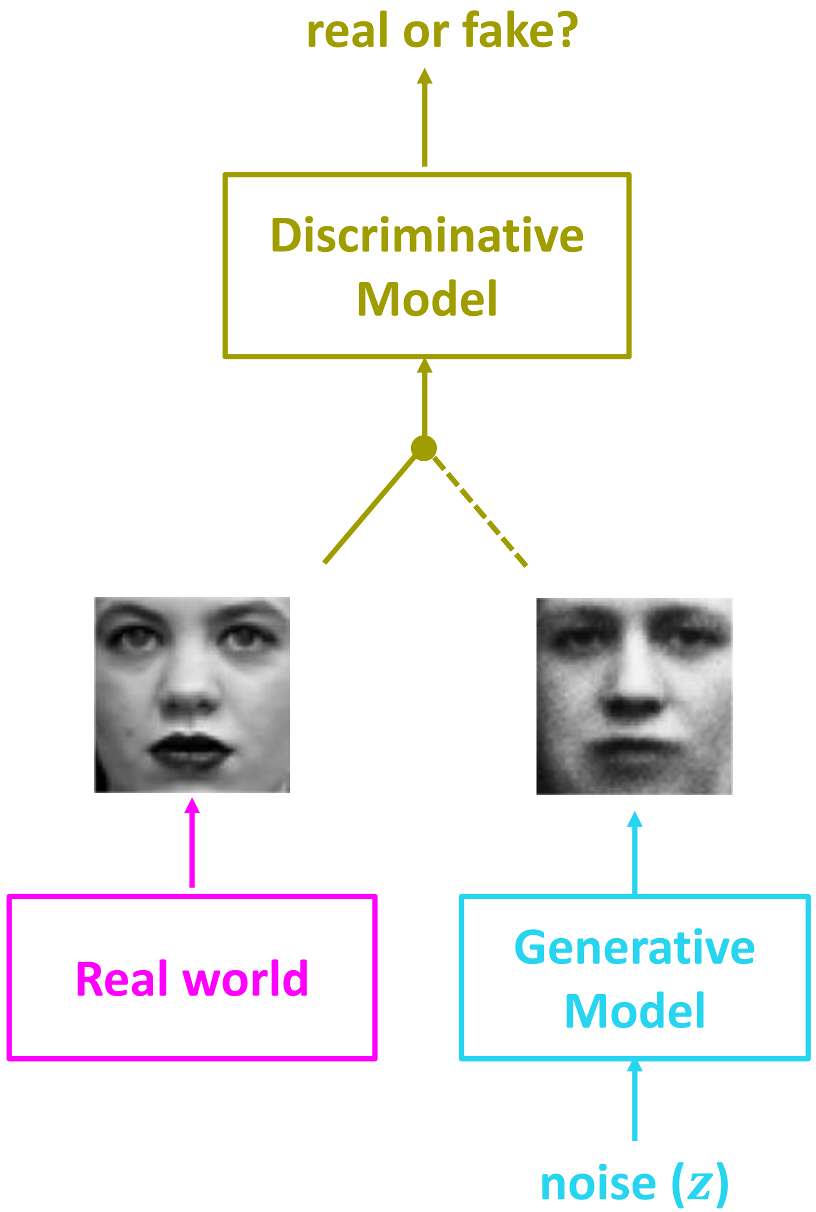
D tries not to be fooled

Models are trained simultaneously

- As G gets better, D has a more challenging task
- As D gets better, G has a more challenging task

Ultimately, we don't care about the D

- Its role is to force G to work harder



Loss Functions

Loss function for D

- maximize the likelihood that model says 'real' to samples from the world and 'fake' to generated samples
- $\mathcal{L}_D = -\frac{1}{2}\mathbb{E}_{x\sim\text{world}} \ln D(x) - \frac{1}{2}\mathbb{E}_z \ln\left(1 - D(G(z))\right)$

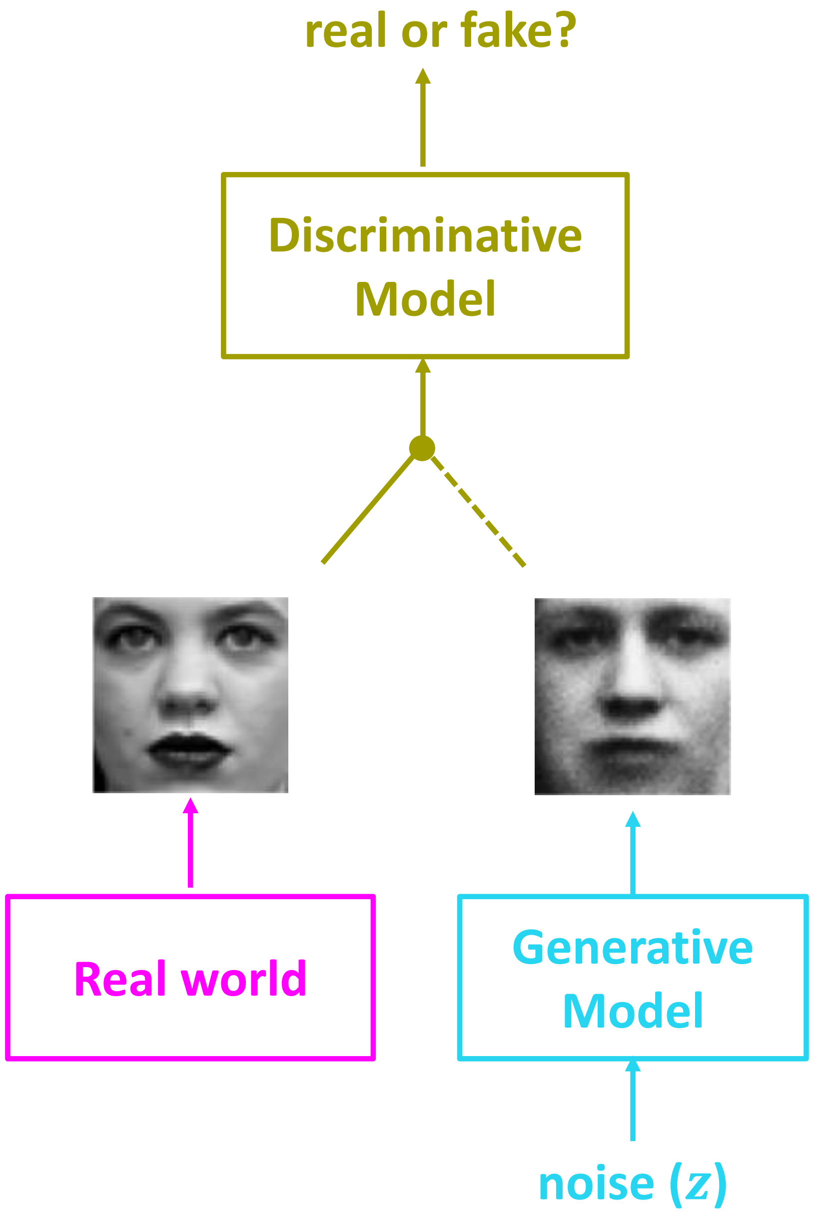
What should the loss function be for G?

- $\mathcal{L}_G = -\mathcal{L}_D$

But because first term doesn't matter for G (why?)

- $\mathcal{L}_D = \frac{1}{2}\mathbb{E}_z \ln\left(1 - D(G(z))\right)$

Known as a minimax procedure



Training Procedure

Train both models simultaneously via stochastic gradient descent using minibatches consisting of

- some generated samples
- some real-world samples

Training of D is straightforward

Error for G comes via back propagation through D

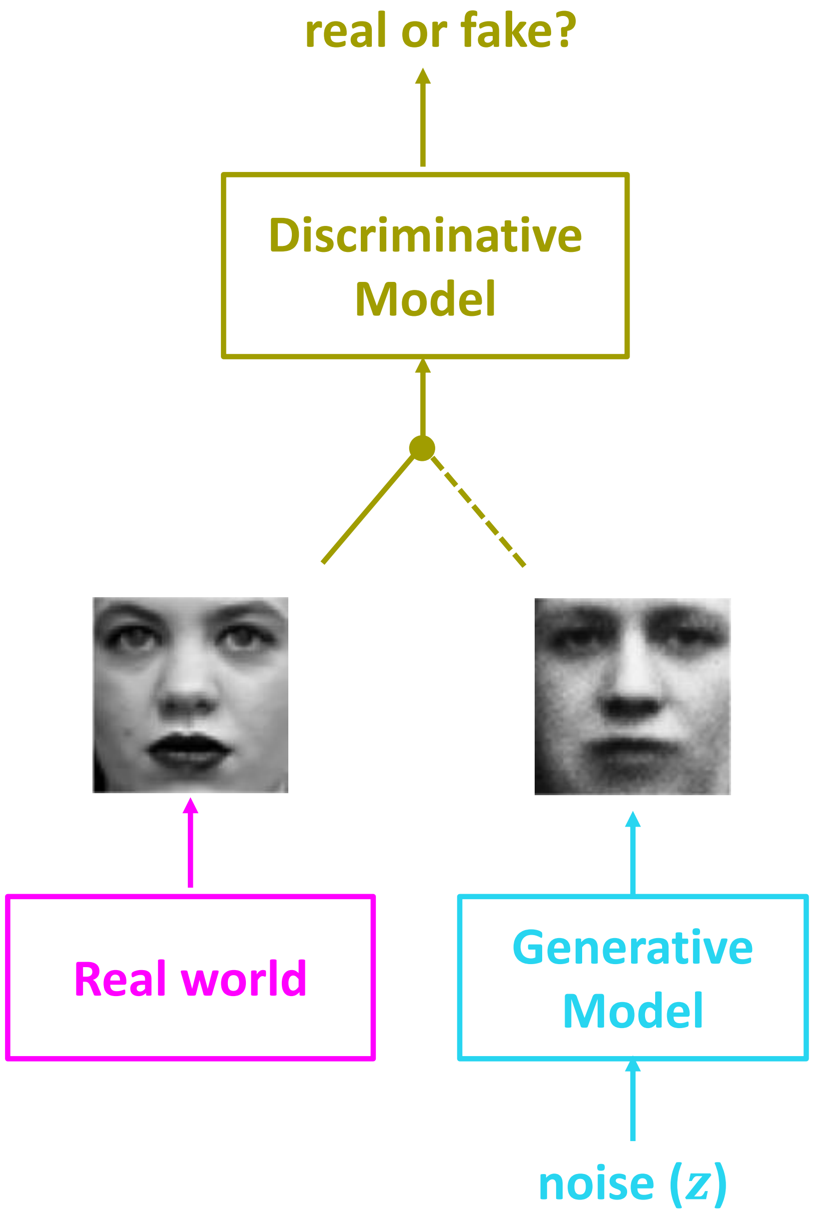
- Two ways to think about training

  (1) freeze D weights and propagate $\mathcal{L}_G$ through D to determine $\partial \mathcal{L}_G / \partial x$

  (2) Compute $\partial \mathcal{L}_D / \partial x$ and then $\partial \mathcal{L}_G / \partial x = -\partial \mathcal{L}_D / \partial x$

D can be trained without altering G, and vice versa

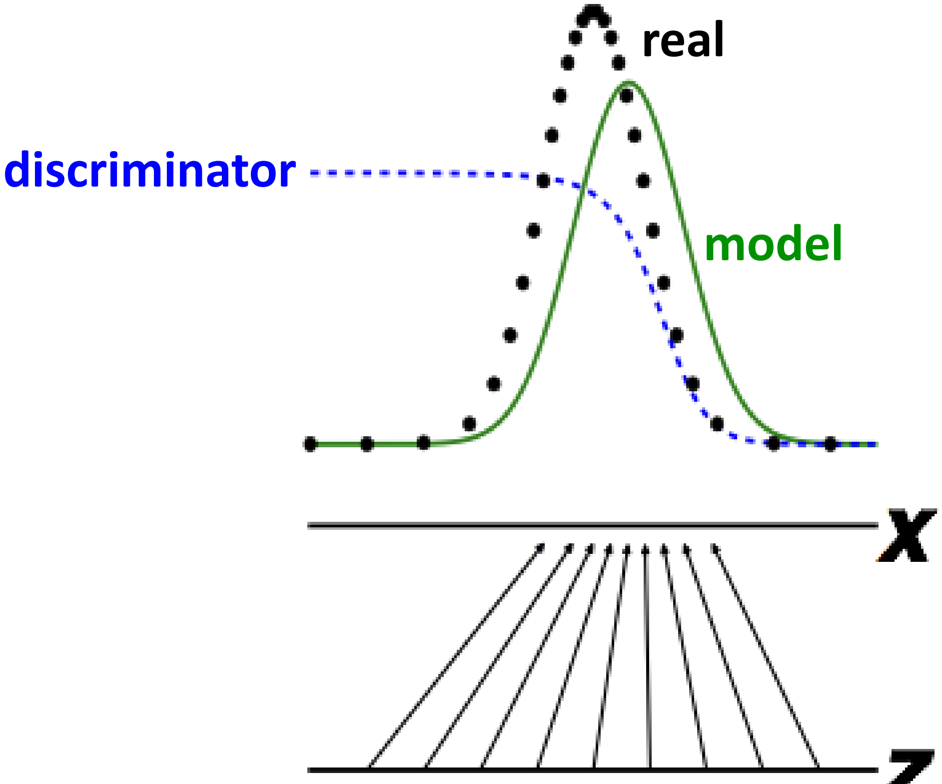
- May want multiple training epochs of just D so it can stay ahead
- May want multiple training epochs of just G because it has a harder task

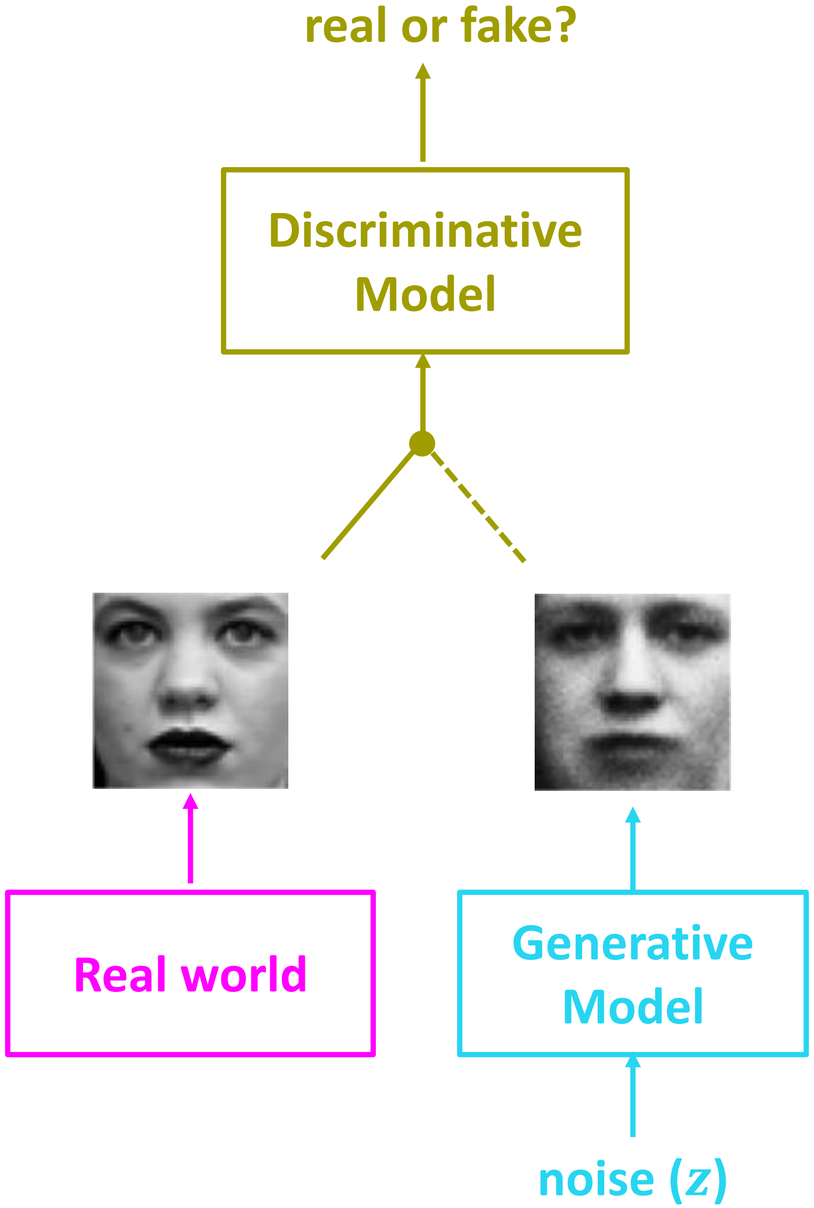


The Discriminator Has a Straightforward Task

D has learned when

- $D(x) = Pr(\text{real}|x) = \dfrac{Pr(x|\text{real})}{Pr(x|\text{real})+Pr(x|\text{synthesized})}$





Goodfellow (2017)

Three Reasons That It's a Miracle GANs Work

G has a reinforcement learning task

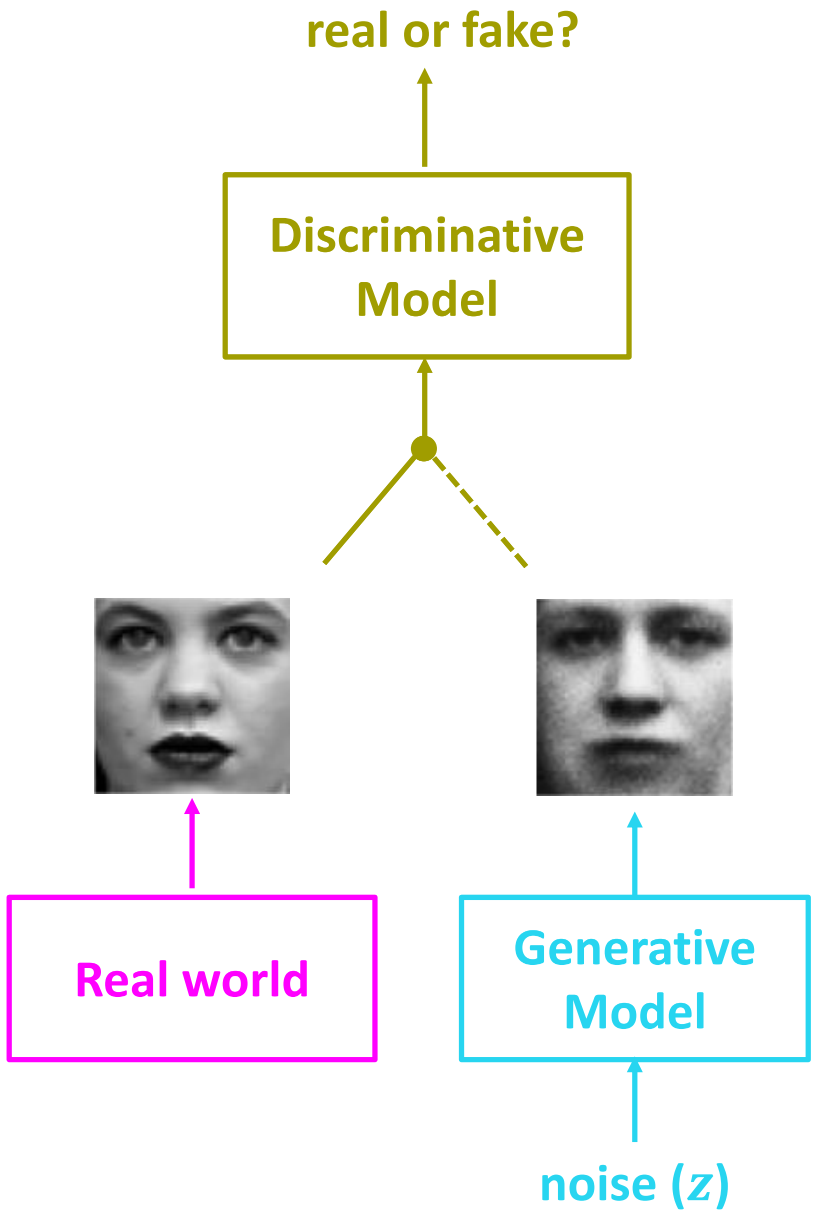
- it knows when it does good (i.e., fools D) but it is not given a supervised signal
- reinforcement learning is hard
- back prop through D provides G with a supervised signal; the better D is, the better this signal will be

Can't describe optimum via a single loss

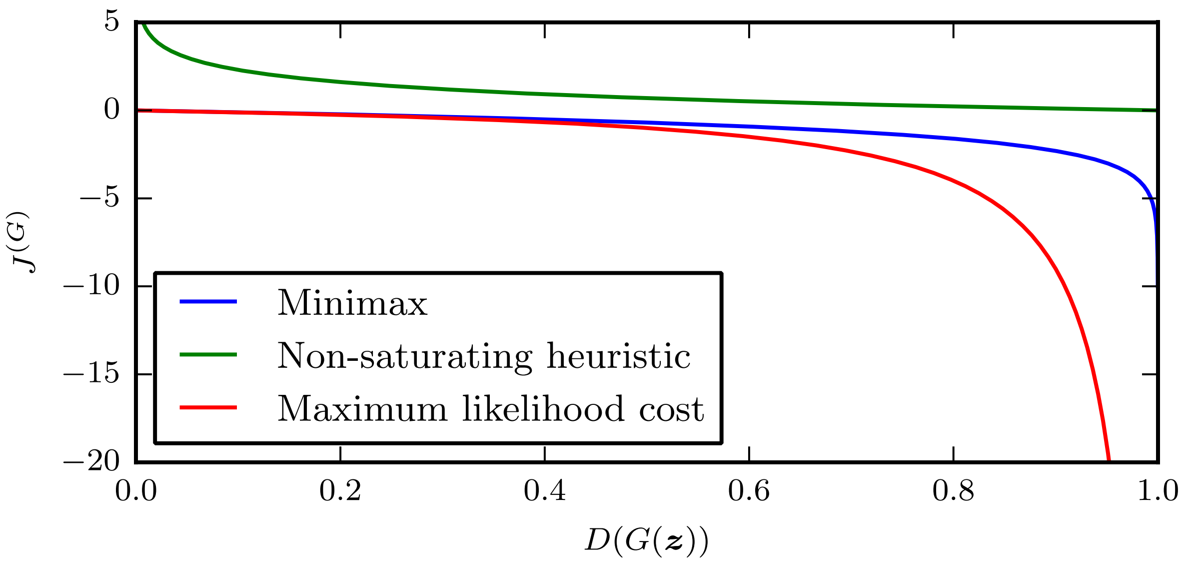
- Will there be an equilibrium?

D is seldom fooled

- but G still learns because it gets a gradient telling it how to change in order to do better the next round.

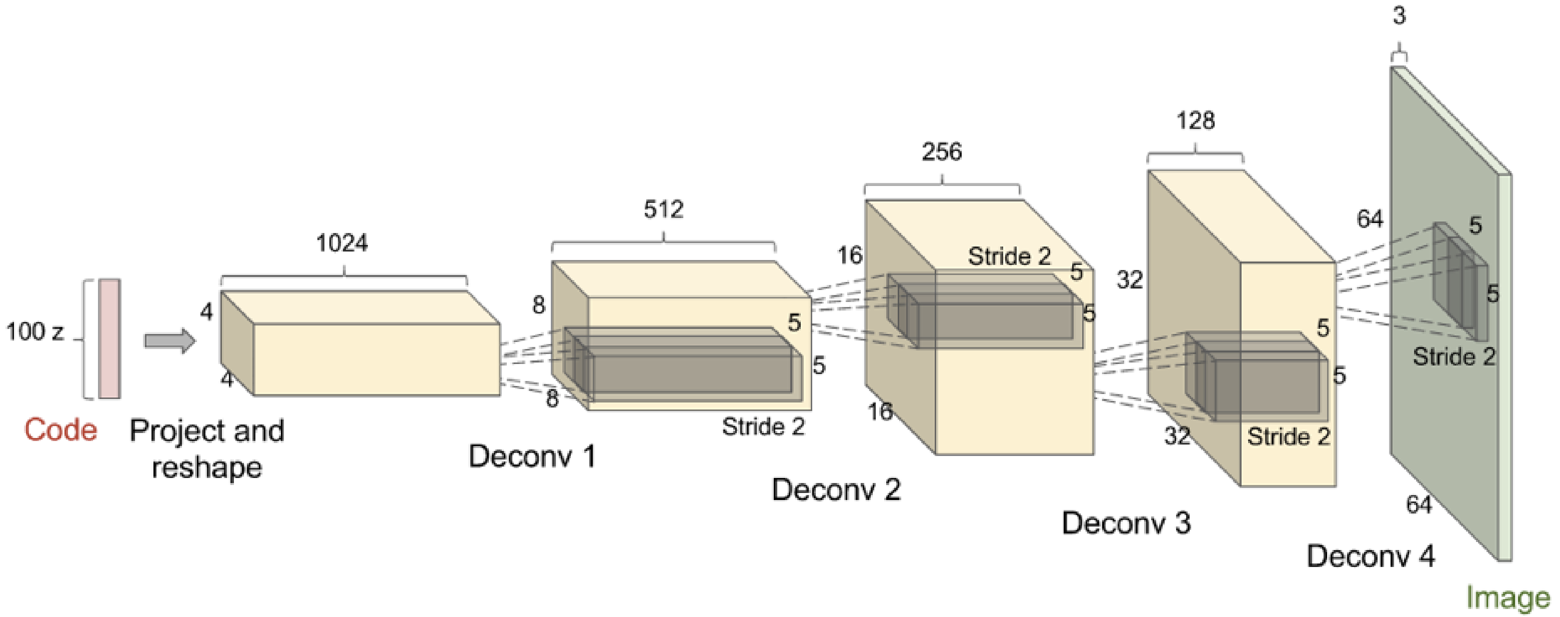


Do Generator Losses Matter?
(Goodfellow, 2014)



All losses seem to produce sharp samples

Deconvolutional GANs (DCGAN)
(Radford et al., 2015)

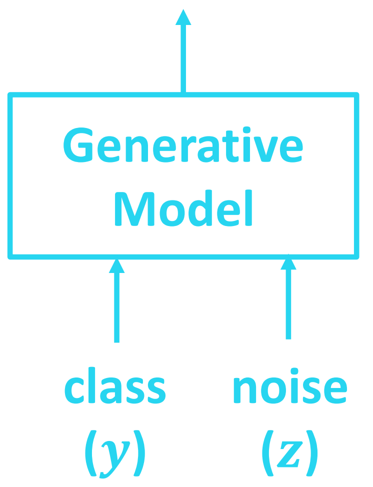


Batch normalization important here, apparently

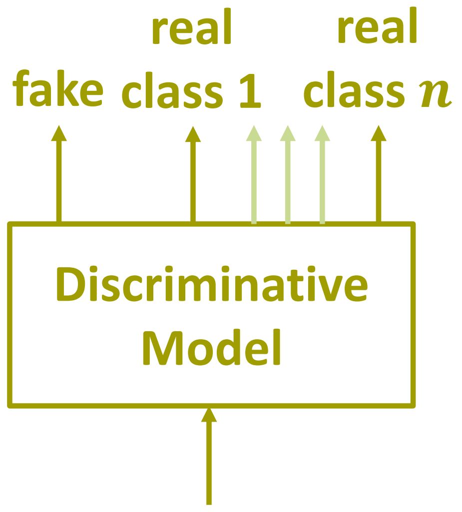
Goodfellow (2017)

Using Labels Can Improve Generated Samples

Denton et al. (2015)

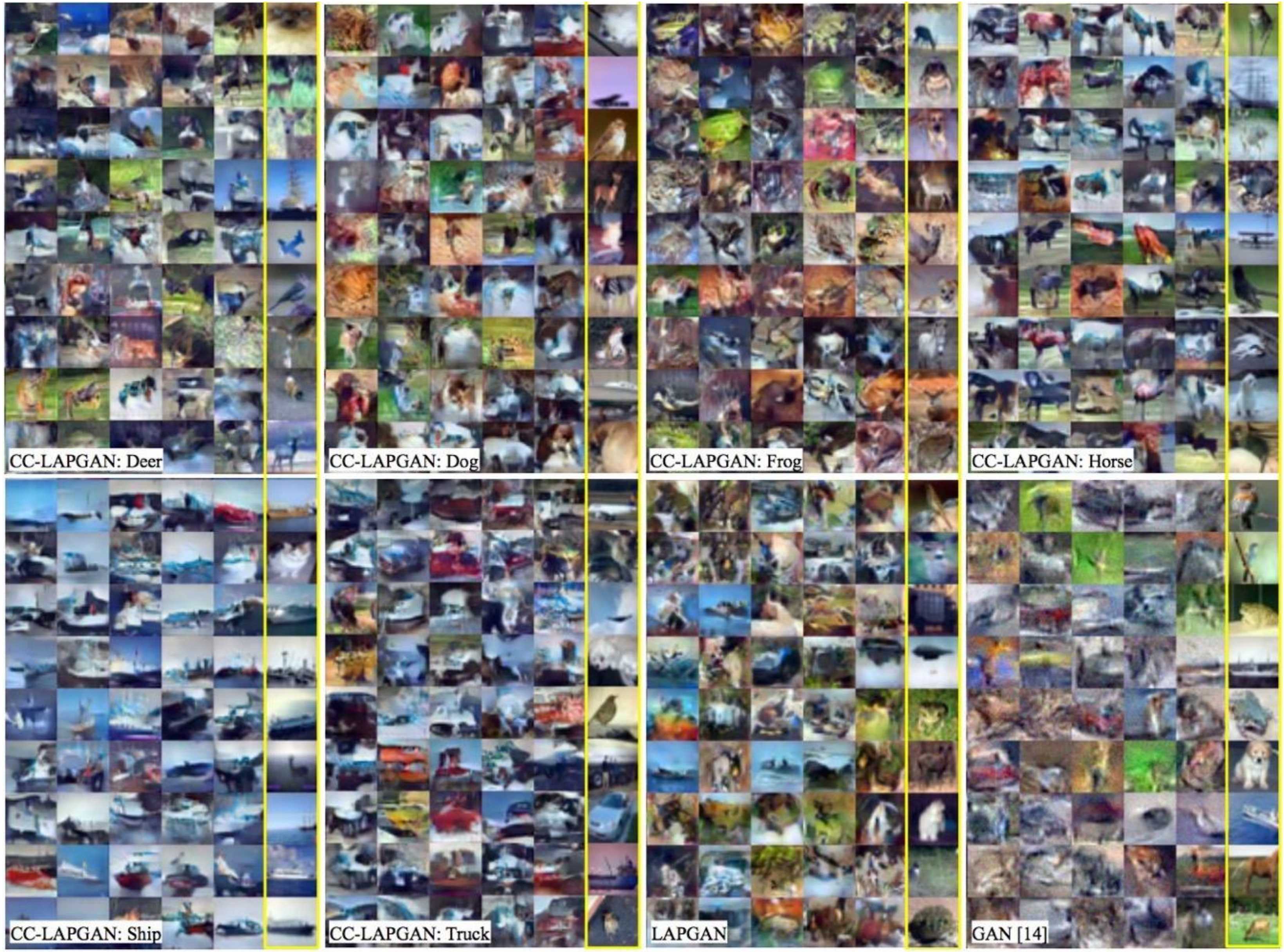


Salimans et al. (2016)



Using Labels Can Improve Generated Samples

(Denton et al., 2015)

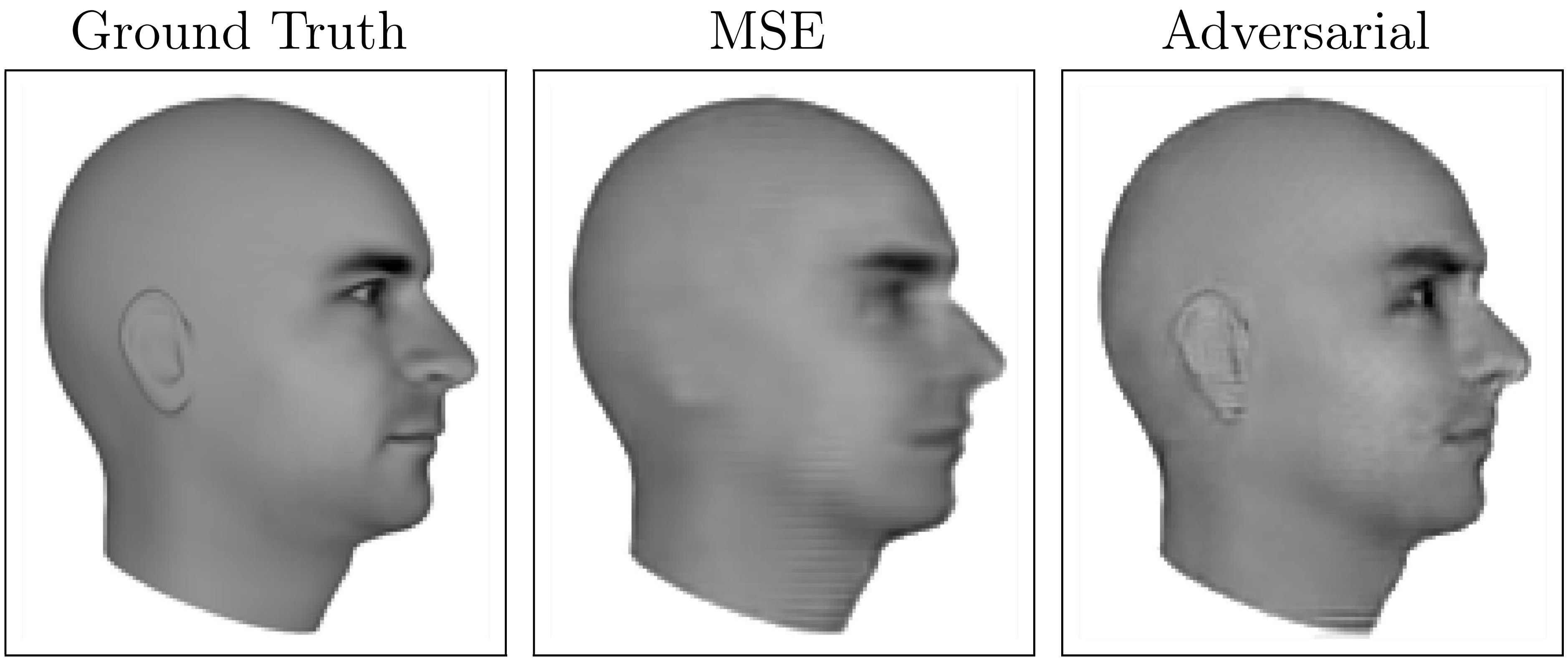


LAPGAN: multiresolution deconvolutional pyramid

CC-LAPGAN: class conditional LAPGAN

GAN: original Goodfellow model

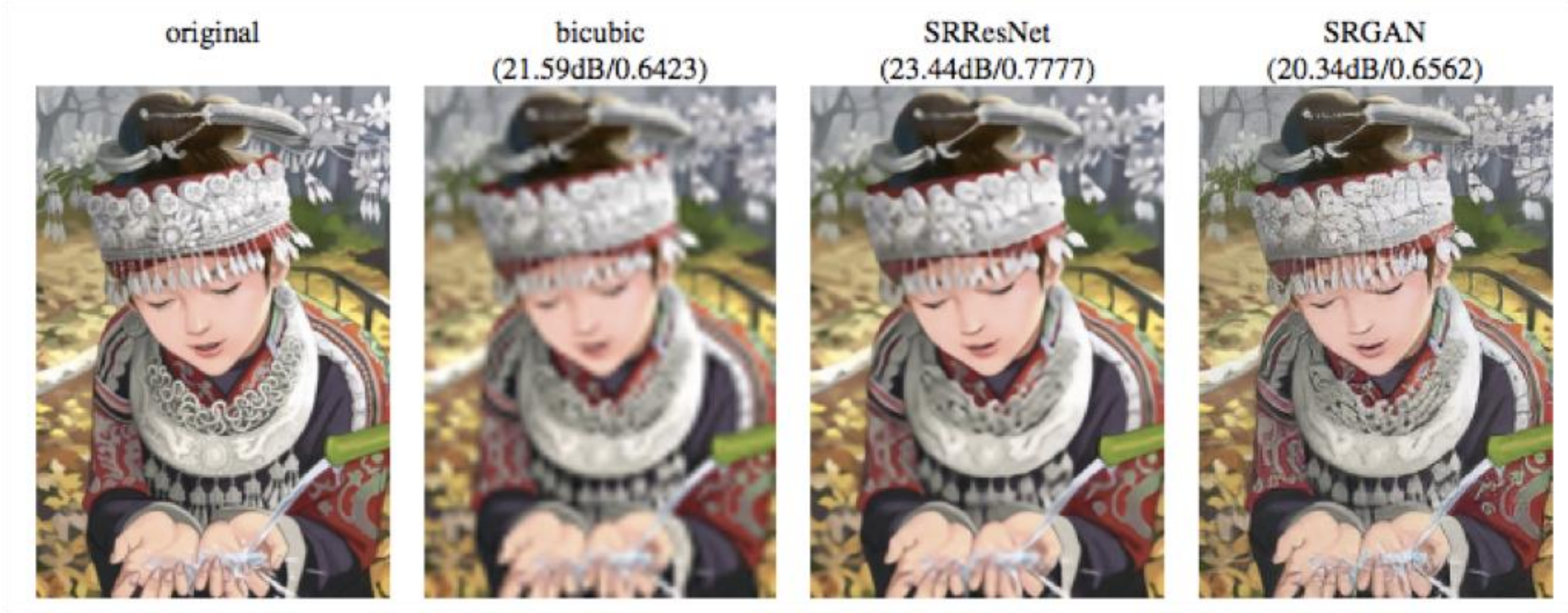
Beyond Labels: Providing Images as Input to Generator: Next Video Frame Prediction (Lotter et al., 2016)



MSE tends to produce blurry images on any task

- when you can't predict well, predict the expectation

Beyond Labels: Providing Images as Input to Generator: Image Super-Resolution (Ledig et al., 2016)



Visually-Aware Fashion Recommendation and Design With GANs (Kang et al., 2017)

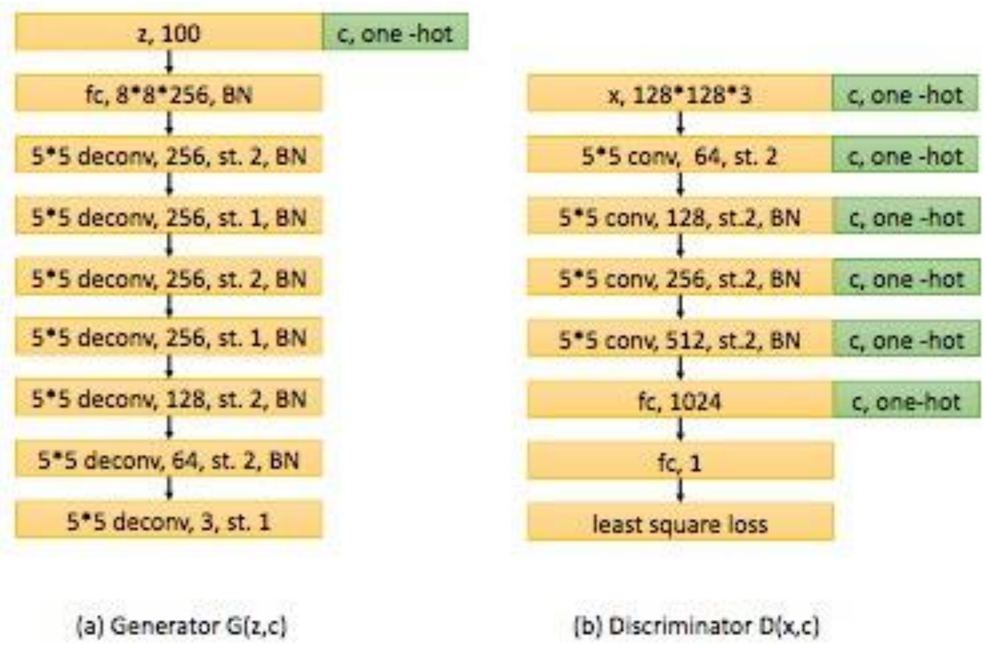
Recommender systems predict how much a particular user will like a particular item

- **Can predict based on features of item (e.g., movie director, dress length)**
- **Can also predict directly from images**

Twist here is that instead of predicting from a predefined set, *generate images* that would be liked.

Visually-Aware Fashion Recommendation and Design With GANs (Kang et al., 2017)

Use class-conditional generator and discriminator



Visually-Aware Fashion Recommendation and Design With GANs (Kang et al., 2017)

Optimize with GAN

- find latent representation z that obtains the highest recommendation score
- gradient ascent search

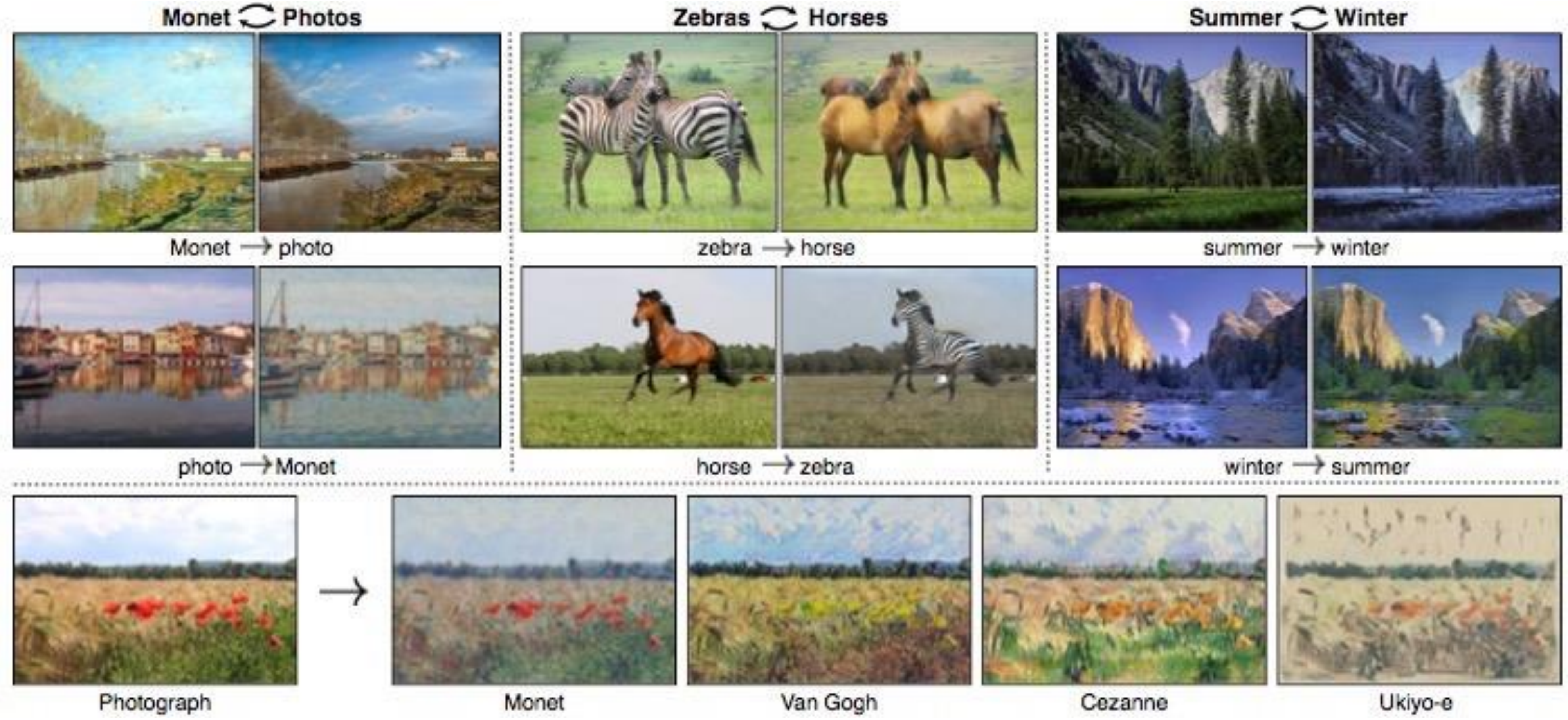

(a) Top-3 Results from Dataset
(b) Top-3 Results from GAN
12.29 11.79 11.76 12.89 12.56 12.67
8.07 8.06 7.81 8.14 8.00 7.37
7.07 6.78 6.70 9.49 9.34 8.56
13.28 12.75 12.51 15.05 13.93 13.74
4.27 4.21 4.20 5.37 5.20 4.46
10.28 10.27 10.15 12.67 11.87 10.94

Cycle GANs

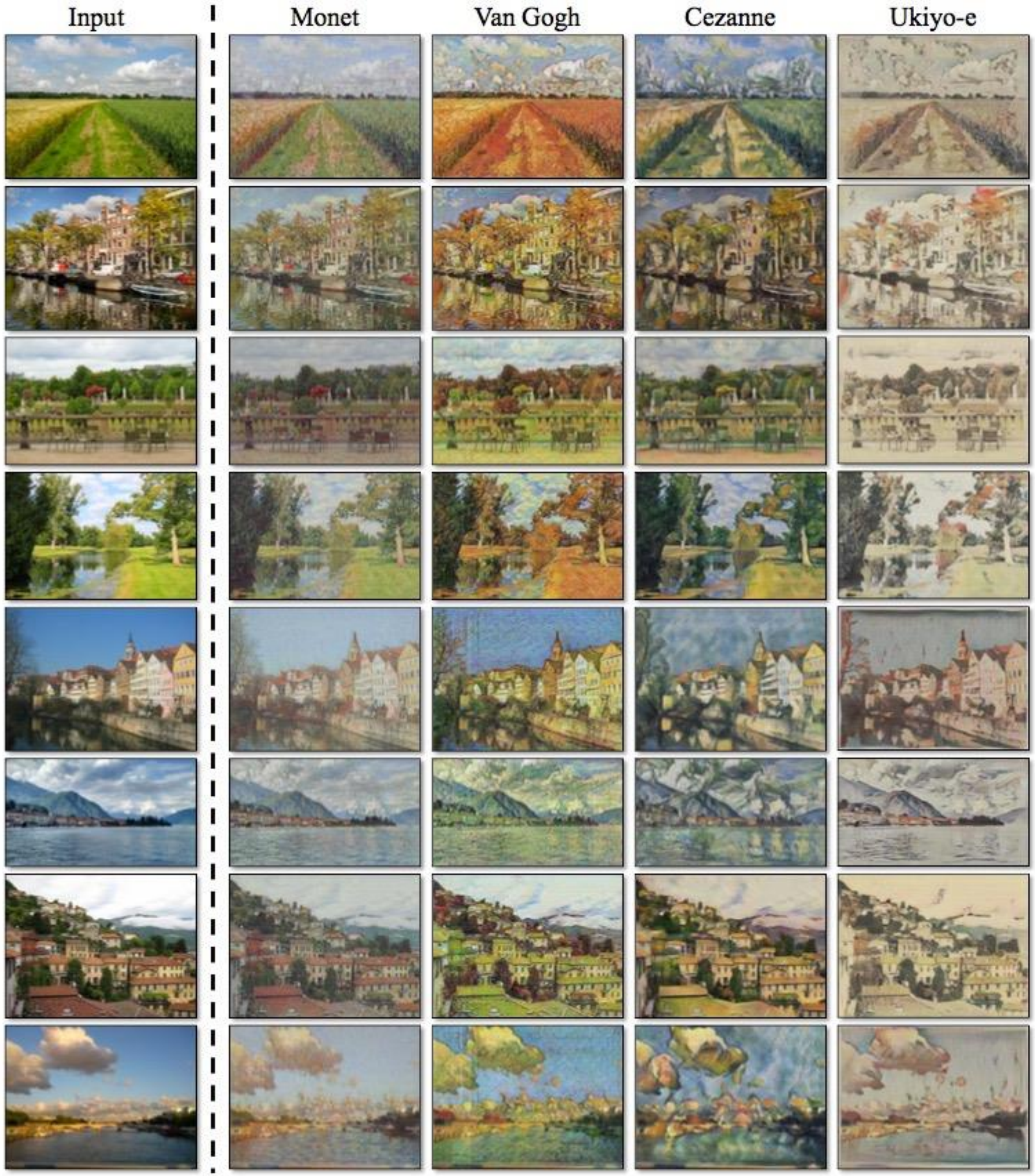
(Zhu et al., 2017; arXiv:1703:10593v2 [cs.CV])

Given two image collections

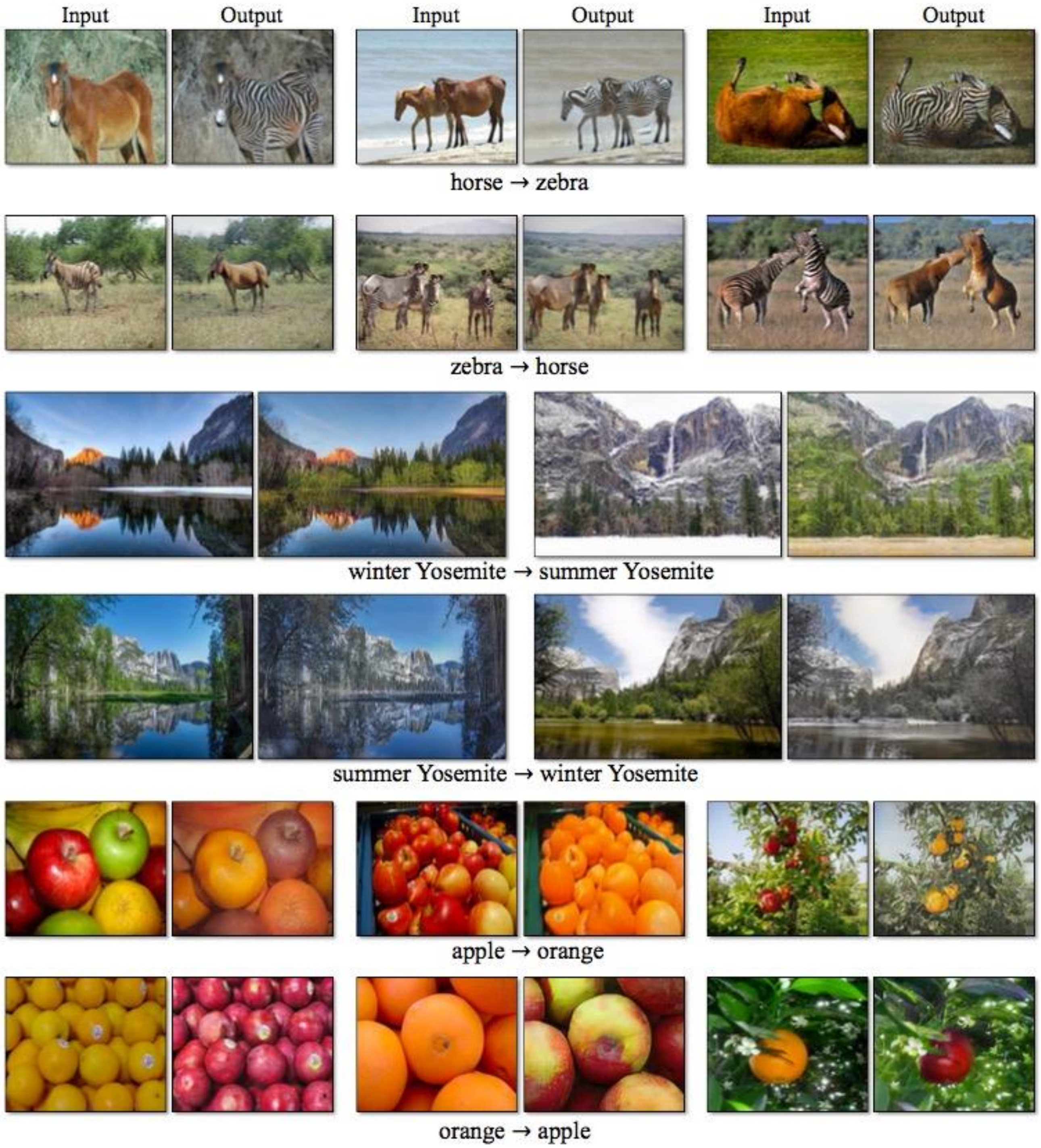
- **algorithm learns to translate an image from one collection to the other**
- **does not require corres-pondence between images**



Photos to paintings

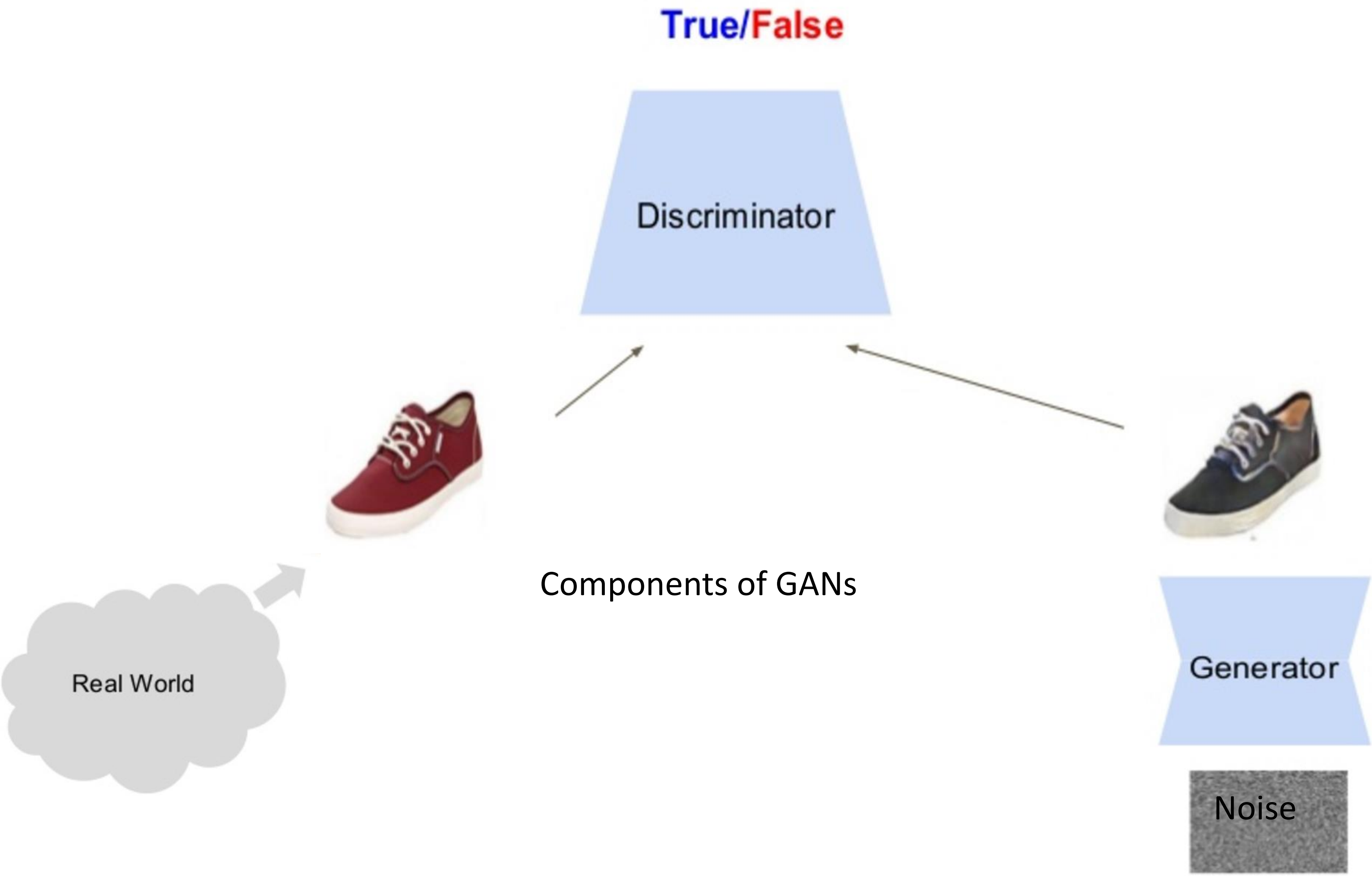


Paintings to photos


Input
Output
Input
Output
Input
Output
horse → zebra
zebra → horse
winter Yosemite → summer Yosemite
summer Yosemite → winter Yosemite
apple → orange
orange → apple

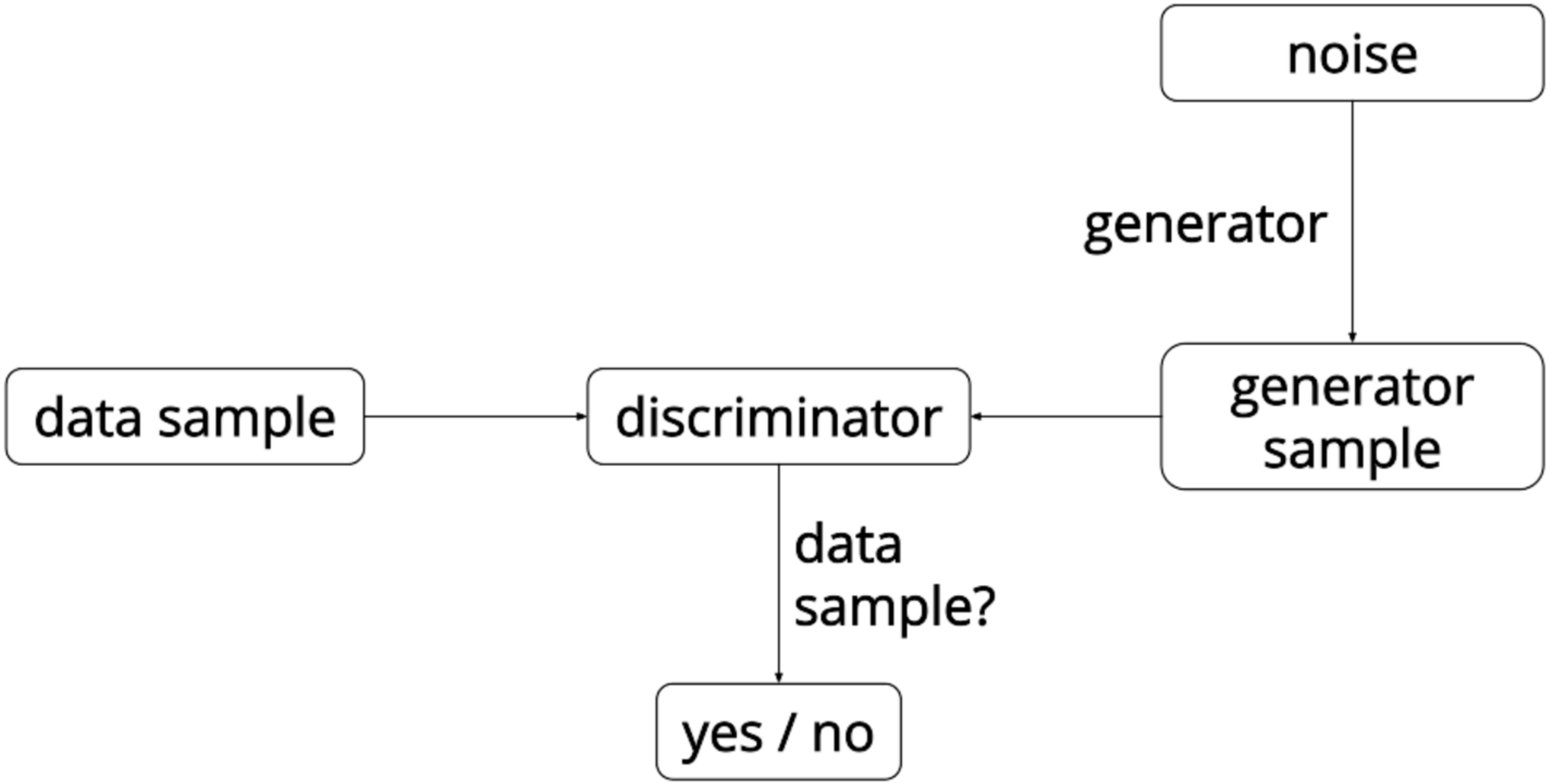
Star GAN

High resolution image synthesis



Components of GANs

Slide credit – Victor Garcia



Overview of GANs
Source: https://ishmaelbelghazi.github.io/ALI

Discriminative Models

A discriminative model learns a function that maps the input data (*x*) to some desired output class label (*y*).

In probabilistic terms, they directly learn the conditional distribution $P(y|x)$.

Generative Models

A generative model tries to learn the joint probability of the input data and labels simultaneously i.e. *$P(x,y)$*.

Potential to understand and explain the underlying structure of the input data even when there are no labels.

How GANs are being used?

Applied for modelling natural images.

Performance is fairly good in comparison to other generative models.

Useful for unsupervised learning tasks.

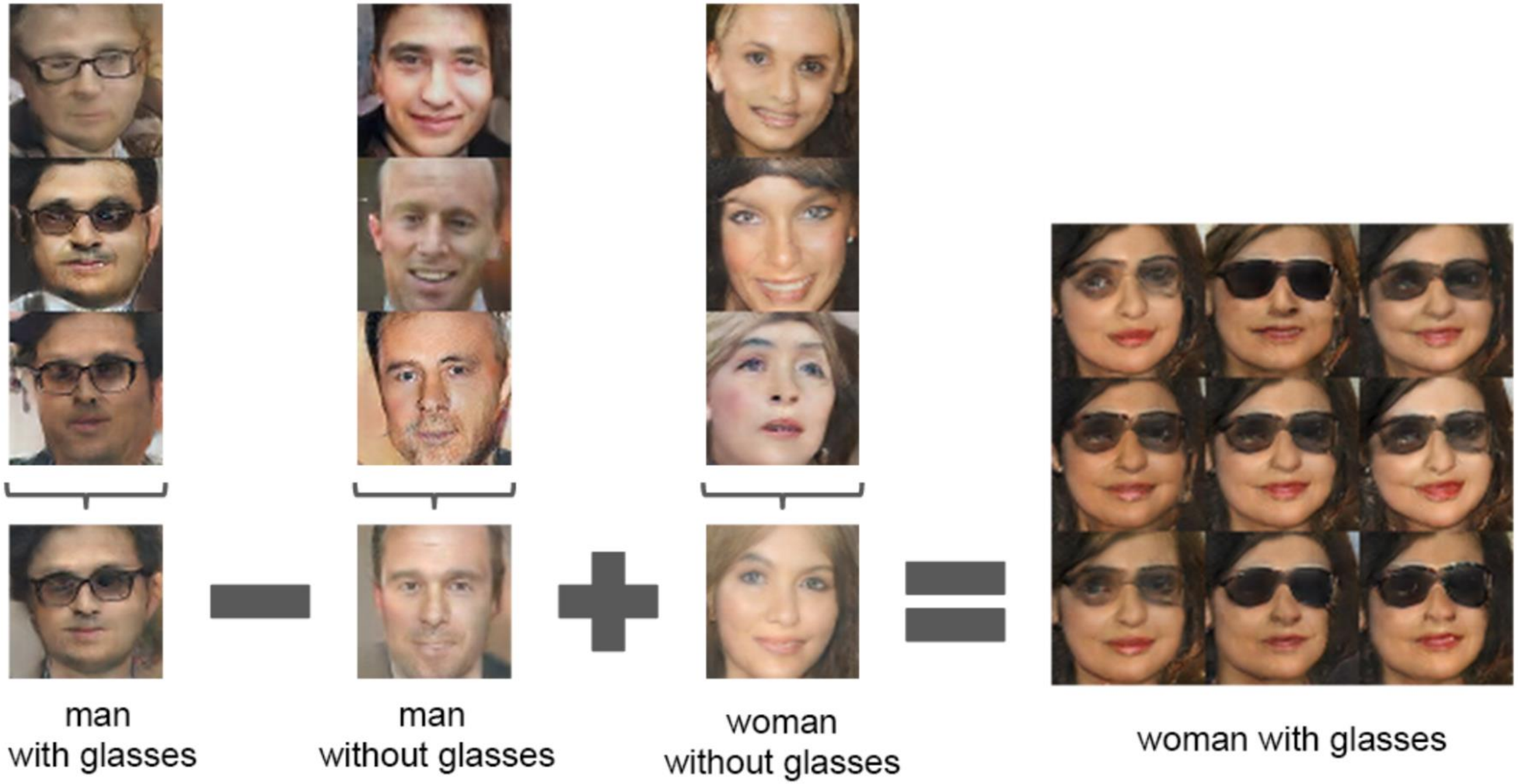
Why GANs?

Use a latent code.

Asymptotically consistent (unlike variational methods) .

No Markov chains needed.

Often regarded as producing the best samples.


man
with glasses
man
without glasses
woman
without glasses
woman with glasses

How to train GANs?

Objective of generative network - increase the error rate of the discriminative network.

Objective of discriminative network – decrease binary classification loss.

Discriminator training - backprop from a binary classification loss.

Generator training - backprop the negation of the binary classification loss of the discriminator.

Loss Functions

$$\mathcal{L}(\hat{x}) = \min_{x \in data} (x - \hat{x})^2$$

Generator

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

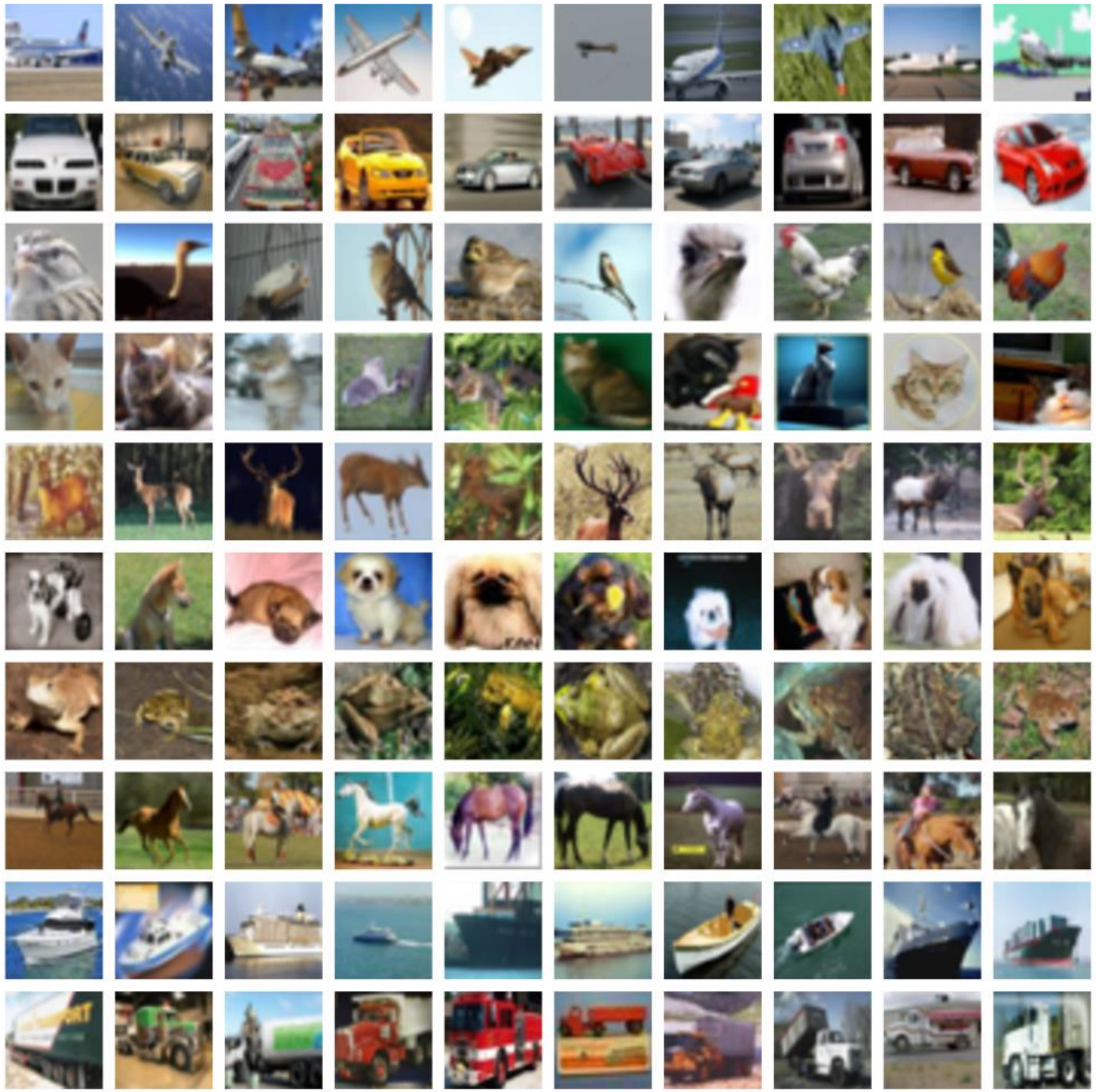
Discriminator

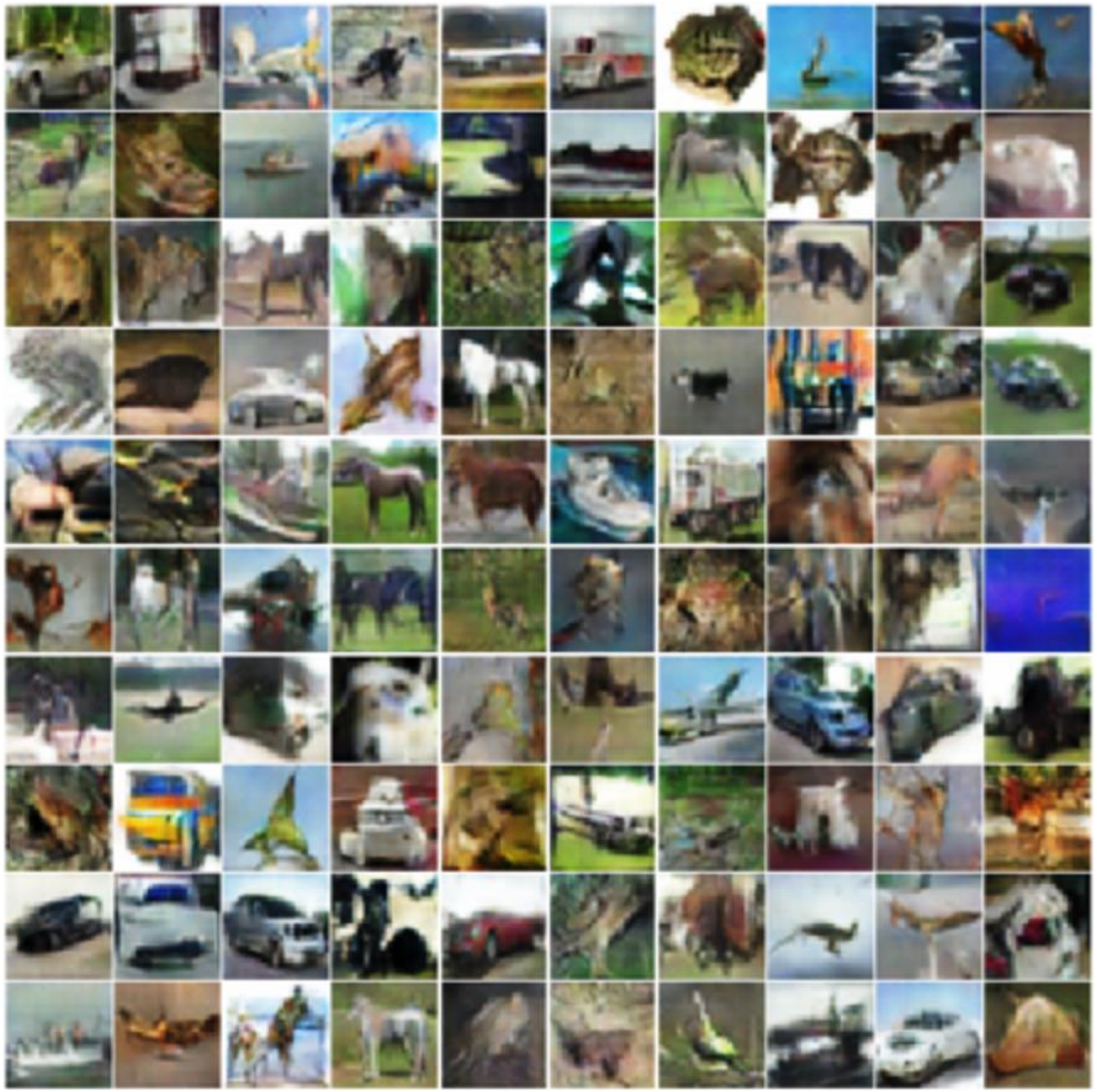
Generated bedrooms. Source: "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" https://arxiv.org/abs/1511.06434v2

“Improved Techniques for Training GANs” by Salimans et. al

One-sided Label smoothing - replaces the 0 and 1 targets for a classifier with smoothed values, like .9 or .1 to reduce the vulnerability of neural networks to adversarial examples.

Virtual batch Normalization - each example x is normalized based on the statistics collected on a reference batch of examples that are chosen once and fixed at the start of training, and on x itself.





Original CIFAR-10 vs. Generated CIFAR-10 samples
Source: "Improved Techniques for Training GANs" https://arxiv.org/abs/1606.03498

Variations to GANs

Several new concepts built on top of GANs have been introduced –

- **InfoGAN – Approximate the data distribution and learn interpretable, useful vector representations of data.**
- **Conditional GANs - Able to generate samples taking into account external information (class label, text, another image). Force *G* to generate a particular type of output.**

Major Difficulties

Networks are difficult to converge.

Ideal goal – Generator and discriminator to reach some desired equilibrium but this is rare.

GANs are yet to converge on large problems (E.g. Imagenet).

Common Failure Cases

The discriminator becomes too strong too quickly and the generator ends up not learning anything.

The generator only learns very specific weaknesses of the discriminator.

The generator learns only a very small subset of the true data distribution.

So what can we do?

Normalize the inputs

A modified loss function

Use a spherical Z

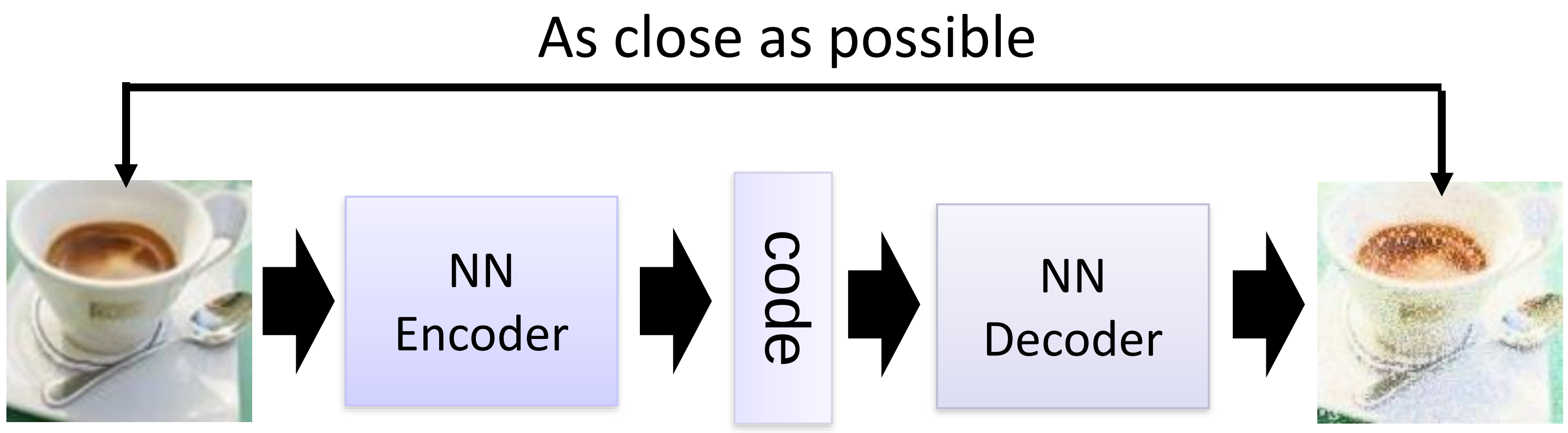
BatchNorm

Avoid Sparse Gradients: ReLU, MaxPool

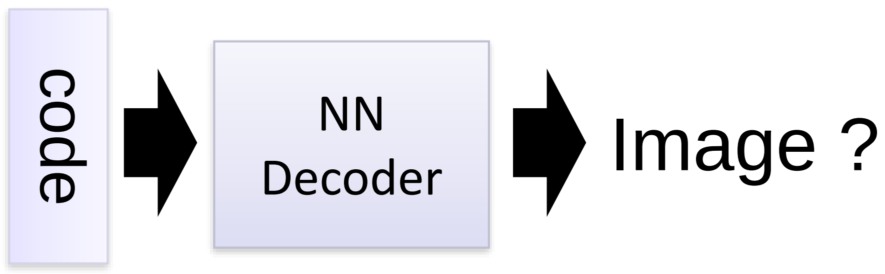
Use Soft and Noisy Labels

DCGAN / Hybrid Models

Autoencoder

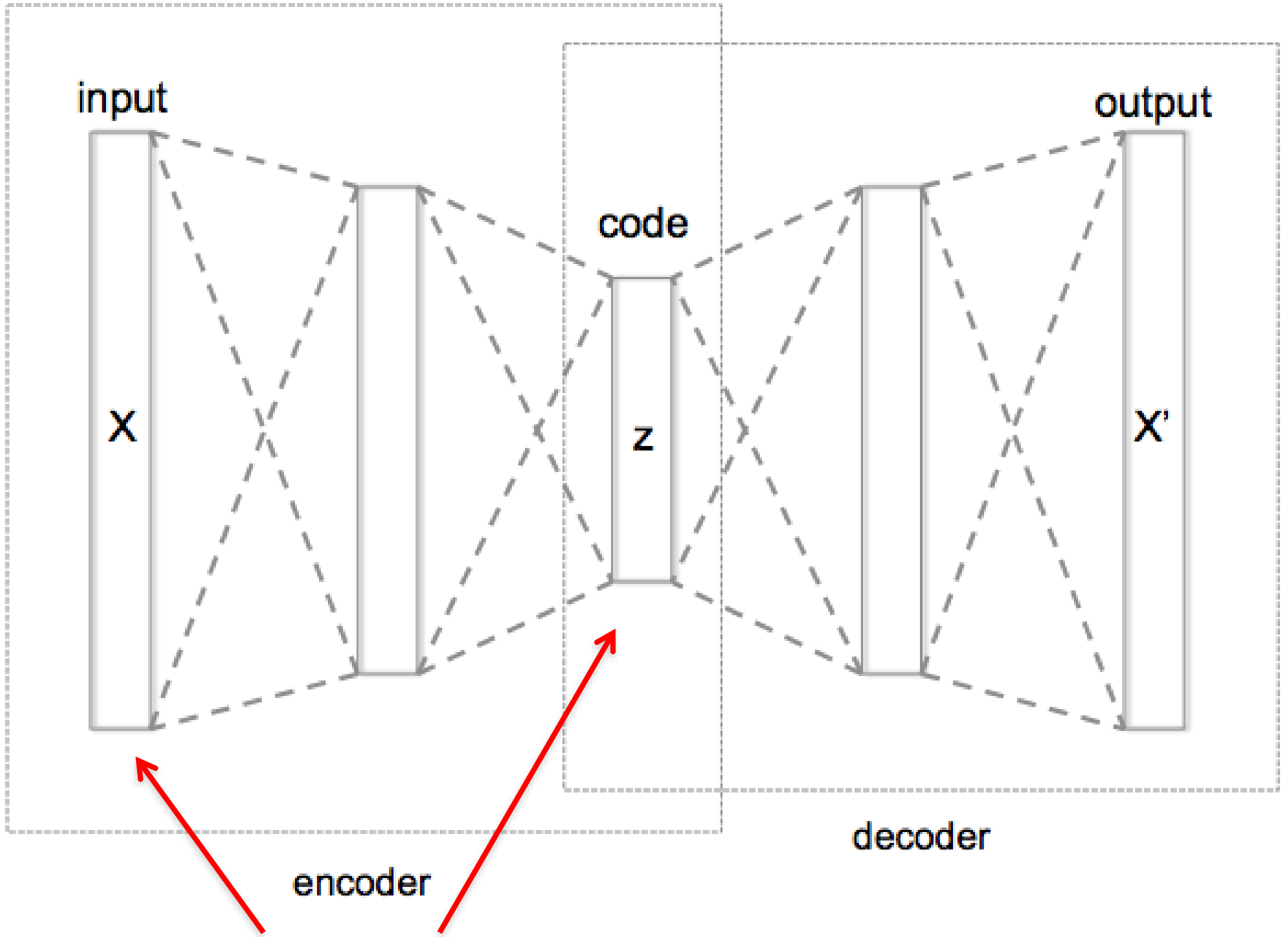


Randomly generate a vector as code



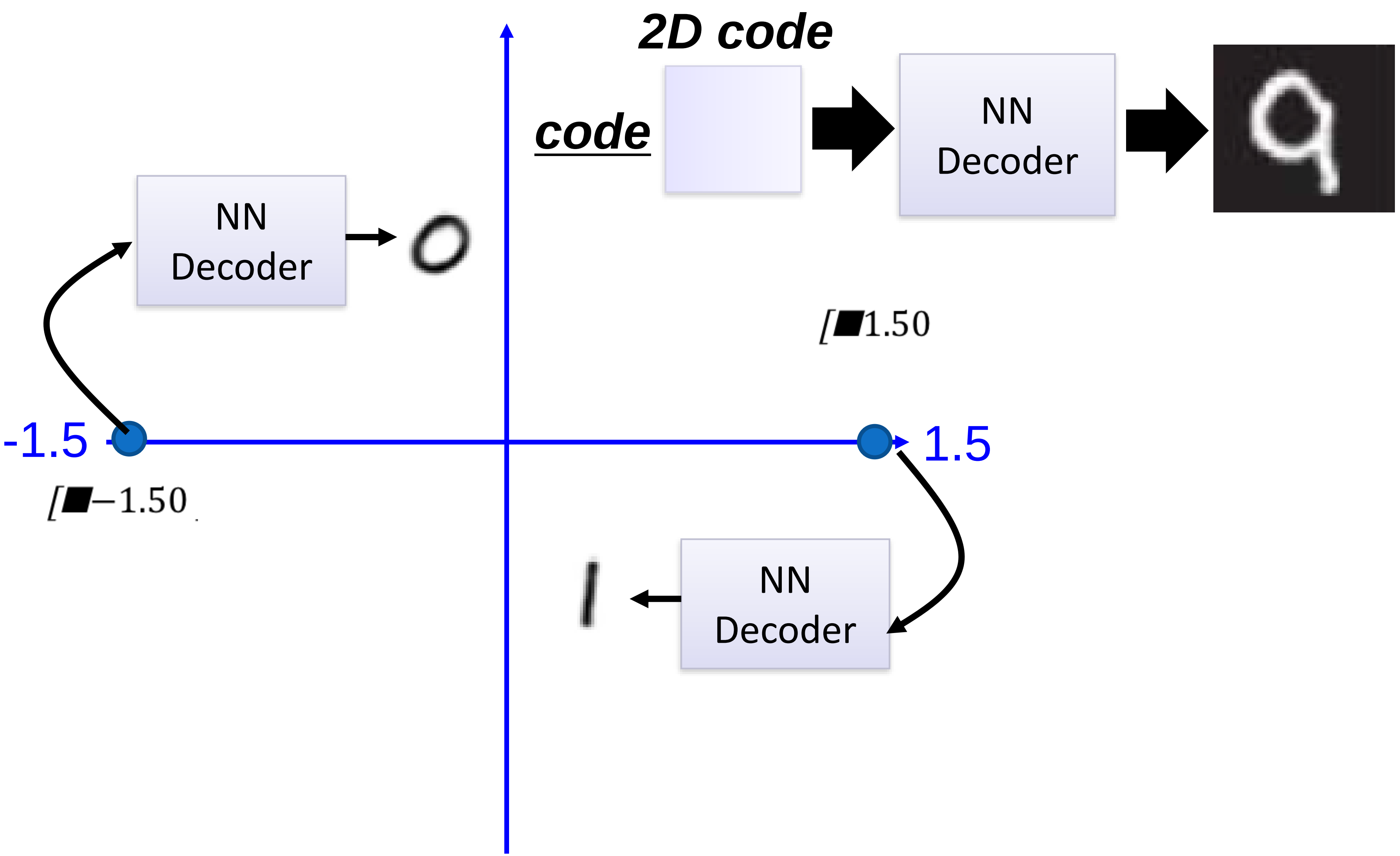
Autoencoder with 3 fully connected layers

Training: model.fit(X,X)
Cost function: $\Sigma_{k=1..N} (x_k - x'_k)^2$

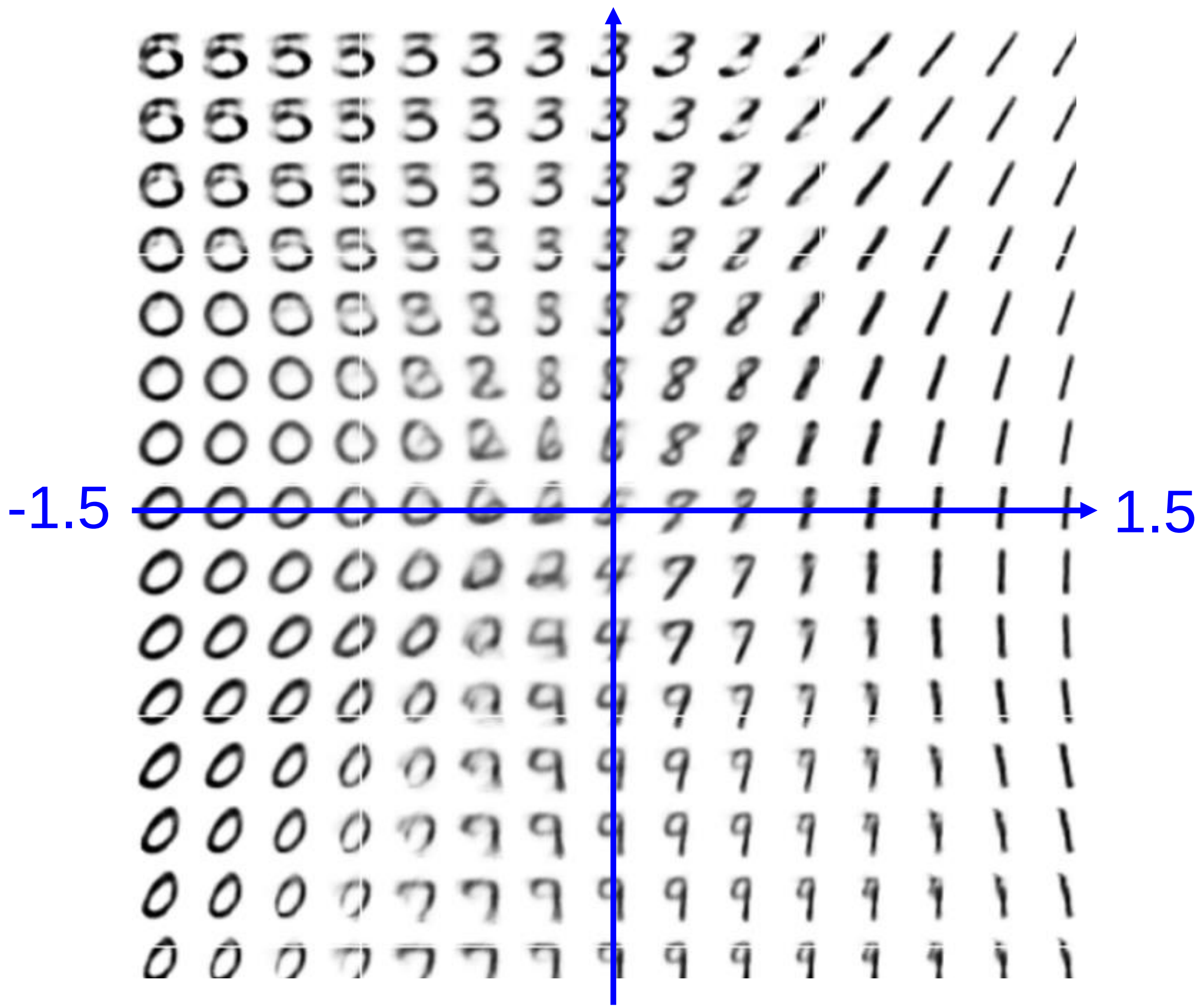


Large → small, learn to compress

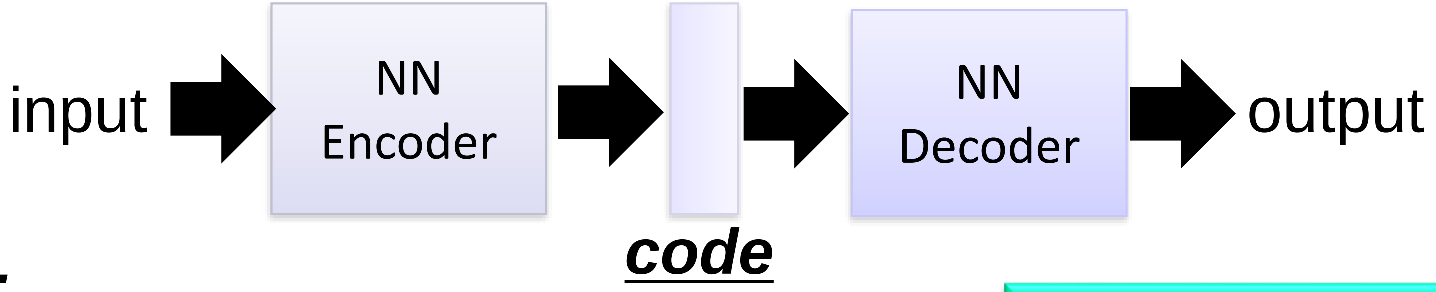
Auto-encoder



Auto-encoder

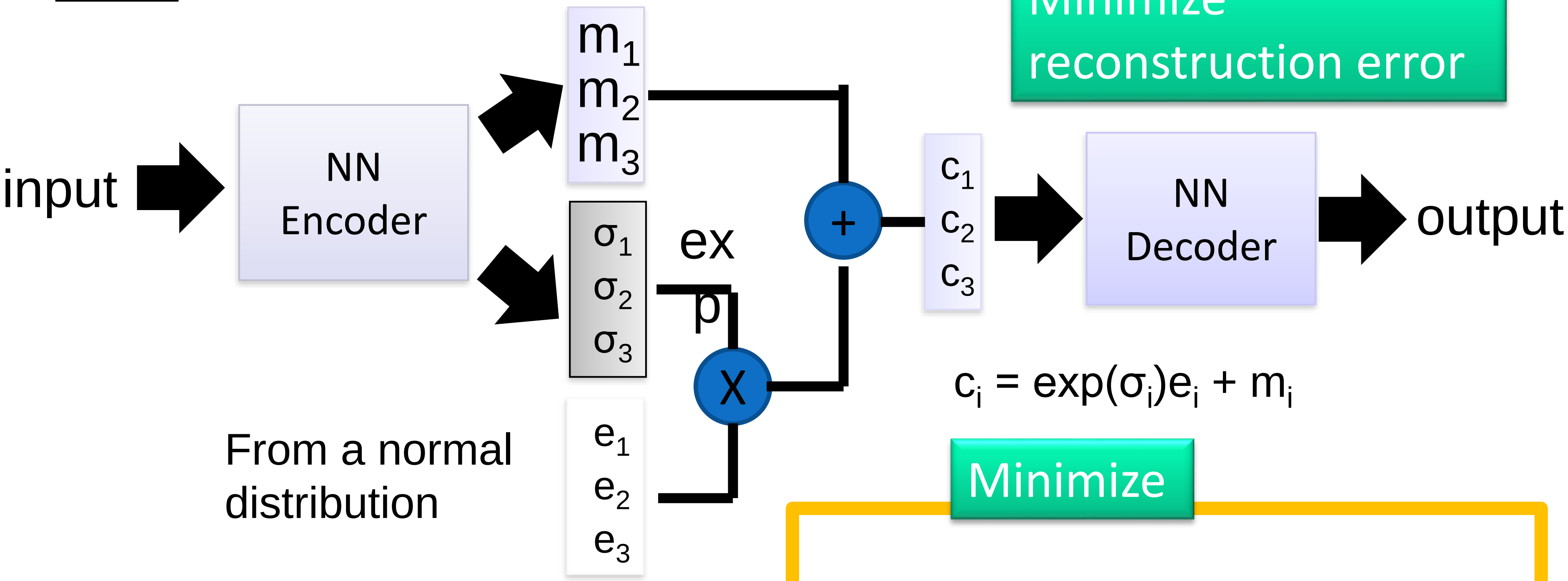


Auto-encoder



VAE

Minimize reconstruction error



Minimize

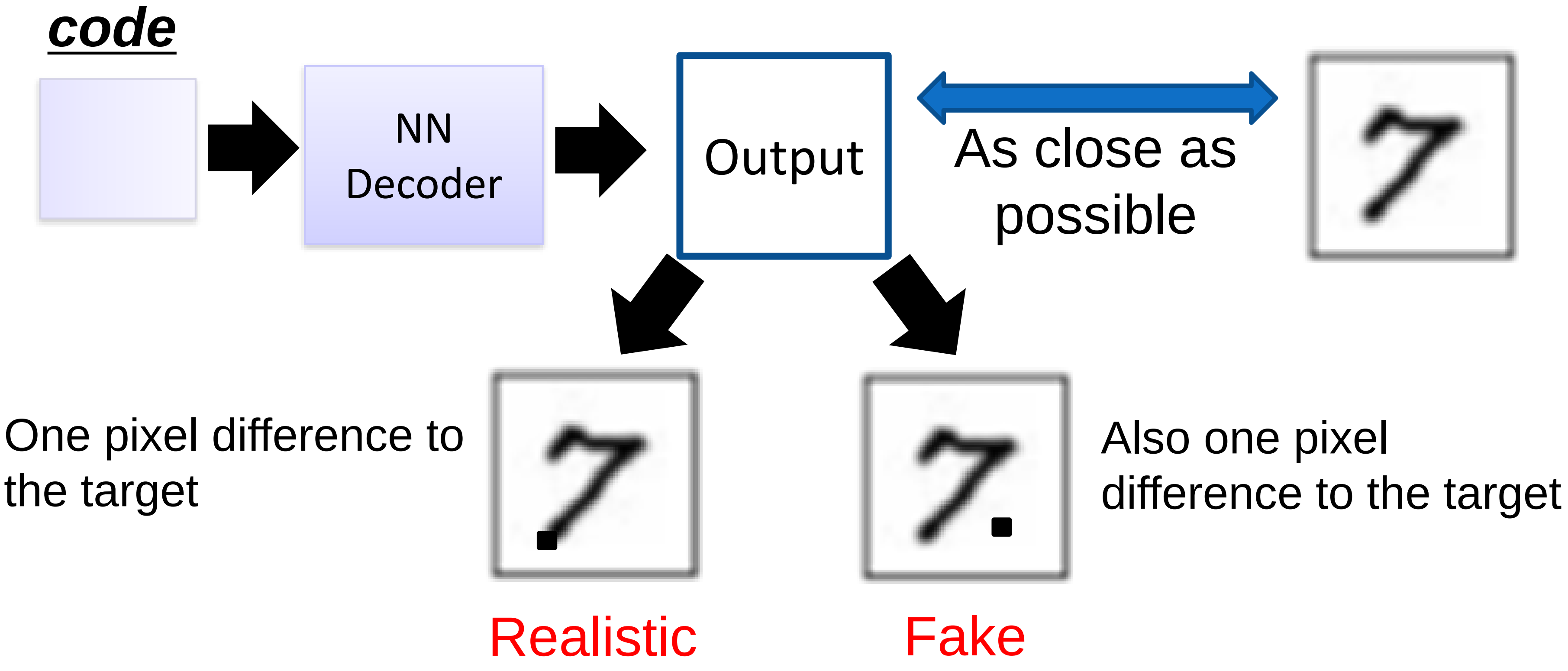
$$\Sigma_{i=1..3}\ [\exp(\sigma_i)-(1+\sigma_i)+(m_i)^2\ ]$$

This constrains σ_i approacing 0 is good

Auto-Encoding Variational Bayes, https://arxiv.org/abs/1312.6114

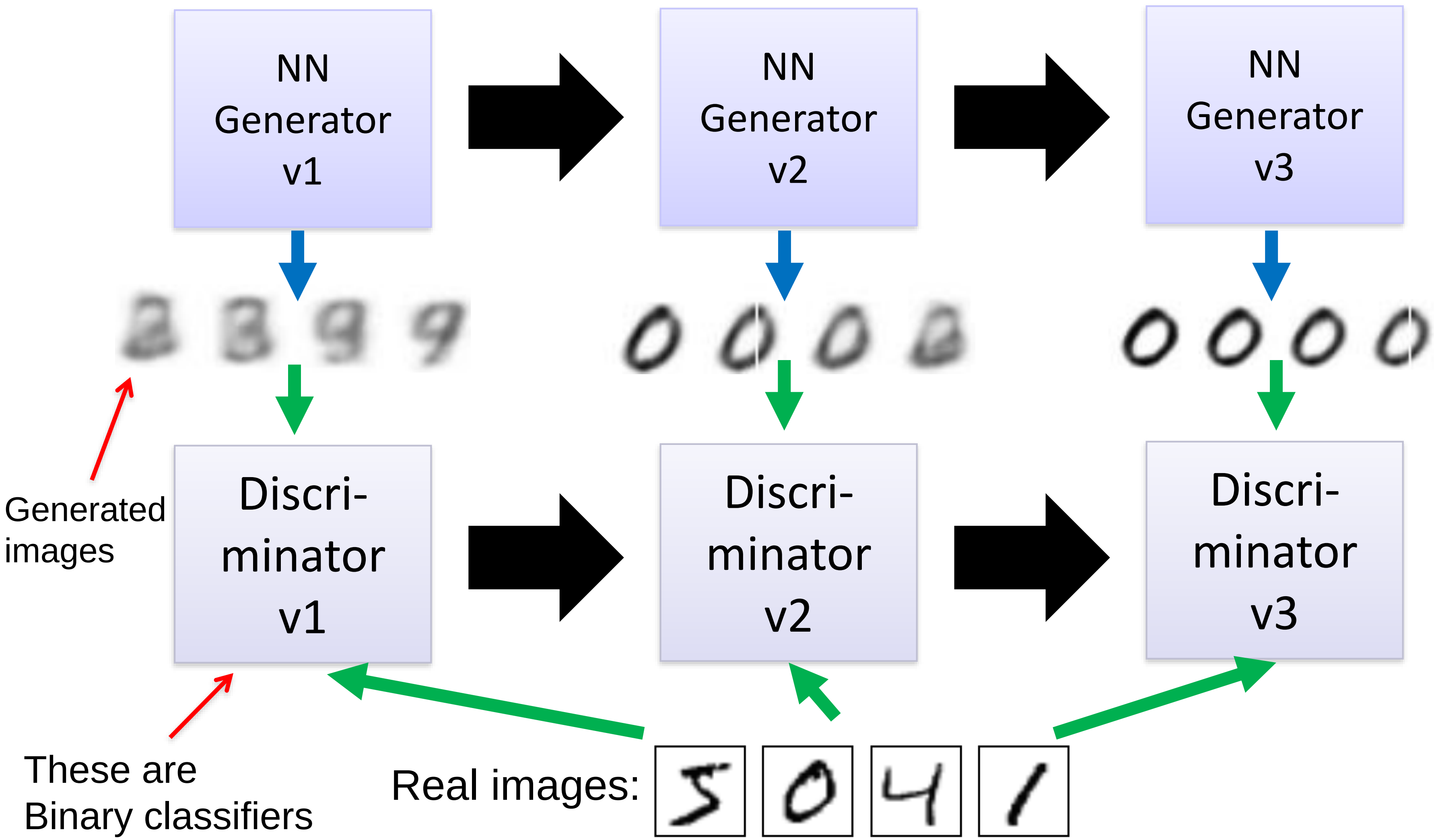
Problems of VAE

It does not really try to simulate real images

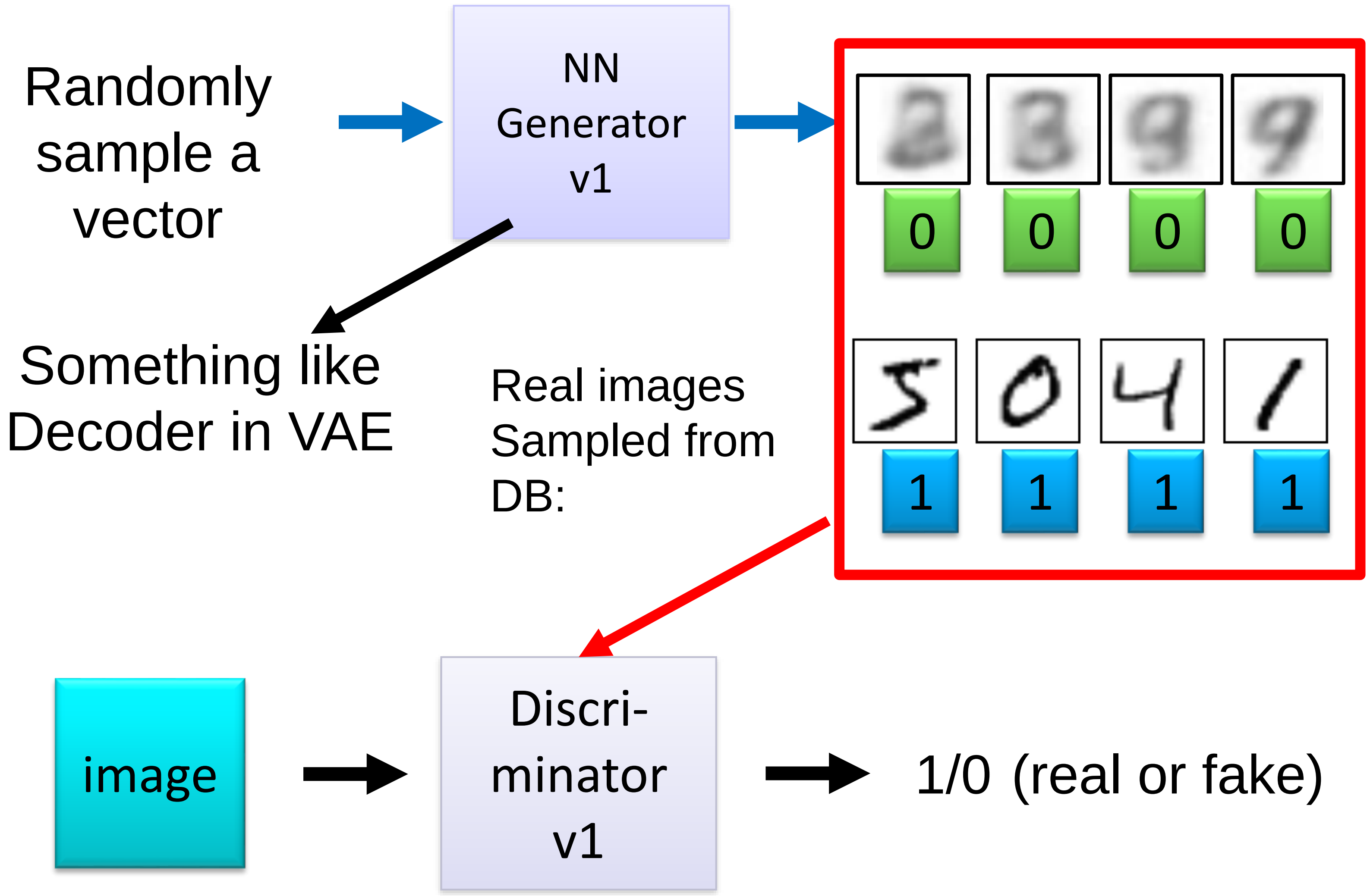


VAE treats these the same

Gradual and step-wise generation



GAN – Learn a discriminator



GAN – Learn a generator

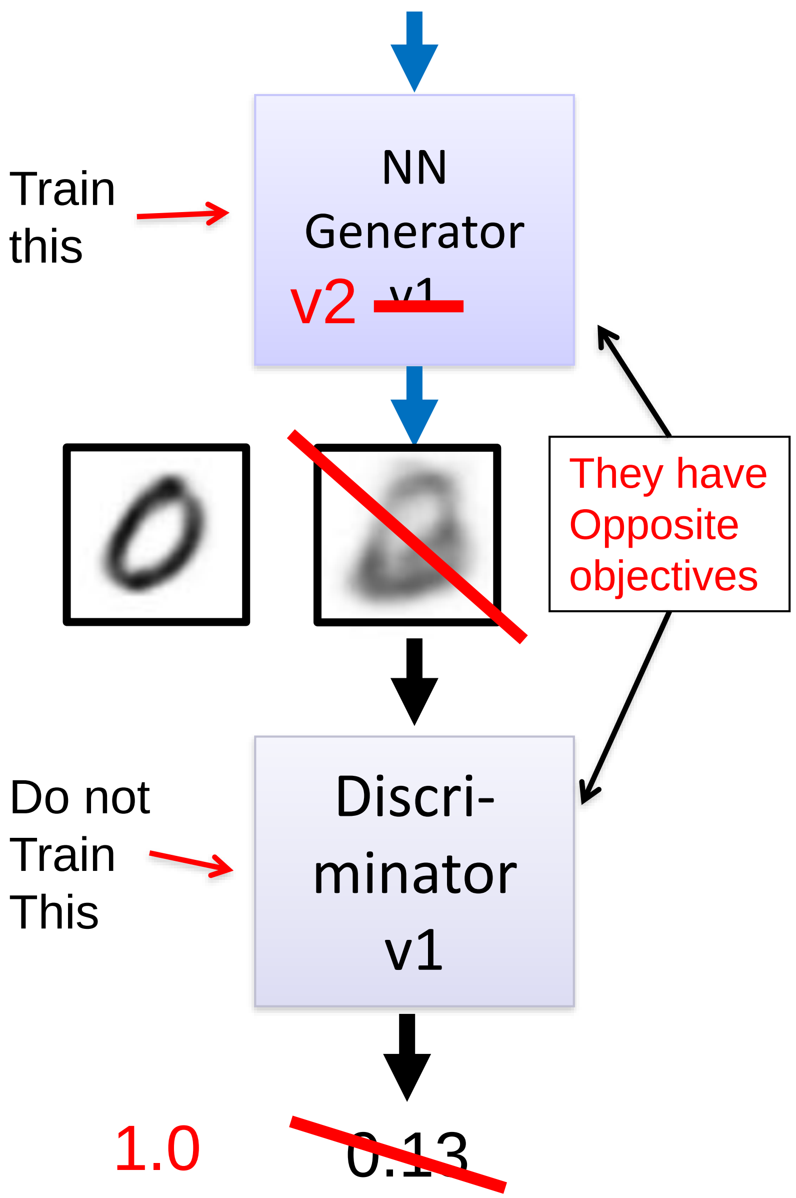
Updating the parameters of generator

➡ The output be classified as “real” (as close to 1 as possible)

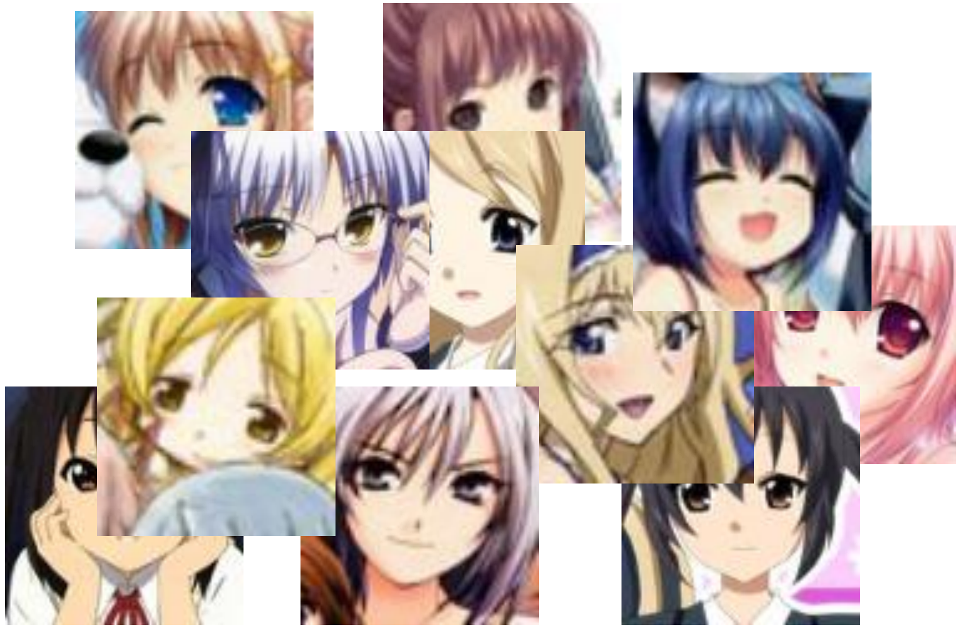
Generator + Discriminator = a network

Using gradient descent to update the parameters in the generator, but fix the discriminator

Randomly sample a vector



Generating 2nd element figures

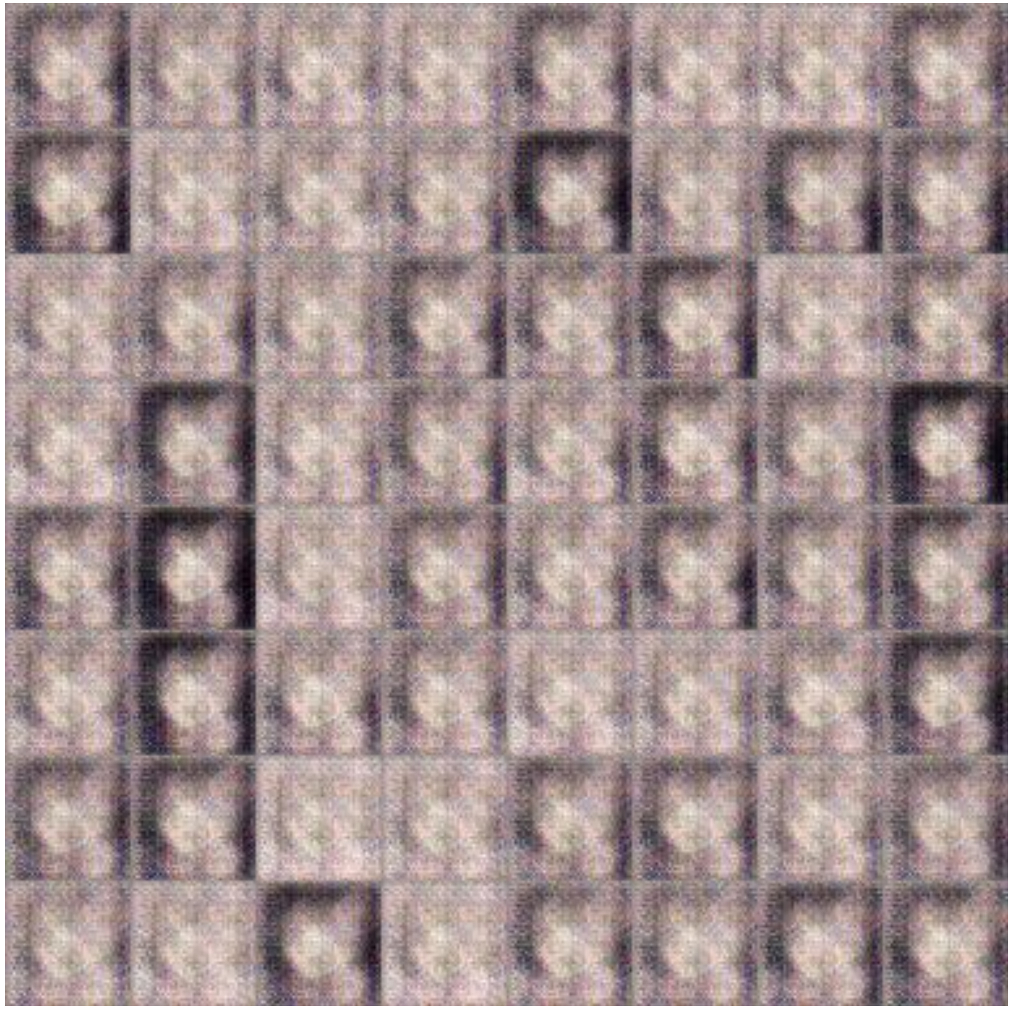


You can use the following to start a project (but this is in Chinese):

Source of images: https://zhuanlan.zhihu.com/p/24767059
From Dr. HY Lee's notes.

DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

GAN – generating 2nd element figures



100 rounds

This is fast, I think you can use your CPU

GAN – generating 2nd element figures

1000 rounds

GAN – generating 2nd element figures

2000 rounds

GAN – generating 2nd element figures

5000 rounds

GAN – generating 2nd element figures

10,000 rounds

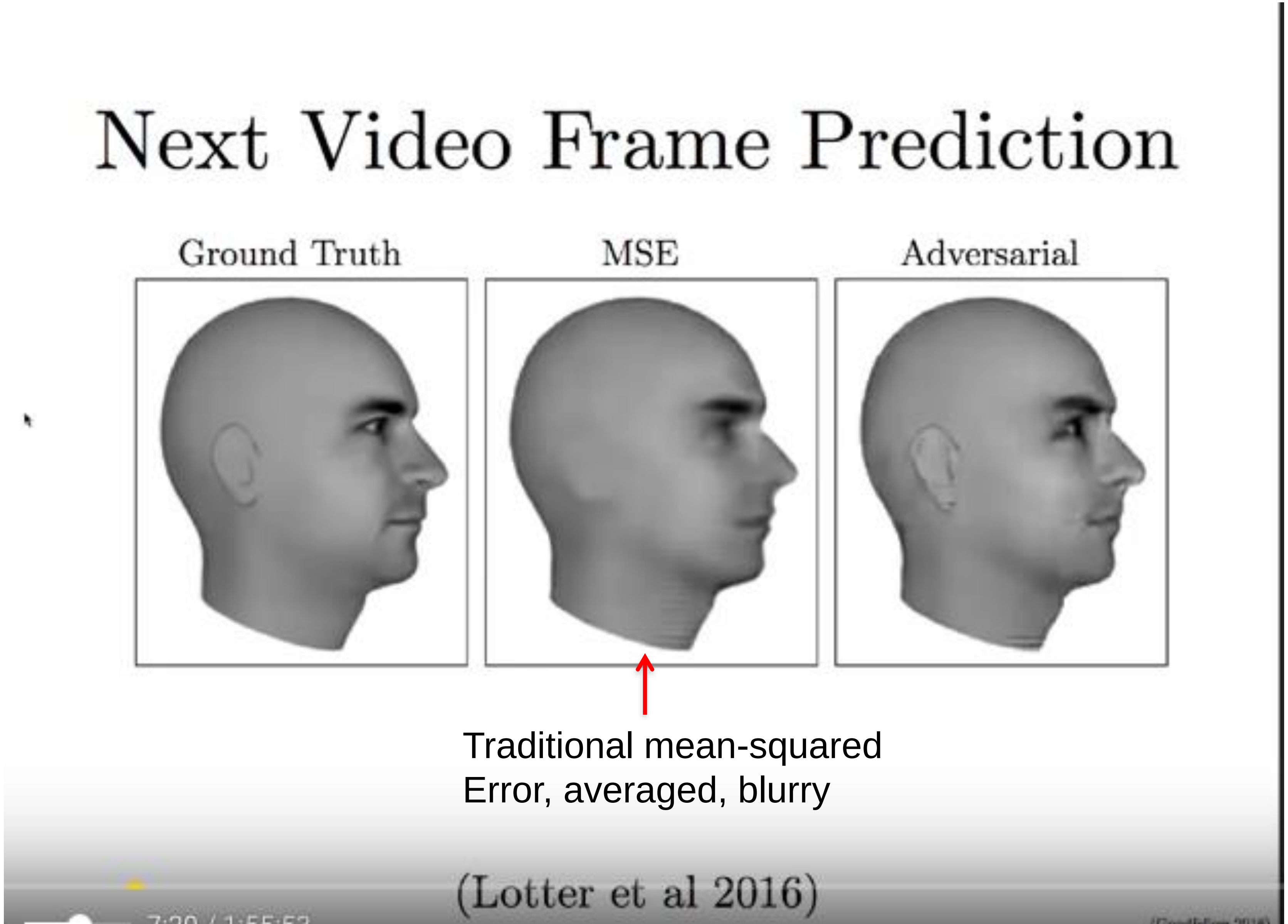
GAN – generating 2nd element figures

20,000 rounds

GAN – generating 2nd element figures

50,000 rounds

Next few images from Goodfellow lecture



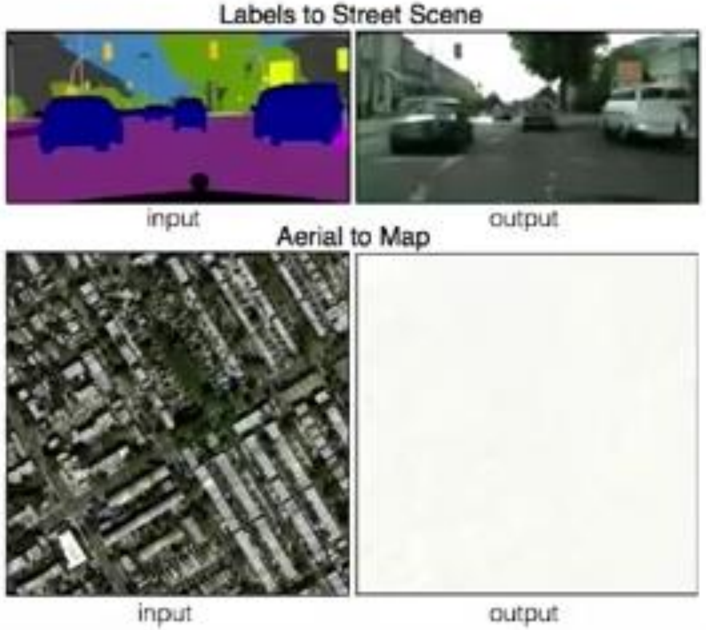
Traditional mean-squared
Error, averaged, blurry

Single Image Super-Resolution

(Ledig et al 2016)

Last 2 are by deep learning approaches.

Image to Image Translation

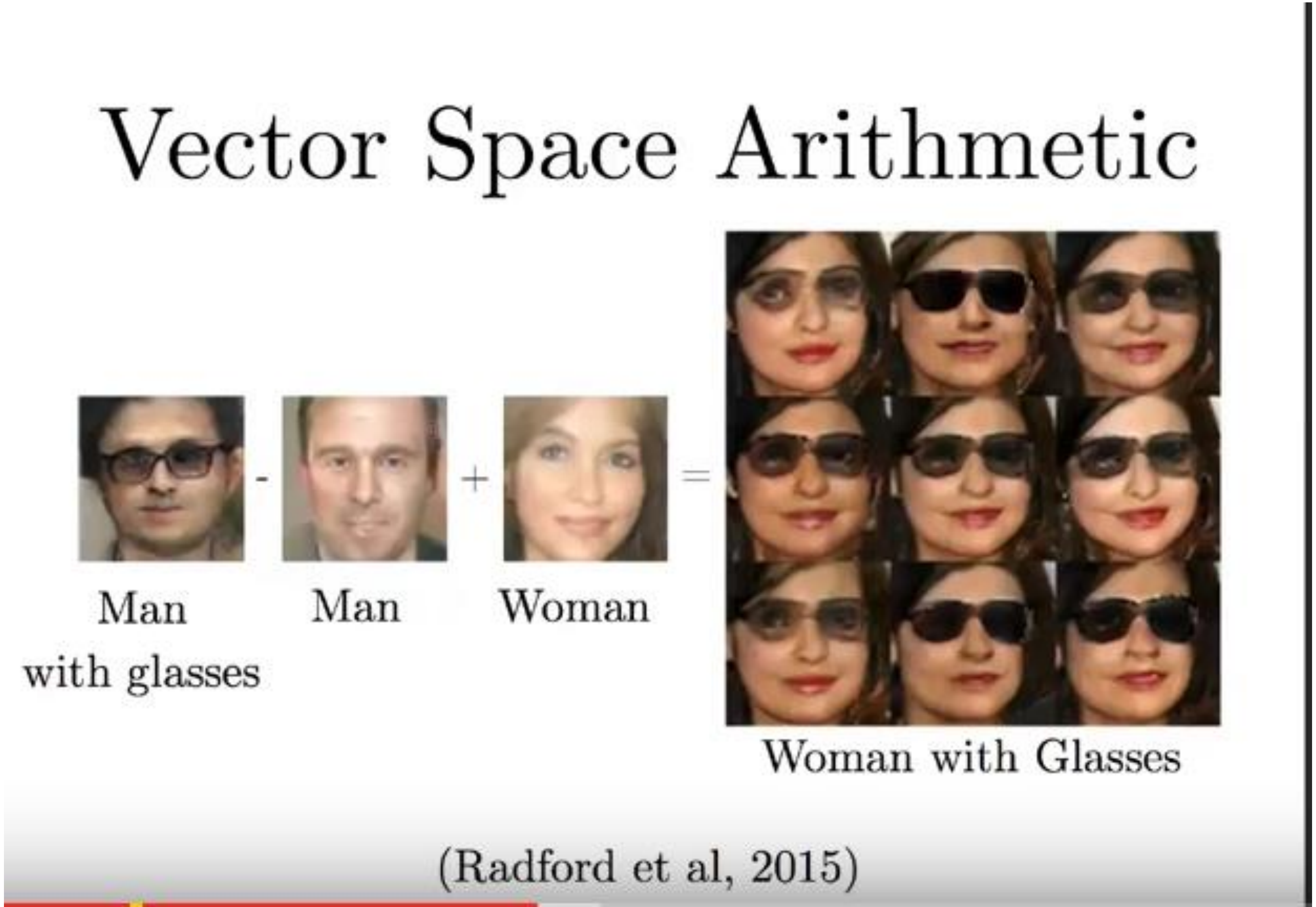


(Isola et al 2016)

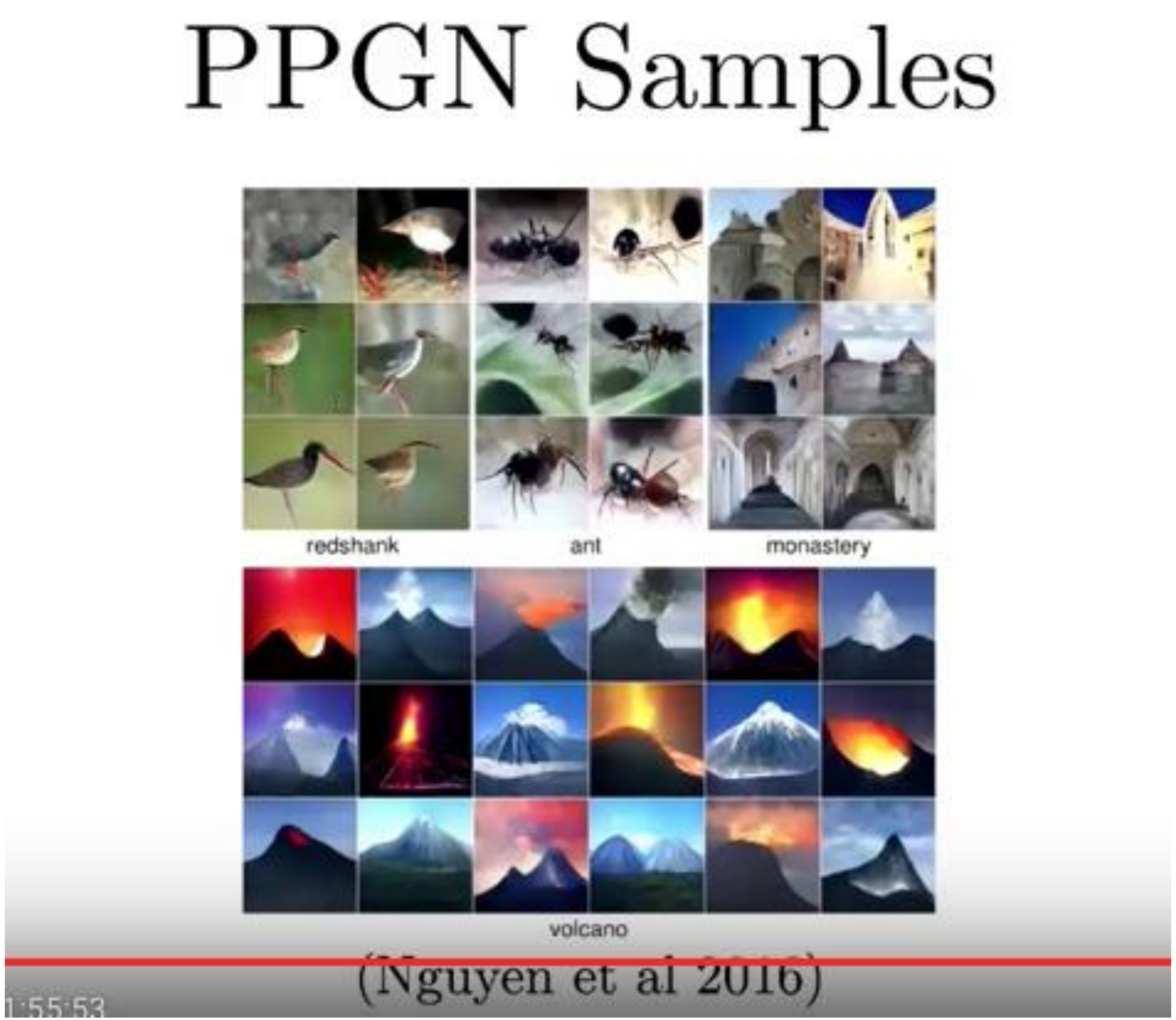
DCGANs for LSUN Bedrooms

(Radford et al 2015)

Similar to word embedding (DCGAN paper)



256x256 high resolution pictures by Plug and Play generative network



From natural language to pictures

PPGN for caption to image

oranges on a table next to a liquor bottle

(Nguyen et al 2016)

Deriving GAN

During the rest of this lecture, we will go thru the original ideas and derive GAN.

I will avoid the continuous case and stick to simple explanations.

Maximum Likelihood Estimation

Give a data distribution $P_{data}(x)$

We use a distribution $P_G(x;\theta)$ parameterized by θ to approximate it

- E.g. $P_G(x;\theta)$ is a Gaussian Mixture Model, where θ contains means and variances of the Gaussians.
- We wish to find θ s.t. $P_G(x;\theta)$ is close to $P_{data}(x)$

In order to do this, we can sample

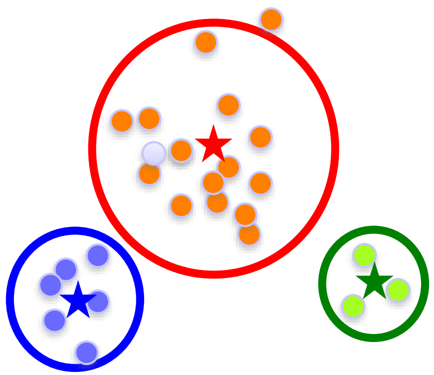
$\{x^1, x^2, \ldots x^m\}$ from $P_{data}(x)$

The likelihood of generating these

x^i's under P_G is

$$L = \Pi_{i=1\ldots m} P_G(x^i; \theta)$$

Then we can find θ^* maximizing the L.



KL (Kullback-Leibler) divergence

Discrete:

$D_{KL}(P||Q) = \Sigma_i P(i)\log[P(i)/Q(i)]$

Continuous:

$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x)\log[p(x)/q(x)]$

Explanations:

Entropy: $- \Sigma_i P(i)\log P(i)$ - expected code length (also optimal)

Cross Entropy: $- \Sigma_i P(i)\log Q(i)$ – expected coding length using optimal code for Q

$D_{KL} = \Sigma_i P(i)\log[P(i)/Q(i)] = \Sigma_i P(i)[\log P(i) – \log Q(i)]$, extra bits

JSD$(P||Q) = ½ D_{KL}(P||M)+ ½ D_{KL}(Q||M)$, $M= ½ (P+Q)$, symmetric KL

* JSD = Jensen-Shannon Divergency

Maximum Likelihood Estimation

$\theta^* = \arg\max_\theta \Pi_{i=1..m} P_G(x^i; \theta) \rightarrow$

$\arg\max_\theta \log \Pi_{i=1..m} P_G(x^i; \theta)$

$= \arg\max_\theta \Sigma_{i=1..m} \log P_G(x^i; \theta)$, $\{x^1, \ldots, x^m\}$ sampled from $P_{data}(x)$

$= \arg\max_\theta \Sigma_{i=1..m} P_{data}(x^i) \log P_G(x^i; \theta)$ --- this is cross entropy

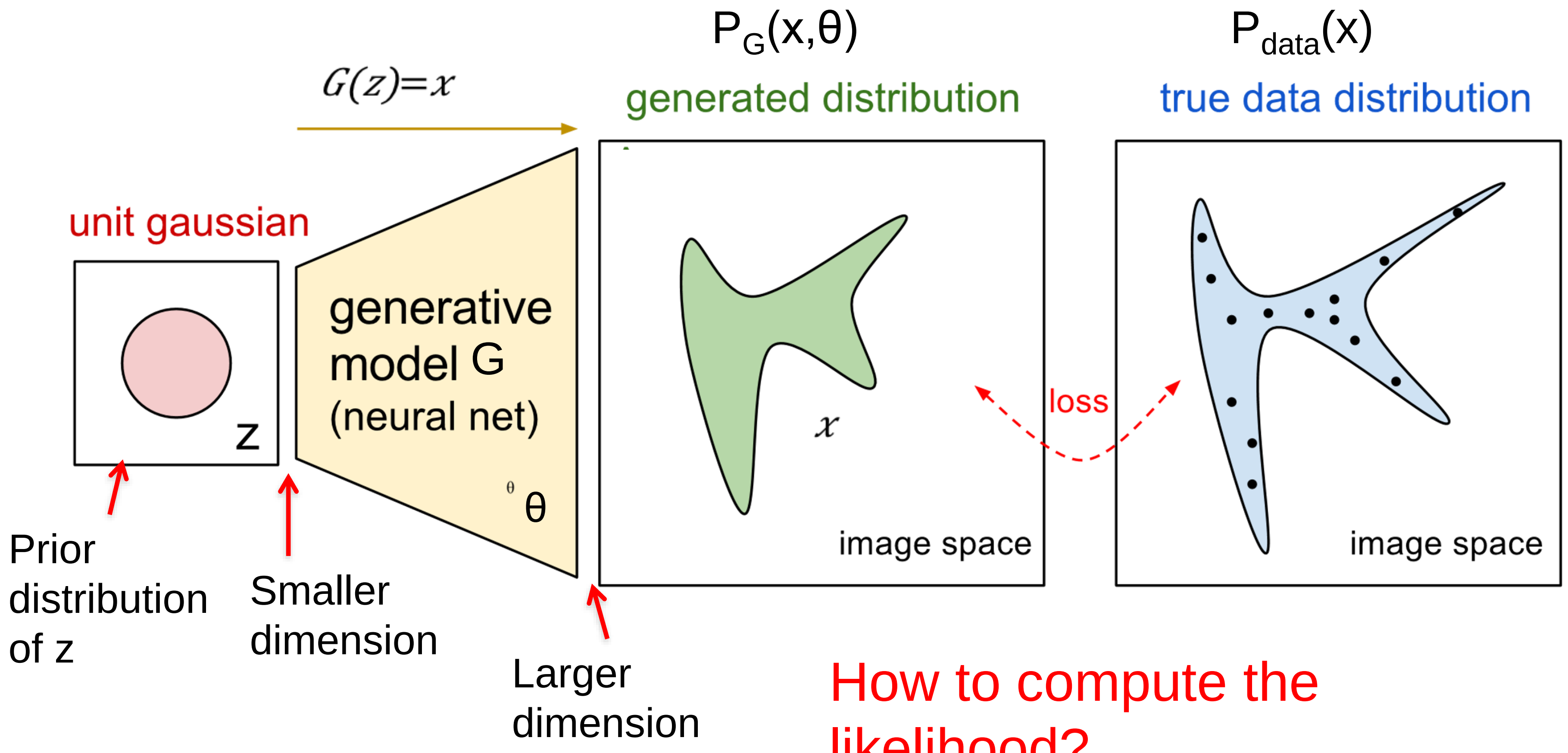
$\cong \arg\max_\theta \Sigma_{i=1..m} P_{data}(x^i) \log P_G(x^i; \theta) - \Sigma_{i=1..m} P_{data}(x^i) \log P_{data}(x^i)$

$= \arg\min_\theta KL(P_{data}(x) || P_G(x; \theta))$ --- this is KL divergence

Note: P_G is Gaussian mixture model, finding best θ will still be Gaussians, this only can generate a few blubs. Thus this above maximum likelihood approach does not work well.

Next we will introduce GAN that will change P_G, not just estimating $P_{G\ is}$ parameters We will find best P_G, which is more complicated and structured, to approximate P_{data}.

Thus let's use an NN as $P_G(x; \theta)$



How to compute the likelihood?

$P_G(x)$ = Integration$_z$ $P_{prior}(z)$ $I_{[G(z)=x]}dz$

https://blog.openai.com/generative-models/

Basic Idea of GAN

Generator G

Hard to learn P_G by maximum likelihood

- G is a function, input z, output x
- Given a prior distribution $P_{prior}(z)$, a probability distribution $P_G(x)$ is defined by function G

Discriminator D

- D is a function, input x, output scalar
- Evaluate the "difference" between $P_G(x)$ and $P_{data}(x)$

In order for D to find difference between P_{data} from P_G, we need a cost function V(G,D):

$$G^* = \arg\min_G \max_D V(G,D)$$

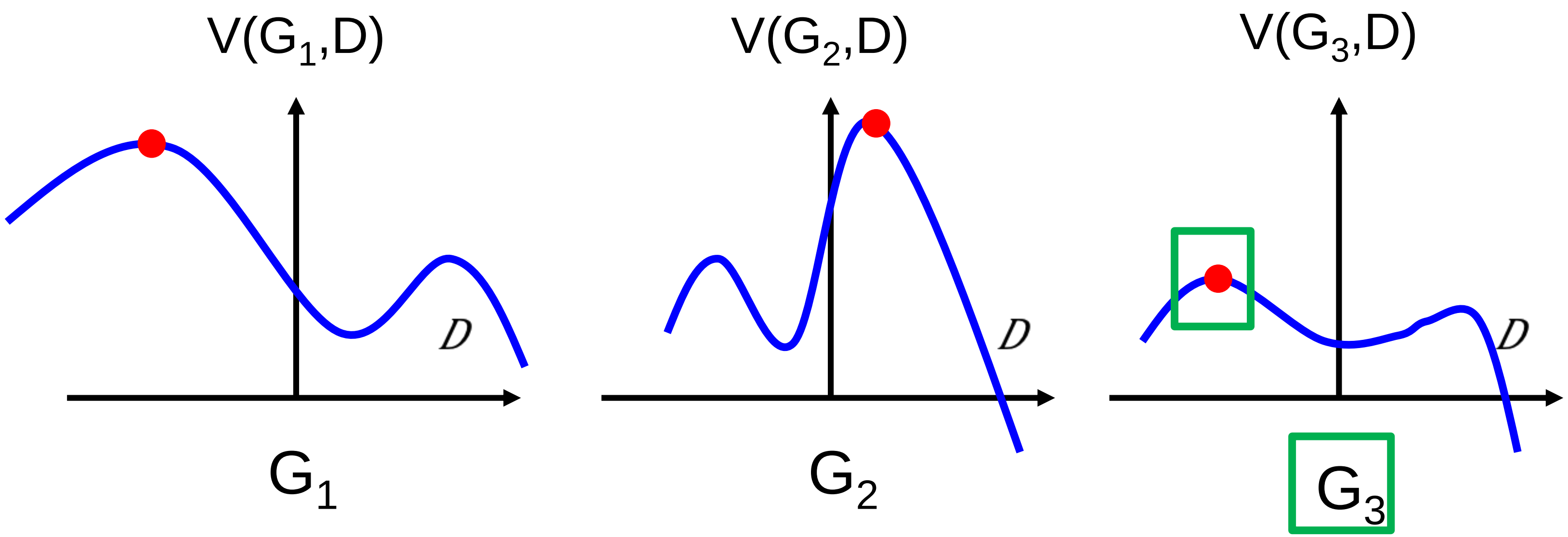
Basic Idea

$$\boxed{G^*} = \arg\min_G \underline{\max_D V(G,D)}$$

Pick JSD function: $V = E_{x\sim P\_data}[\log D(x)] + E_{x\sim P\_G}[\log(1-D(x))]$

Given a generator G, $\max_D V(G,D)$ evaluates the "difference" between P_G and P_{data}

Pick the G s.t. P_G is most similar to P_{data}



$\mathrm{Max}_D V(G,D)$, $G^*=\arg\min_G \max_D V(G,D)$

Given G, what is the optimal D* maximizing

$$V = E_{x\sim P\_data}[\log D(x)] + E_{x\sim P\_G}[\log(1-D(x))]$$
$$= \Sigma\, [\, P_{data}(x) \log D(x) + P_G(x) \log(1-D(x)) \,]$$

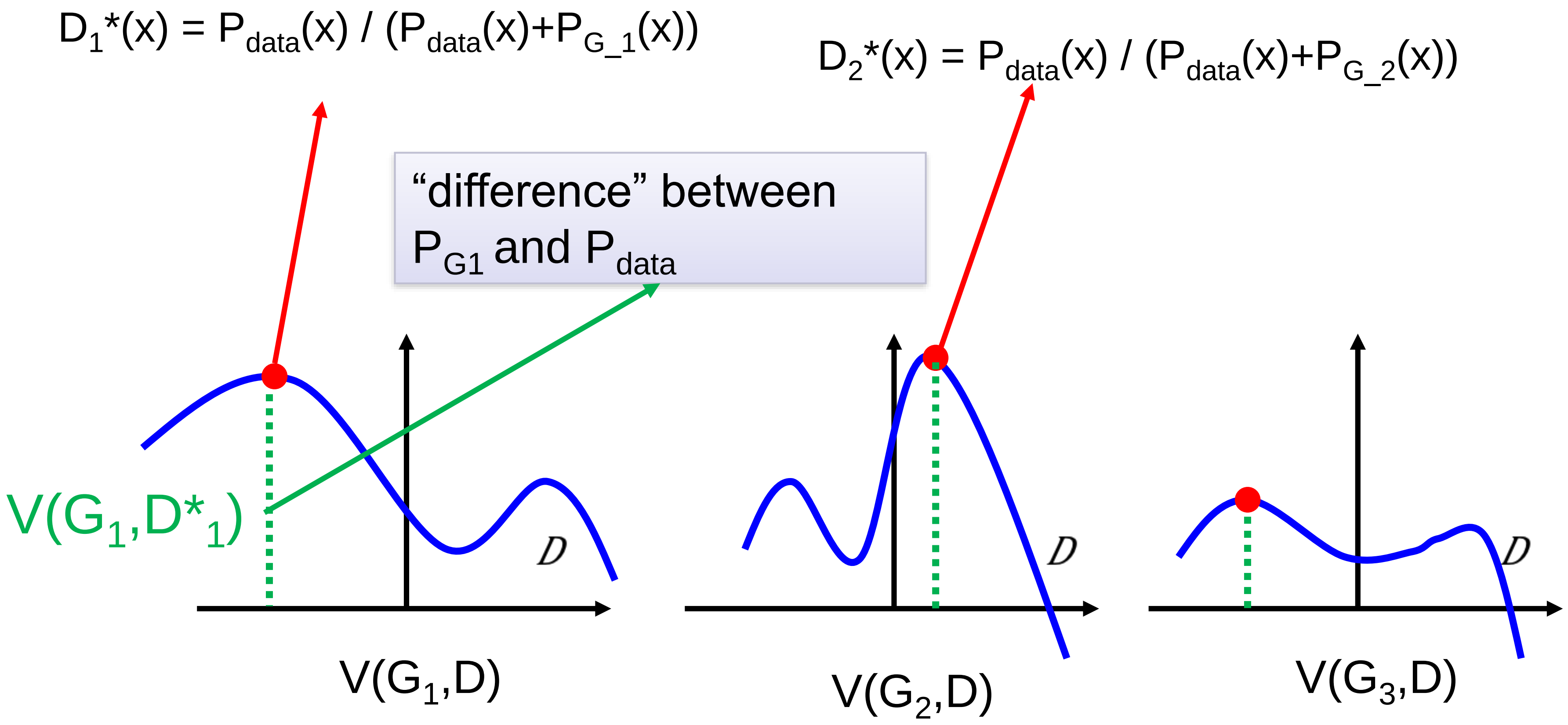
Thus: $D^*(x) = P_{data}(x) / (P_{data}(x)+P_G(x))$

Assuming D(x) can have any value here

Given x, the optimal D* maximizing is:

$f(D) = a\log D + b\log(1-D) \Rightarrow D^*=a/(a+b)$

$\max_D V(G,D)$, $G^* = \arg\min_G \max_D V(G,D)$



$\max_D V(G,D)$

$$\boxed{\begin{aligned} V = &E_{x\sim P\_data}\,[\log D(x)] \\ &+ E_{x\sim P\_G}[\log(1-D(x))] \end{aligned}}$$

$$\begin{aligned}
&\max_D V(G,D) \\
&\quad = V(G,D^*),\ \text{where } D^*(x) = P_{data} / (P_{data} + P_G), \text{ and} \\
&\qquad\qquad\qquad\qquad 1-D^*(x) = P_G / (P_{data} + P_G) \\
&\quad = E_{x\sim P_data} \log D^*(x) + E_{x\sim P_G} \log(1-D^*(x)) \\
&\quad \approx \Sigma\, [\, P_{data}(x) \log D^*(x) + P_G(x) \log(1-D^*(x))\,] \\
&\quad = -2\log 2 + 2\, JSD(P_{data} \,||\, P_G),
\end{aligned}$$

$JSD(P||Q)$ = Jensen-Shannon divergence

$= \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M)$

where $M = \frac{1}{2}(P+Q)$.

$D_{KL}(P||Q) = \Sigma\, P(x) \log P(x)/Q(x)$

Summary:

$$V = E_{x\sim P\_data}[\log D(x)] + E_{x\sim P\_G}[\log(1-D(x))]$$

Generator G, Discriminator D

Looking for G* such that

$$G^* = \arg\min_G \max_D V(G,D)$$

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

$$= -2\log 2 + 2JSD(P_{data}(x)\ ||\ P_G(x))$$

What is the optimal G? It is G that makes JSD smallest = 0:

$$P_G(x) = P_{data}(x)$$

Algorithm

$$G^* = \arg\min_G \boxed{\max_D V(G,D)}$$

L(G), this is the loss function

To find the best G minimizing the loss function L(G):

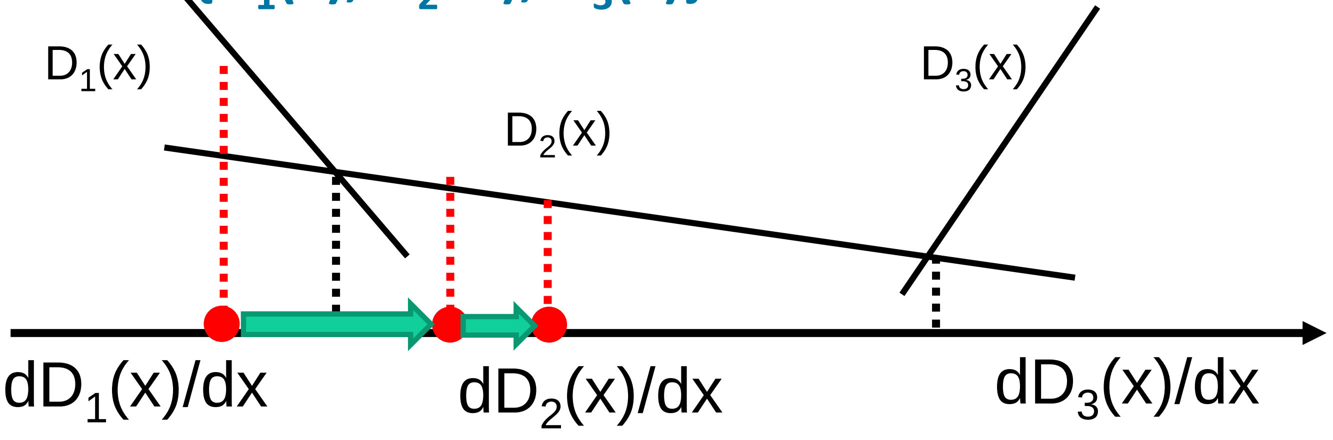
$\theta_G \leftarrow \theta_G = -\eta\ \partial L(G)/\partial \theta_G$, θ_G defines G

Solved by gradient descent. Having max ok. Consider simple case:

$df(x)/dx = ?$

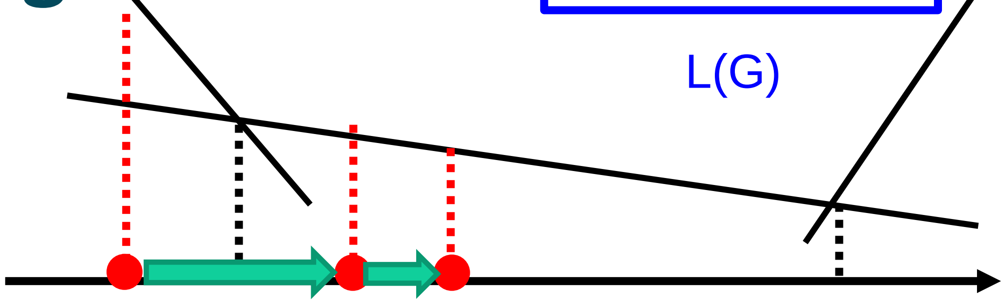
If $D_i(x)$ is the Max in that region, then do $dD_i(x)/dx$

$f(x) = \max\{D_1(x), D_2(x), D_3(x)\}$



Algorithm

$$G^* = \arg\min_G \boxed{\max_D V(G,D)}$$



Given G_0

Find D^*_0 maximizing $V(G_0,D)$

$V(G_0,D_0^*)$ is the JS divergence between $P_{data}(x)$ and $P_{G0}(x)$

$\theta_G \leftarrow \theta_G - \eta\ \Delta V(G,D_0^*) / \theta_G$ ➔ Obtaining G_1 (decrease JSD)

Find D_1^* maximizing $V(G_1,D)$

$V(G_1,D_1^*)$ is the JS divergence between $P_{data}(x)$ and $P_{G1}(x)$

$\theta_G \leftarrow \theta_G - \eta\ \Delta V(G,D_1^*) / \theta_G$ ➔ Obtaining G_2 (decrease JSD)

And so on …

In practice …

$V = E_{x\sim P\_data}[\log D(x)] + E_{x\sim P\_G}[\log(1-D(x))]$

Given G, how to compute $\max_D V(G,D)$?

- Sample $\{x^1, \ldots, x^m\}$ from P_{data}
- Sample $\{x^{*1}, \ldots, x^{*m}\}$ from generator P_G

Maximize:

$$V' = 1/m \sum_{i=1..m} \log D(x^i) + 1/m \sum_{i=1..m} \log(1-D(x^{*i}))$$

Positive example
D must accept

Negative example
D must reject

This is what a Binary Classifier do

Output is D(x) Minimize Cross-entropy

If x is a positive example → Minimize $-\log D(x)$

If x is a negative example → Minimize $-\log(1-D(x))$

Binary Classifier

Output is $f(x)$ Minimize Cross-entropy

If x is a positive example → Minimize $-\log f(x)$

If x is a negative example → Minimize $-\log(1-f(x))$

D is a binary classifier (can be deep) with parameters θ_d

$\{x^1, x^2, \ldots x^m\}$ from $P_{data}(x)$ → Positive examples

$\{x^{*1}, x^{*2}, \ldots x^{*m}\}$ from $P_G(x)$ → Negative examples

Minimize $L = - V'$

or

Maximize $V' = \Sigma_{i=1..m} \log D(x^i) + 1/m \; \Sigma_{i=1..m} \log(1-D(x^{*i}))$

Algorithm

Initialize θ_d for D and θ_g for G

Can only find lower bound of JSD or $\max_D V(G,D)$

In each training iteration

Ian Goodfellow comment: this is also done once

Learning D

- **Sample m examples $\{x^1, x^2, \ldots x^m\}$ from data distribution $P_{data}(x)$**
- **Sample m noise samples $\{z^1, \ldots, z^m\}$ from a simple prior $P_{prior}(z)$**
- **Obtain generated data $\{x^{*1}, \ldots, x^{*m}\}$, $x^{*i}=G(z^i)$**
- **Update discriminator parameters θ_d to maximize**

  $\tilde{V} = 1/m \, \Sigma_{i=1..m} \log D(x^i) + 1/m \, \Sigma_{i=1..m} \log(1-D(x^{*i}))$

  $\theta_d \leftarrow \theta_d + \eta \Delta \tilde{V}(\theta_d)$ (gradient ascent)

Repeat k times

Learning G

Only Once

- **Simple another m noise samples $\{z^1, z^2, \ldots z^m\}$ from the prior $P_{prior}(z)$, $G(z^i)=x^{*i}$**
- **Update generator parameters θ_g to minimize**

  $\tilde{V} = 1/m \Sigma_{i=1..m} \log D(x^i) + 1/m \, \Sigma_{i=1..m} \log(1-D(x^{*i}))$

  $\theta_g \leftarrow \theta_g - \eta \Delta \tilde{V}(\theta_g)$ (gradient descent)

Objective Function for Generator in Real Implementation

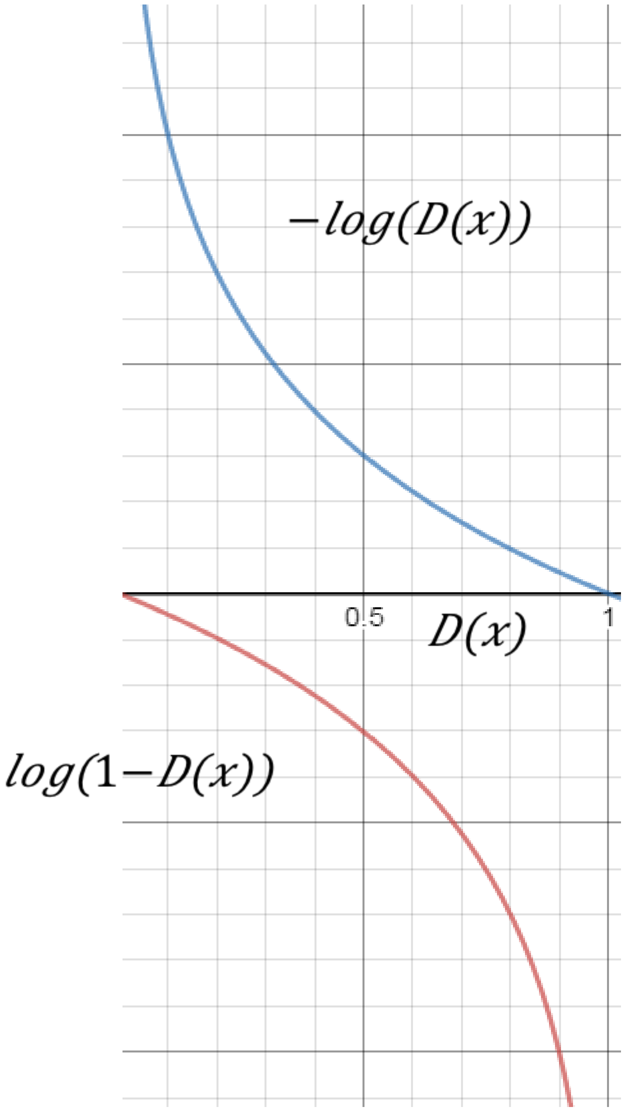
$$V = \cancel{E_{x \sim P\_data}[\log D(x)}$$

$$+ E_{x \sim P\_G}[\log(1-D(x))]$$

Training slow at the beginning

$$V = E_{x \sim P\_G}[-\log(D(x))]$$

Real implementation:
label x from P_G as positive

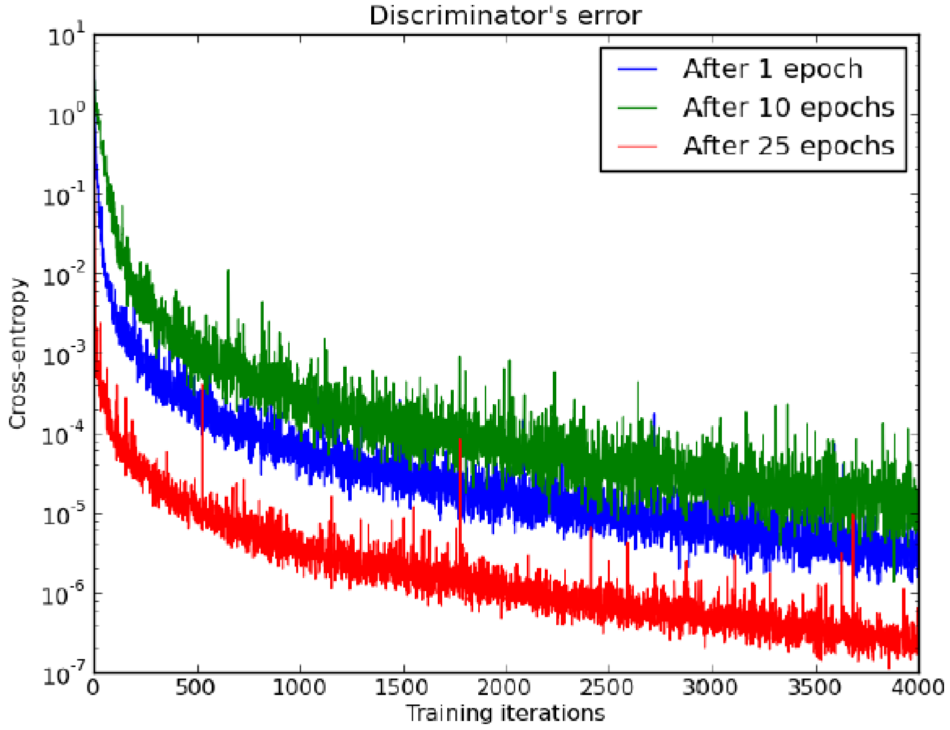


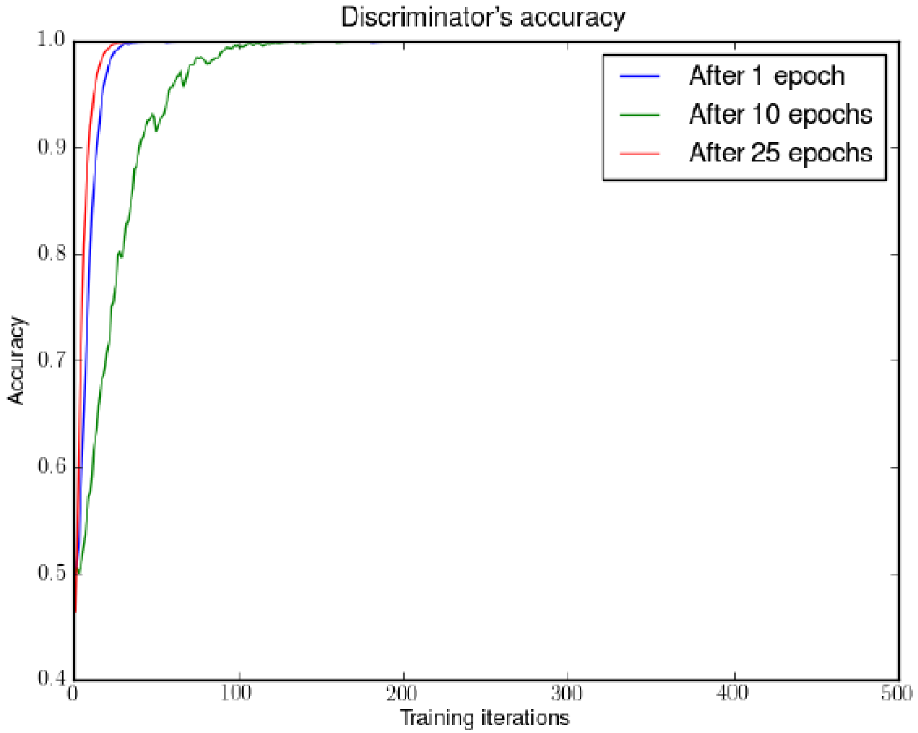
Some issues in training GAN

M. Arjovsky, L. Bottou, Towards principled methods for training generative adversarial networks, 2017.

Evaluating JS divergence

Discriminator is too strong: for all three Generators, JSD = 0



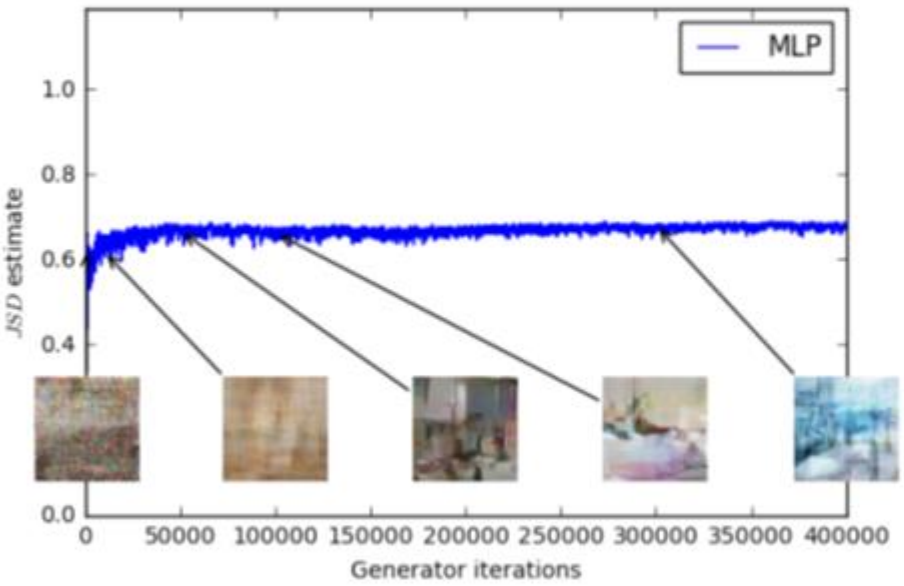


Martin Arjovsky, Léon Bottou, Towards Principled Methods for Training Generative Adversarial Networks, 2017, arXiv preprint

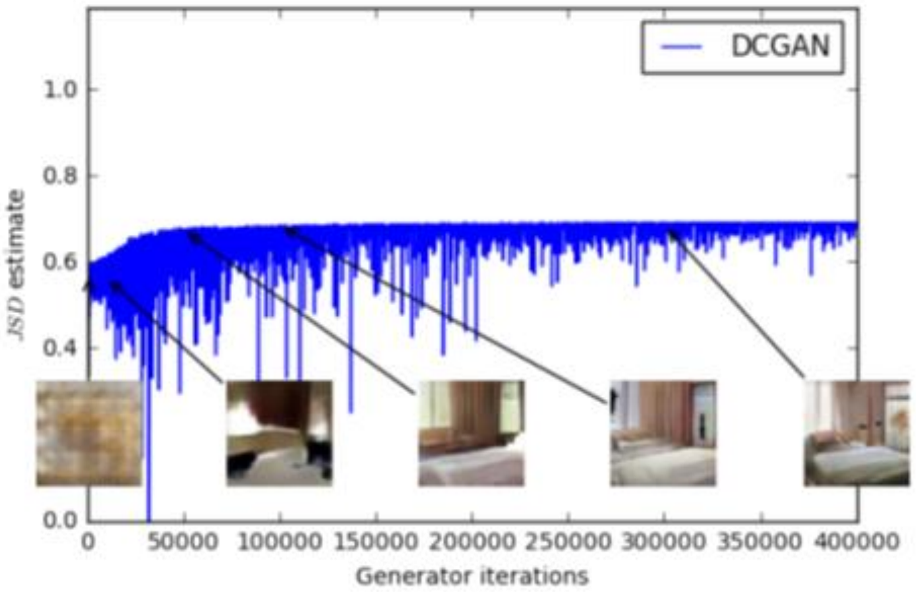
Evaluating JS divergence

https://arxiv.org/abs/1701.07875

JS divergence estimated by discriminator telling little information



Weak Generator



Strong Generator

Discriminator

1 for all positive examples

0 for all negative examples

$$V = E_{x\sim P\_data}[\log D(x)] + E_{x\sim P\_G}[\log(1-D(x))]$$
$$= 1/m\ \Sigma_{i=1..m} \log D(x^i) + 1/m\ \Sigma_{i=1..m} \log(1-D(x^{*i}))$$

$$\max_D V(G,D) = -2\log 2 + 2\ \mathrm{JSD}(P_{data} \,||\, P_G)$$

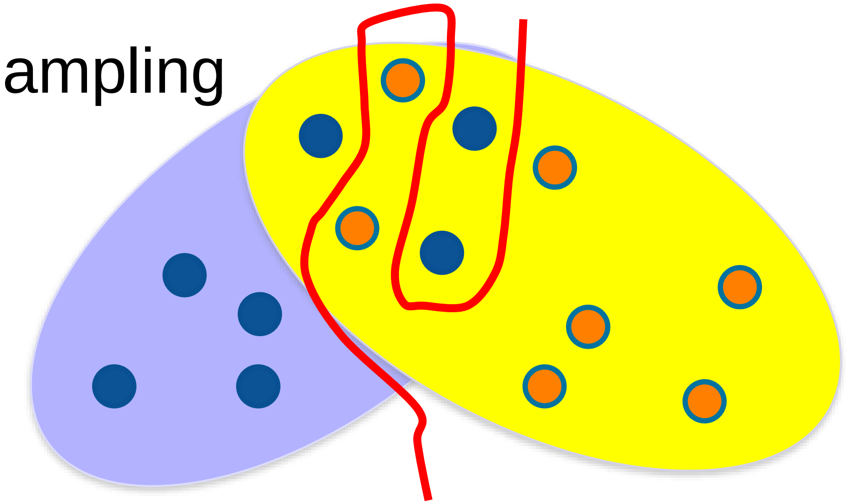
= 0

log 2 when P_{data} and P_G differ completely

Reason 1. Approximate by sampling

Weaken your discriminator?

Can weak discriminator compute JS divergence?



Discriminator

GAN implementation estimation

$$V = E_{x\sim P\_data}\,[\overset{1}{\log D(x)}] + E_{x\sim P\_G}[\log(1-\overset{0}{D(x)})]$$

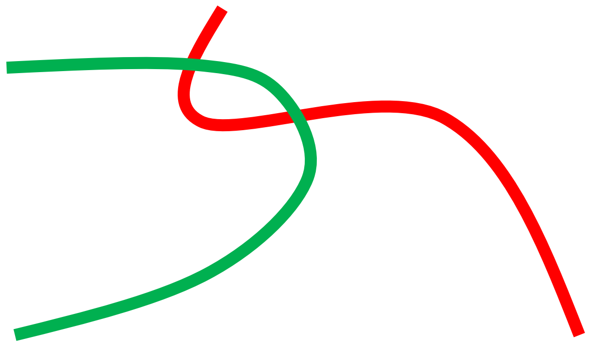
$$= 1/m\ \Sigma_{i=1..m} \log D(x^i) + 1/m\ \Sigma_{i=1..m} \log(1-D(x^{*i}))\ \approx 0$$

$$\max_D V(G,D) = -2\log 2 + \underbrace{2\ JSD(P_{data}\ ||\ P_G\ )}_{\log 2}\ \boxed{= 0}$$

Theoretical estimation

Reason 2. the nature of data

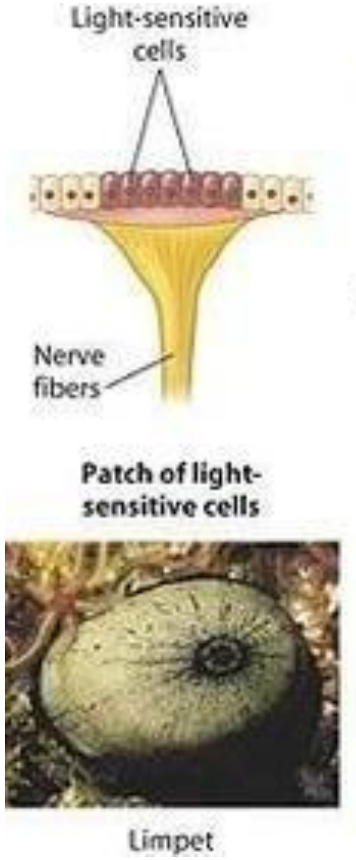
$P_{data}(x)$ and $P_G(x)$ have very little overlap in high dimensional space

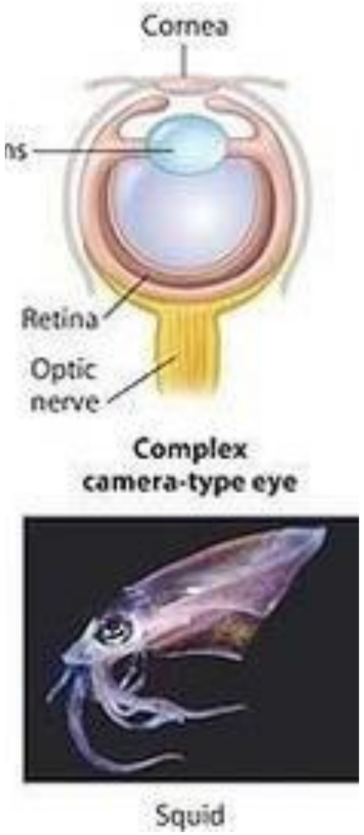


Evolution

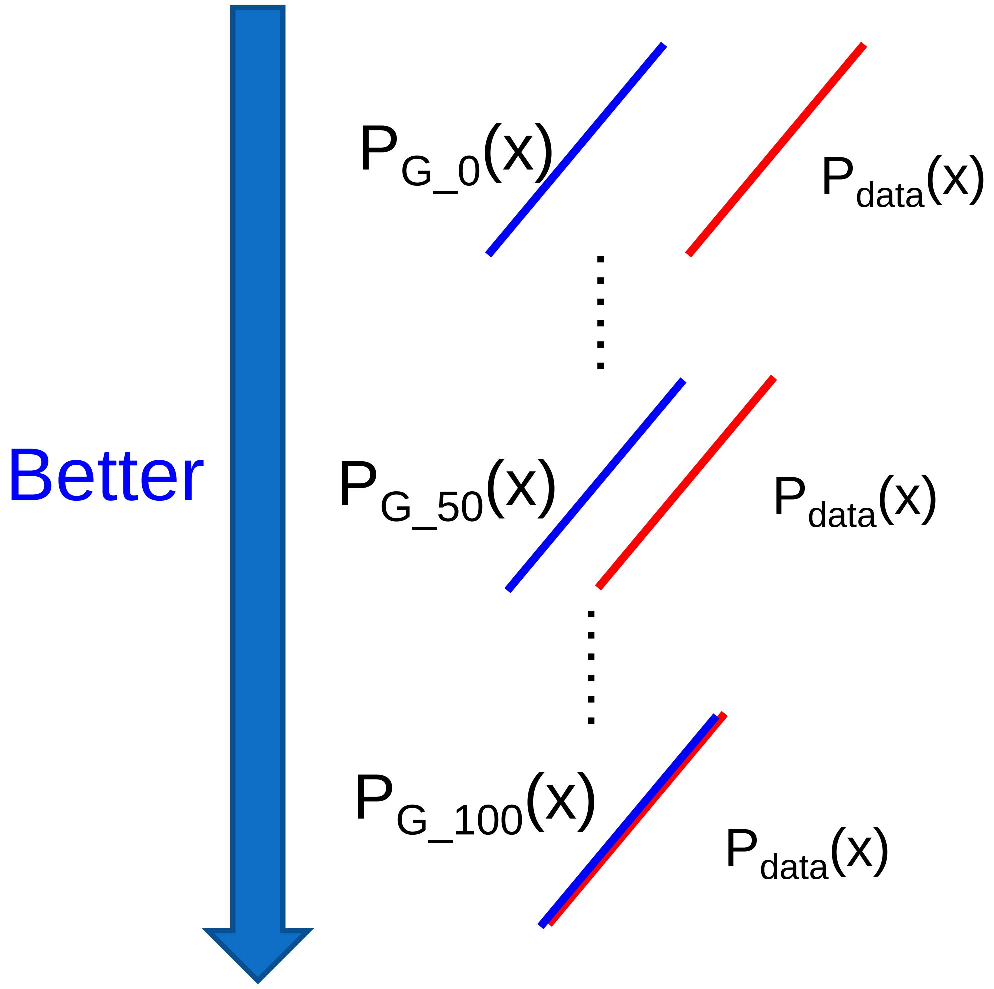
http://www.guokr.com/post/773890/

Better





Evolution needs to be smooth:



$JSD(P_{G\_0} \,||\, P_{data}) = \log 2$

$JSD(P_{G\_50} \,||\, P_{data}) = \log 2$

Not really better ……

$JSD(P_{G\_100} \,||\, P_{data}) = 0$

One simple solution: add noise

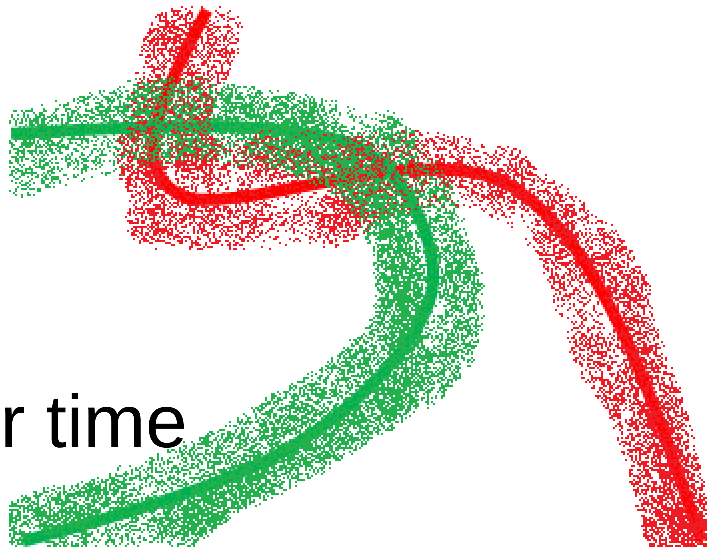
Add some artificial noise to the inputs of discriminator

Make the labels noisy for the discriminator

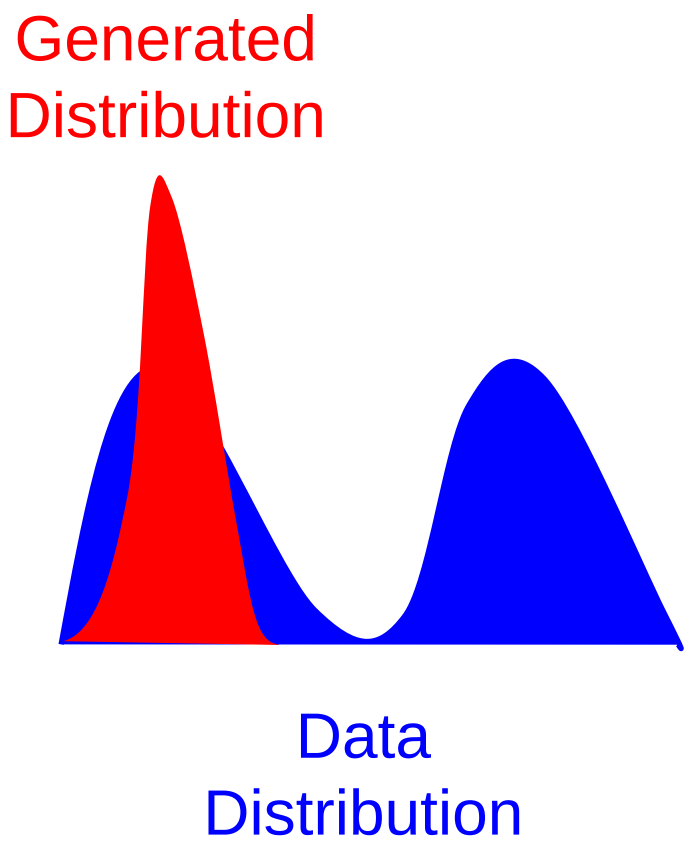
Discriminator cannot perfectly separate real and generated data

$P_{data}(x)$ and $P_G(x)$ have some overlap

Noises need to decay over time



Mode Collapse

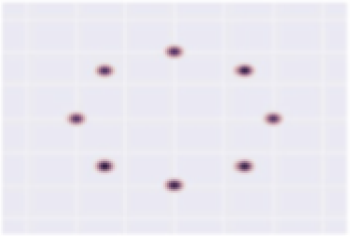


Converge to same faces

Sometimes, this is hard to tell since
one sees only what's generated, but not what's missed.

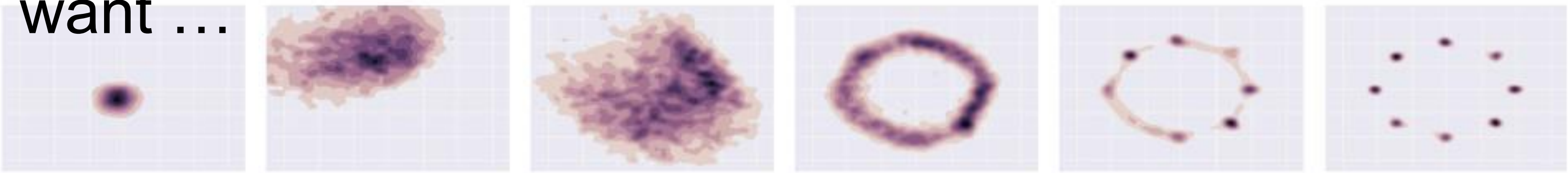
Mode Collapse Example

8 Gaussian distributions: P_{data}



Target

What we want …



Step 0 Step 5k Step 10k Step 15k Step 20k Step 25k

In reality …

Step 0 Step 5k Step 10k Step 15k Step 20k Step 25k

Text to Image, by conditional GAN

Caption	Image
a pitcher is about to throw the ball to the batter	
a group of people on skis stand in the snow	
a man in a wet suit riding a surfboard on a wave	

Text to Image - Results

"red flower with black center"

From CY Lee lecture

Caption	Image
this flower has white petals and a yellow stamen	
the center is yellow surrounded by wavy dark purple petals	
this flower has lots of small round pink petals	

Project topic: Code and data are all on web, many possibilities!

Algorithm WGAN

In each training iteration

Ian Goodfellow comment: this is also done once

- **Sample m examples $\{x^1, x^2, \ldots x^m\}$ from data distribution $P_{data}(x)$**
- **Sample m noise samples $\{z^1, \ldots, z^m\}$ from a simple prior $P_{prior}(z)$**
- **Obtain generated data $\{x^{*1}, \ldots, x^{*m}\}$, $x^{*i}=G(z^i)$**

Learning D

- **Update discriminator parameters θ_d to maximize**

$V' \approx \Sigma_{i=1..m} \log D(x^i) + 1/m\ \Sigma_{i=1..m} \log(1-D(x^{*i}))$

$\theta_d \leftarrow \theta_d + \eta \Delta V'(\theta_d)$ (gradient ascent plus weight clipping)

Repeat k times

Simple another m noise samples $\{z^1, z^2, \ldots z^m\}$ from the prior $P_{prior}(z)$, $G(z^i)=x^{*i}$

Learning G

- **Update generator parameters θ_g to minimize**

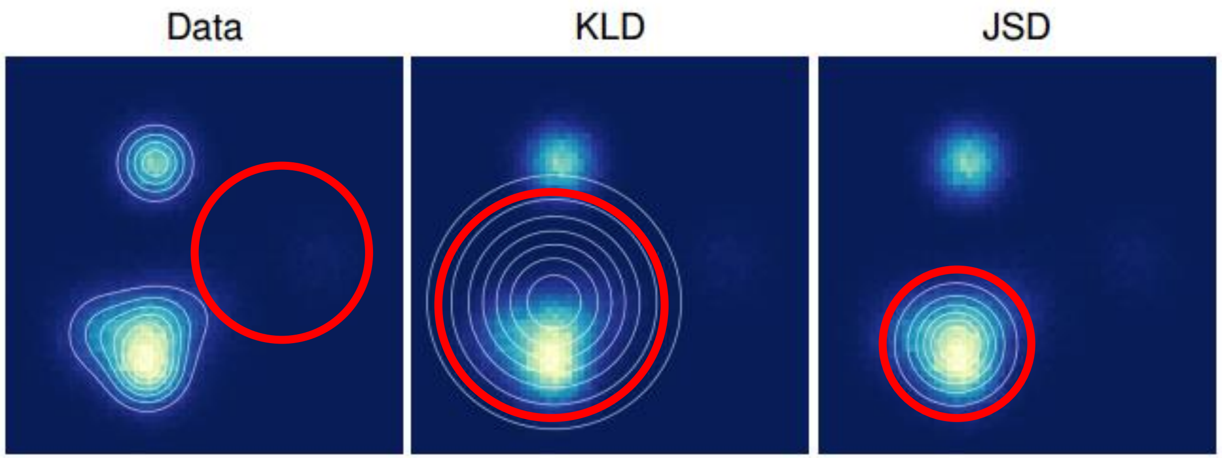
$V' = 1/m\Sigma_{i=1..m} \log D(x^i) + 1/m\ \Sigma_{i=1..m} \log(1-D(x^{*i}))$

$\theta_g \leftarrow \theta_g - \eta \Delta V'(\theta_g)$ (gradient descent)

Only Once

Experimental Results

Approximate a mixture of Gaussians by single mixture



train \ test	KL	KL-rev	JS	Jeffrey	Pearson
KL	**0.2808**	0.3423	0.1314	0.5447	0.7345
KL-rev	0.3518	**0.2414**	0.1228	0.5794	1.3974
JS	0.2871	0.2760	**0.1210**	0.5260	0.92160
Jeffrey	0.2869	0.2975	0.1247	**0.5236**	0.8849
Pearson	0.2970	0.5466	0.1665	0.7085	**0.648**

WGAN Background

We have seen that JSD does not give GAN a smooth and continuous improvement curve.

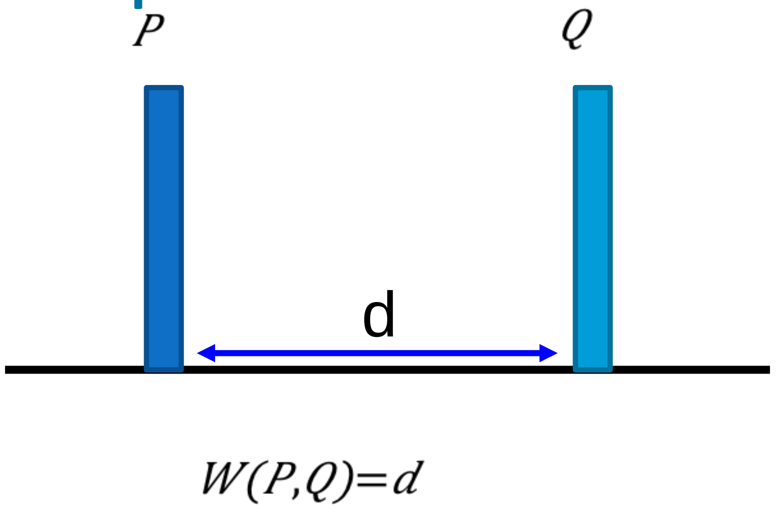
We would like to find another distance which gives that.

This is the Wasserstein Distance or earth mover’s distance.

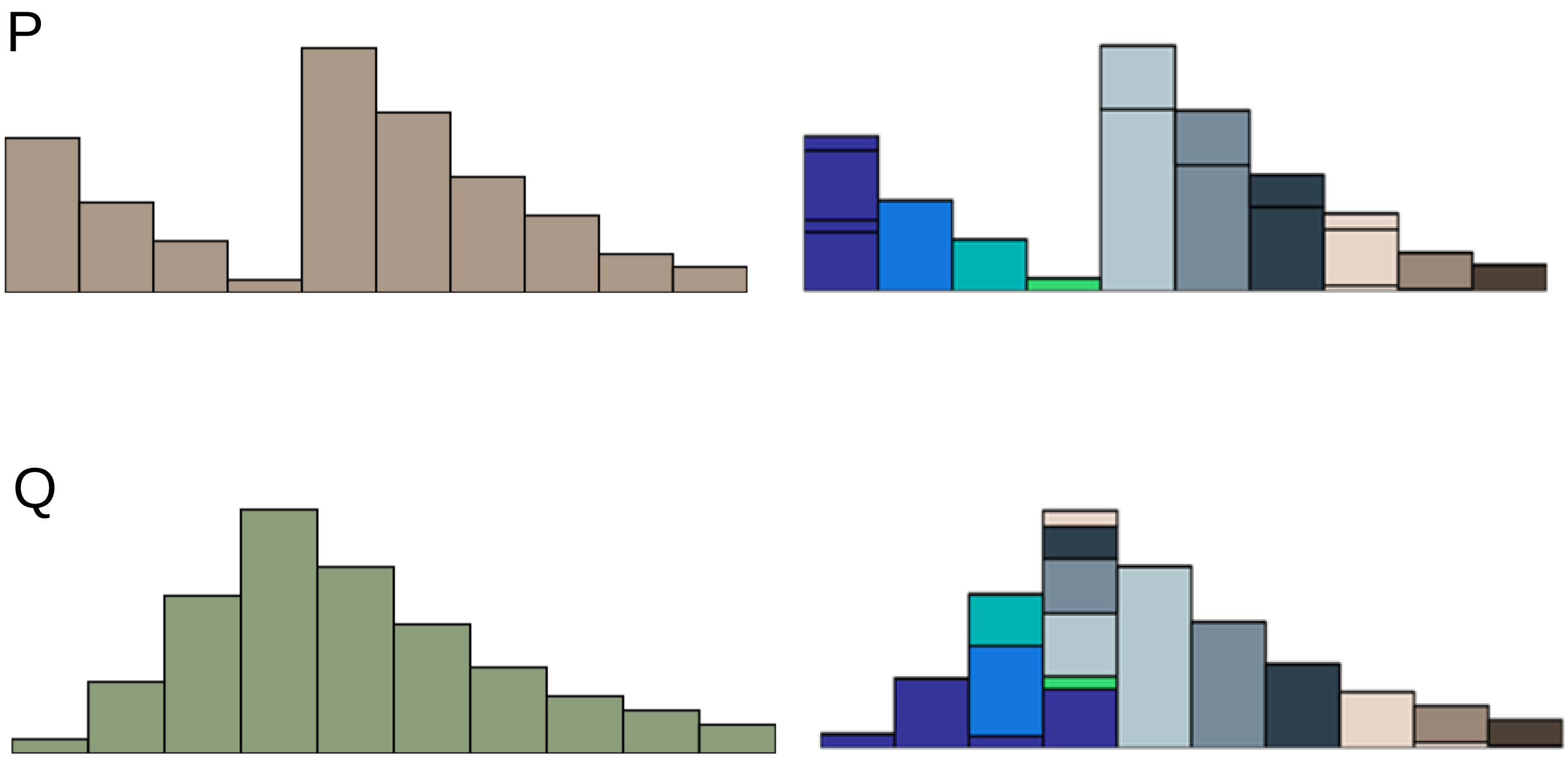
Earth Mover's Distance

Considering one distribution P as a pile of earth (total amount of earth is 1), and another distribution Q (another pile of earth) as the target

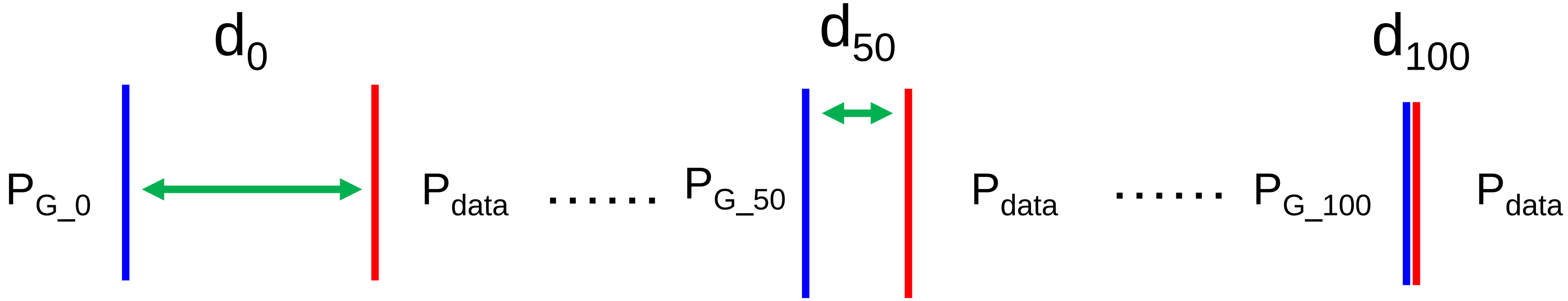
The "earth mover's distance" or "Wasserstein Distance" is the average distance the earth mover has to move the earth in an optimal plan.



Earth Mover's Distance: best plan to move



JS vs Earth Mover's Distance



$JS(P_{G\_0}, P_{data}) = \log 2$ $JS(P_{G\_50}, P_{data}) = \log 2$ $JS(P_{G\_100}, P_{data}) = 0$

$W(P_{G\_0}, P_{data}) = d_0$ $W(P_{G\_50}, P_{data}) = d_{50}$ $W(P_{G\_100}, P_{data}) = 0$

Explaining WGAN

Let W be the Wasserstein distance.

$W(P_{data}, P_G) = \max_{D \text{ is 1-Lipschitz}}[E_{x\sim P\_data}\ D(x) - E_{x\sim P\_G}\ D(x)]$

Where a function f is a

k-Lipschitz function if

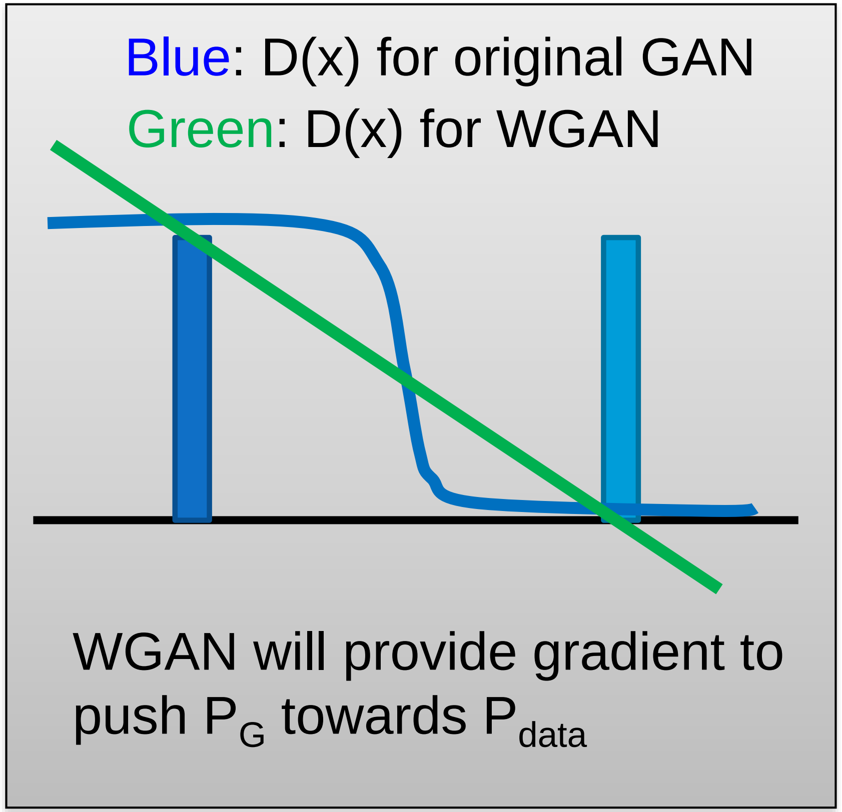
$||f(x_1) - f(x_2) \leq k||x_1 - x_2\ ||$

How to guarantee this?

Weight clipping: for all

parameter updates, if w>c

Then w=c, if w<-c, then w=-c.



Earth Mover Distance Examples:

Multi-layer perceptron