CS307: Modeling and Learning in Data Science

Bo Li

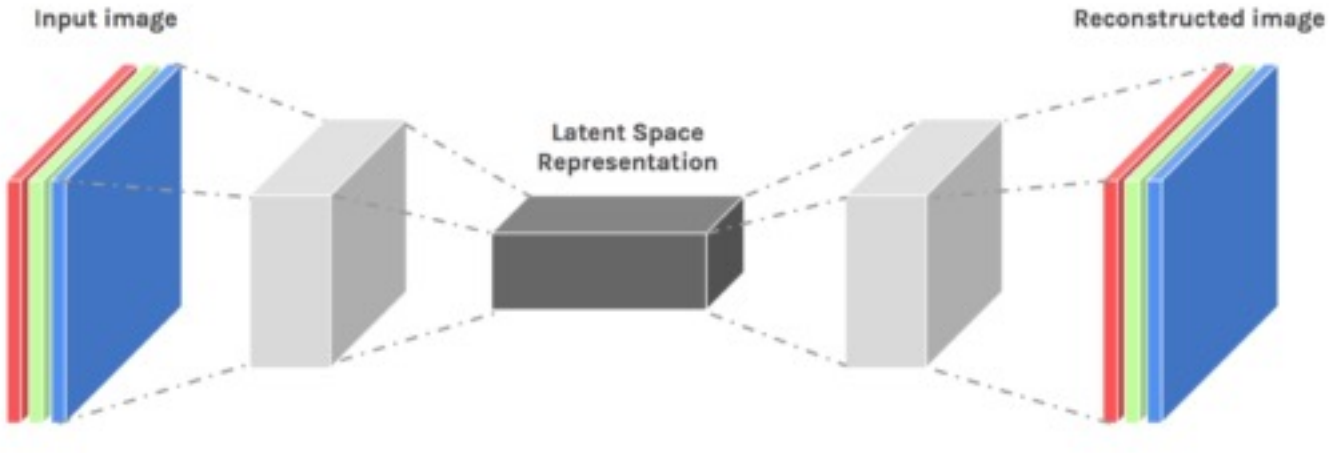
University of Illinois at Urbana-Champaign

Goal of today's class

- Recall Sammon mapping, T-SNE, AE
- VAE
- Generative adversarial networks (GANs)

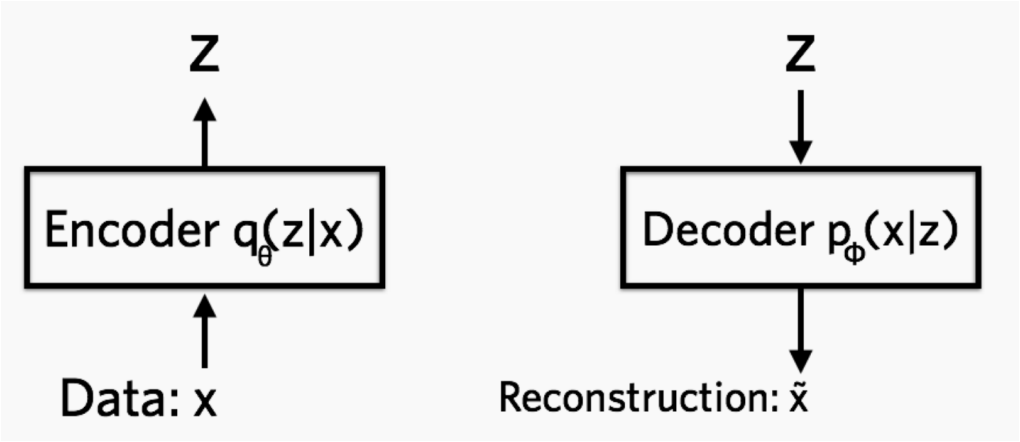
Autoencoders

- Autoencoders are designed to reproduce their input, especially for images.
  - Key point is to reproduce the input from a learned encoding.



Variational Autoencoder (VAE)

- Key idea: make both the encoder and the decoder probabilistic.
- I.e., the latent variables, z, are drawn from a probability distribution depending on the input, X, and the reconstruction is chosen probabilistically from z.

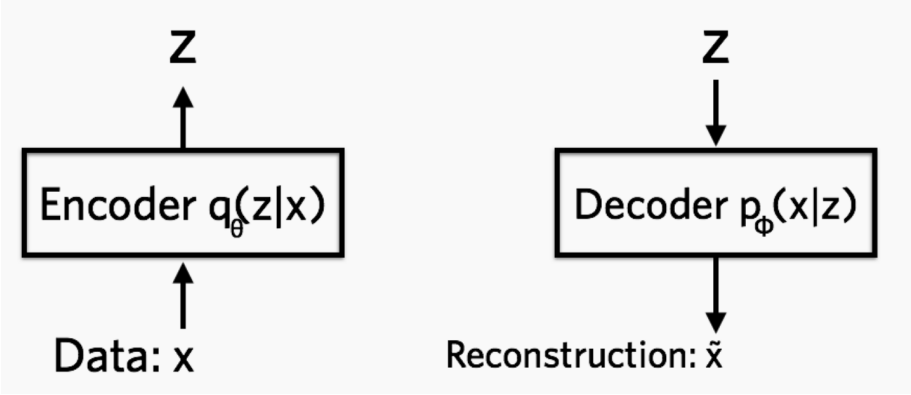


VAE Encoder

- The encoder takes input and returns parameters for a probability density (e.g., Gaussian): I.e., $q_\theta(z \mid x)$ gives the mean and co-variance matrix.
- We can sample from this distribution to get random values of the lower-dimensional representation *z*.
- Implemented via a neural network: each input *x* gives a vector mean and diagonal covariance matrix that determine the Gaussian density $q_\theta(z \mid x)$
- Parameters θ for the NN need to be learned – need to set up a loss function.

VAE Decoder

- The decoder takes latent variable *z* and returns parameters for a distribution. E.g., $p_\phi(x|z)$ gives the mean and variance for each pixel in the output.
- Reconstruction $\tilde{x}$ is produced by sampling.
- Implemented via neural network, the NN parameters ϕ are learned.



VAE loss function

- Loss function for autoencoder: L_2 distance between output and input (or clean input for denoising case)
- For VAE, we need to learn parameters of two probability distributions. For a single input, x_i, we maximize the expected value of returning x_i or minimize the expected negative log likelihood.

$$-\mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)]$$

- This takes expected value wrt z over the current distribution $q_\theta(z|x_i)$ of the loss $-\log p_\phi(x_i|z)$

VAE loss function

- **Problem:** the weights may adjust to memorize input images via *z.* I.e., input that we regard as similar may end up very different in *z* space.
- We prefer continuous latent representations to give meaningful parameterizations. E.g., smooth changes from one digit to another.
- **Solution:** Try to force $q_\theta(z|x_i)$ to be close to a standard normal (or some other simple density).

VAE loss function

- For a single data point x_i we get the loss function

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \,||\, p(z))$$

- The first term promotes recovery of the input.
- The second term keeps the encoding continuous – the encoding is compared to a fixed $p(z)$ regardless of the input, which inhibits memorization.
- With this loss function the VAE can (almost) be trained using gradient descent on minibatches.

VAE loss function

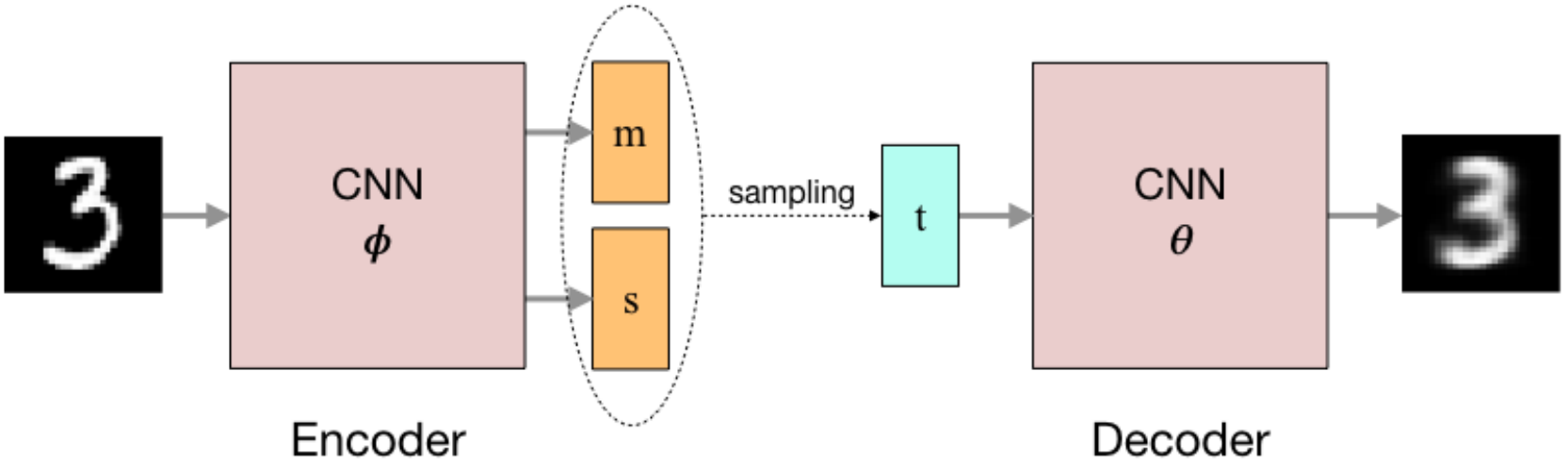
- For a single data point x_i we get the loss function

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \,||\, p(z))$$

- **Problem:** The expectation would usually be approximated by choosing samples and averaging. This is not differentiable wrt θ and ϕ.

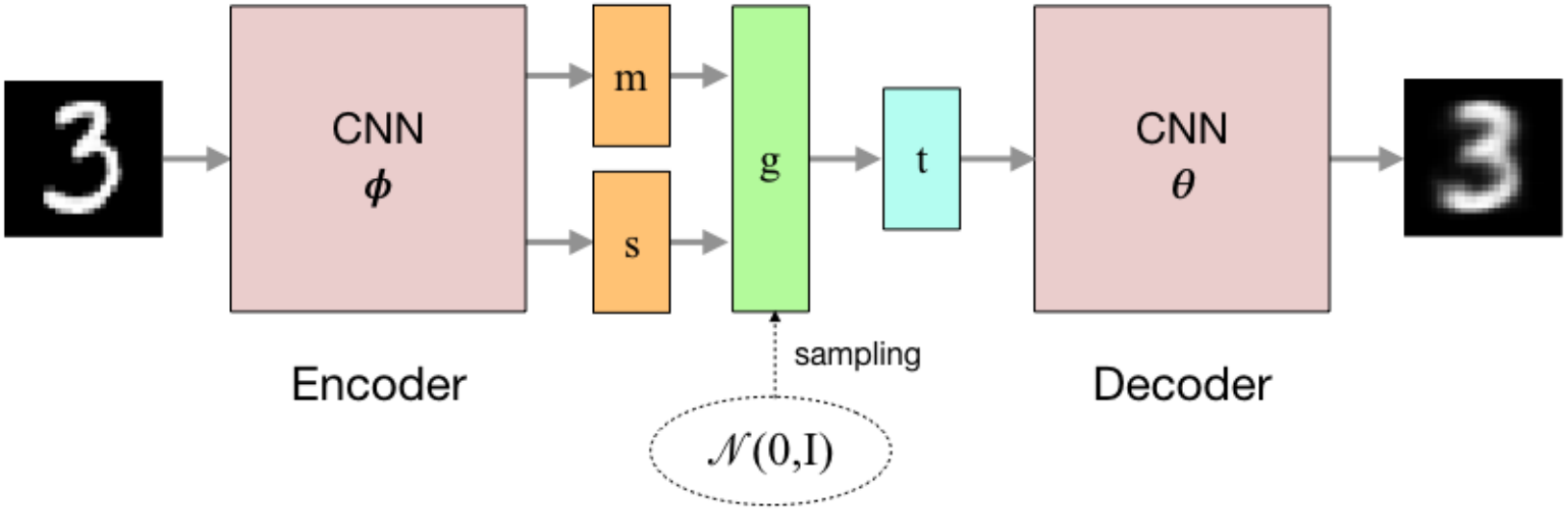
VAE loss function

- **Problem:** The expectation would usually be approximated by choosing samples and averaging. This is not differentiable wrt θ and ϕ.



VAE loss function

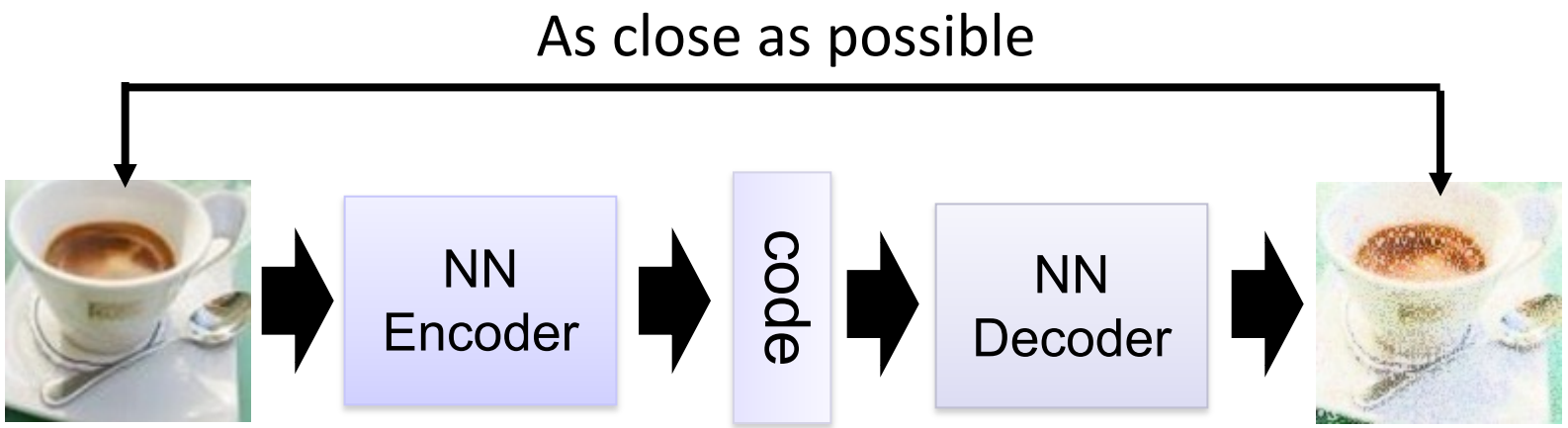
- **Reparameterization:** If *z* is $N(\mu(x_i), \Sigma(x_i))$, then we can sample *z* using $z = \mu(x_i) + \sqrt{(\Sigma(x_i))}\,\epsilon$, where ϵ is *N*(0,1). So we can draw samples from *N*(0,1), which doesn't depend on the parameters.



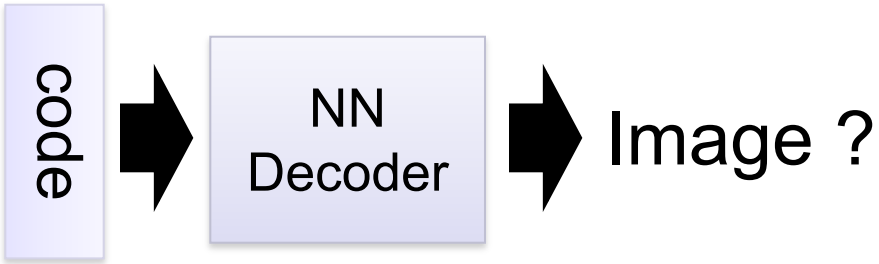
VAE generative model

- After training, $q_\theta(z|x_i)$ is close to a standard normal, N(0,1) – easy to sample.
- Using a sample of *z* from $q_\theta(z|x_i)$ as input to sample from $p_\phi(x|z)$ gives an approximate reconstruction of x_i, at least in expectation.
- If we sample any *z* from N(0,1) and use it as input to to sample from $p_\phi(x|z)$ then we can approximate the entire data distribution *p*(*x*). I.e., we can generate new samples that look like the input but aren't in the input.

Autoencoder

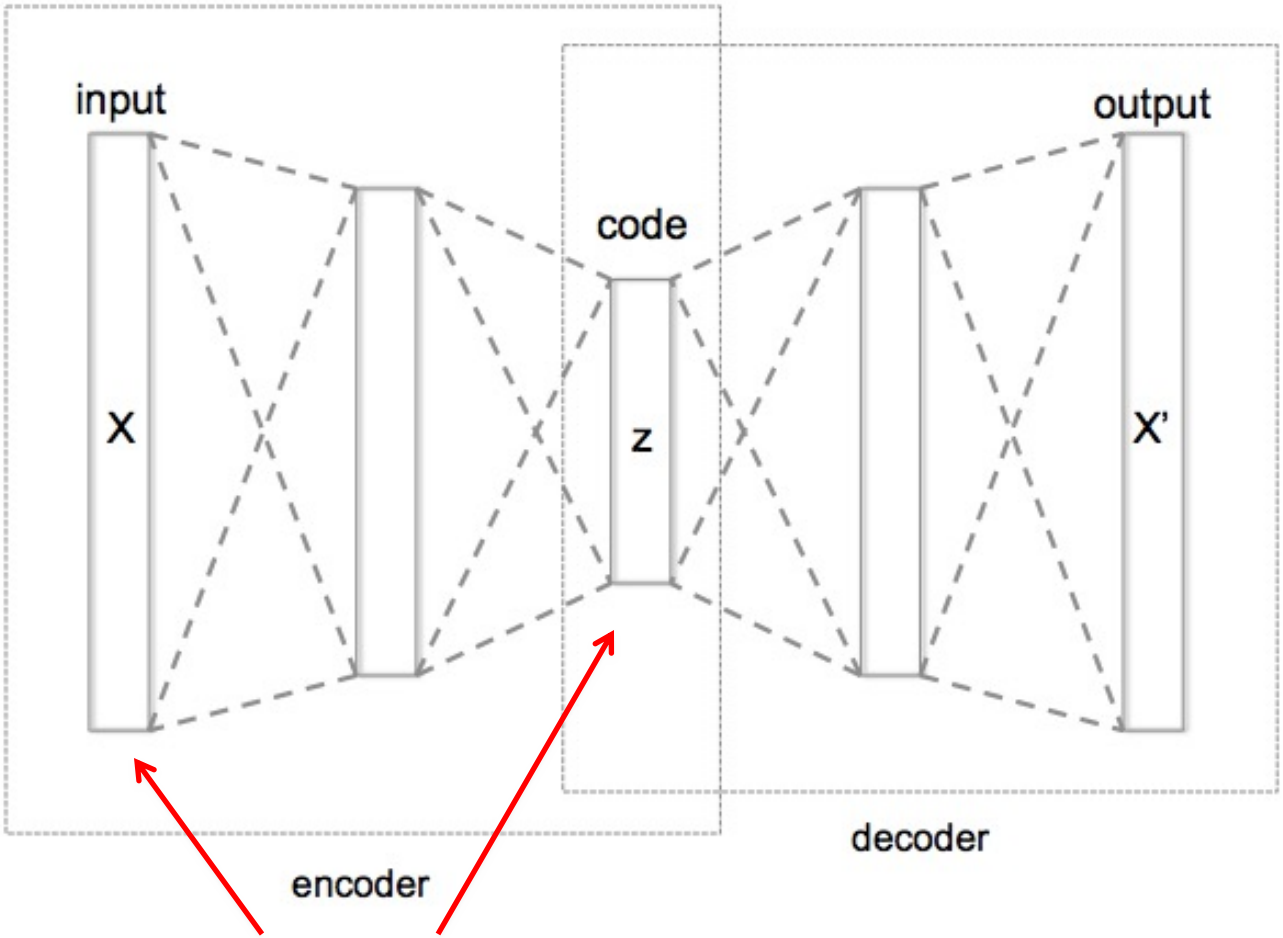


Randomly generate a vector as code



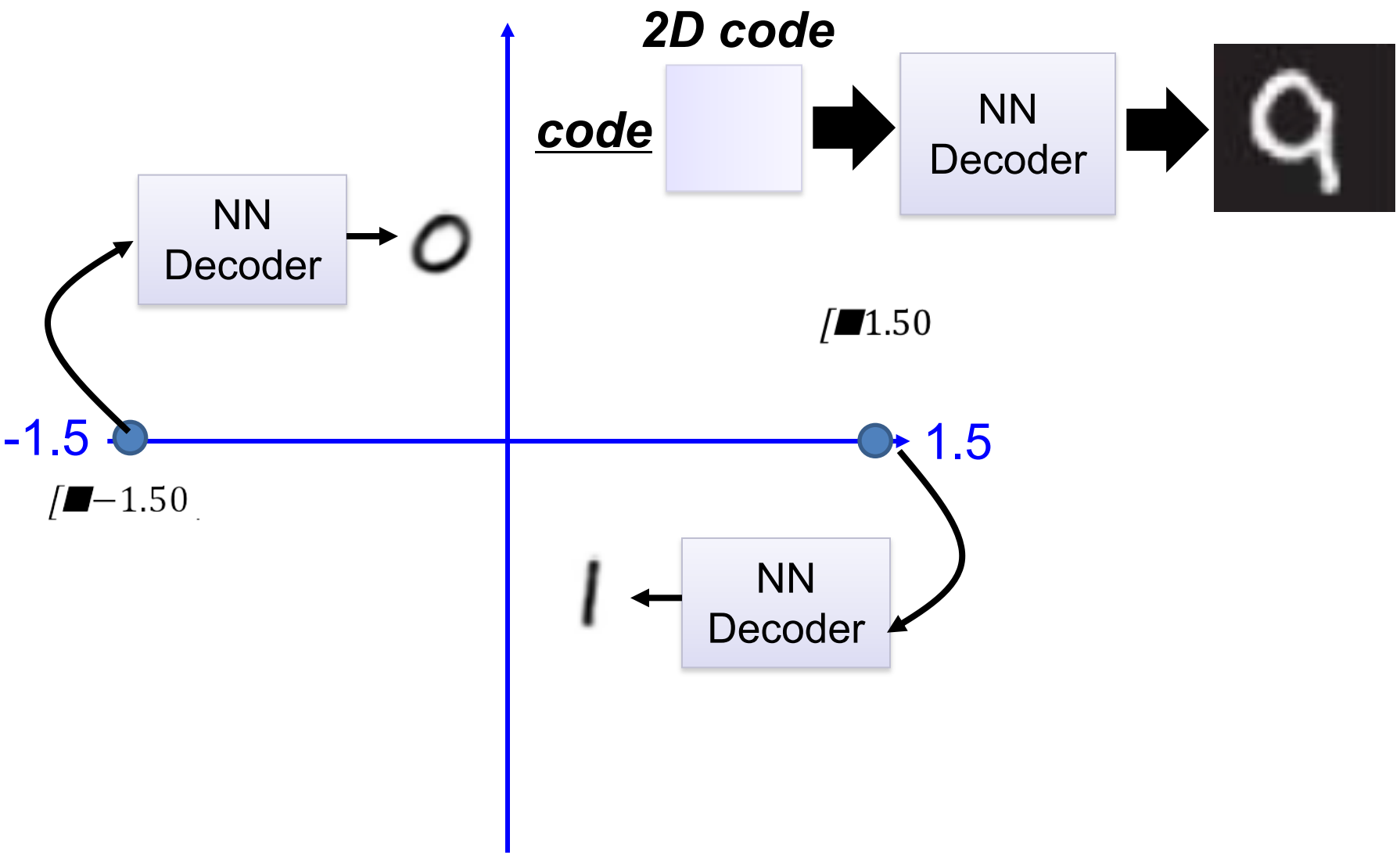
Autoencoder with 3 fully connected layers

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

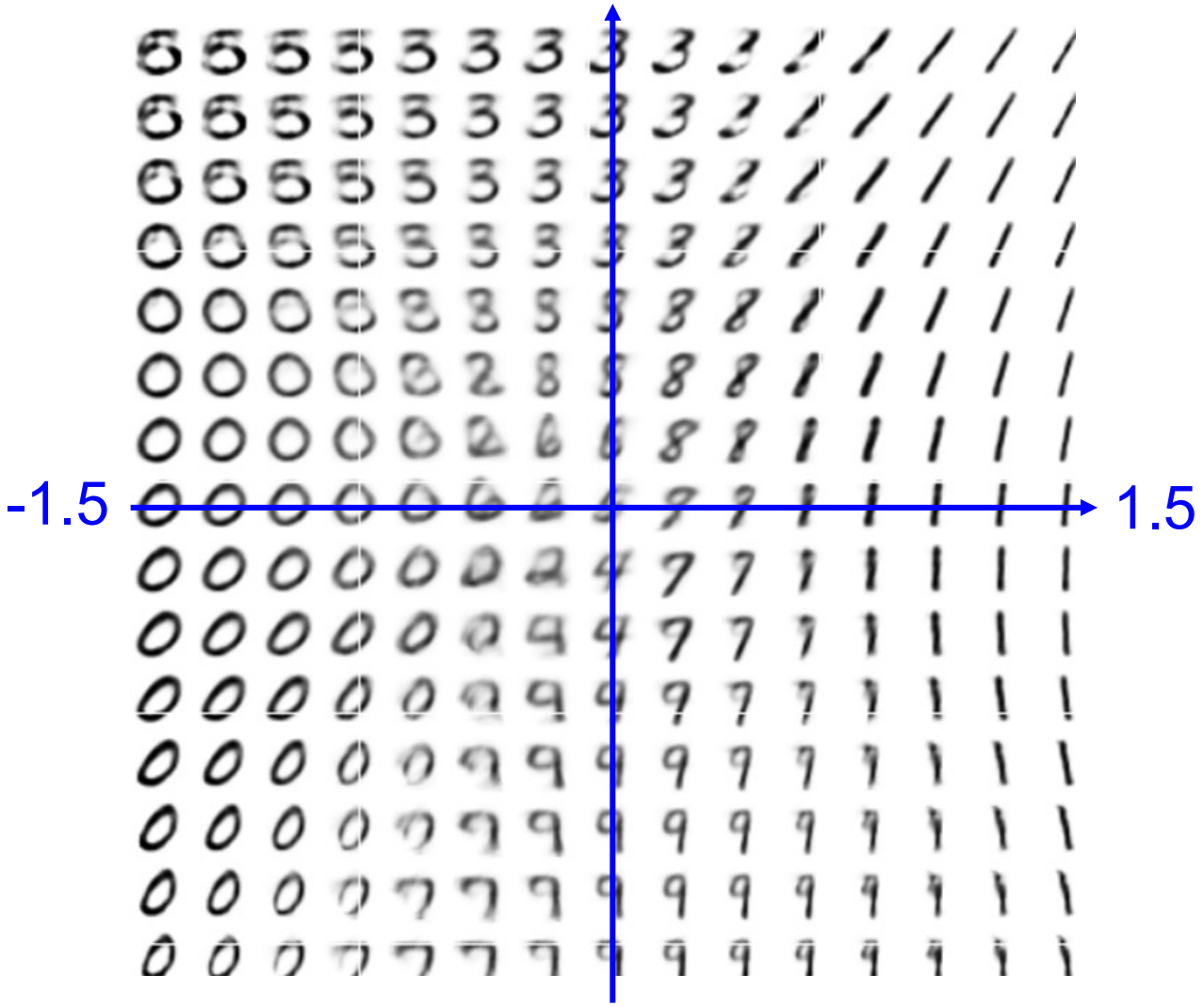


Large → small, learn to compress

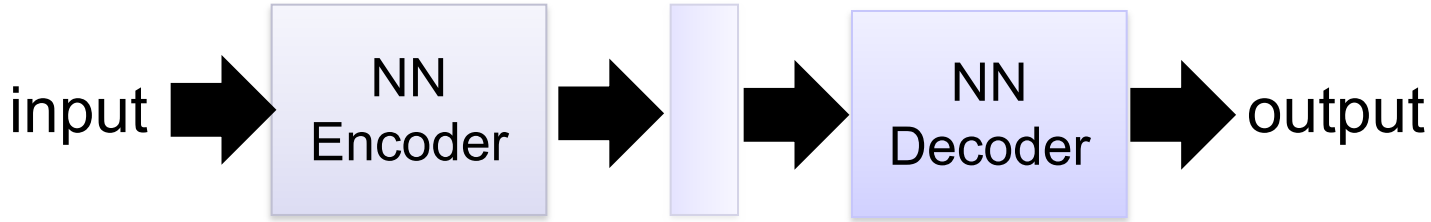
Auto-encoder



Auto-encoder

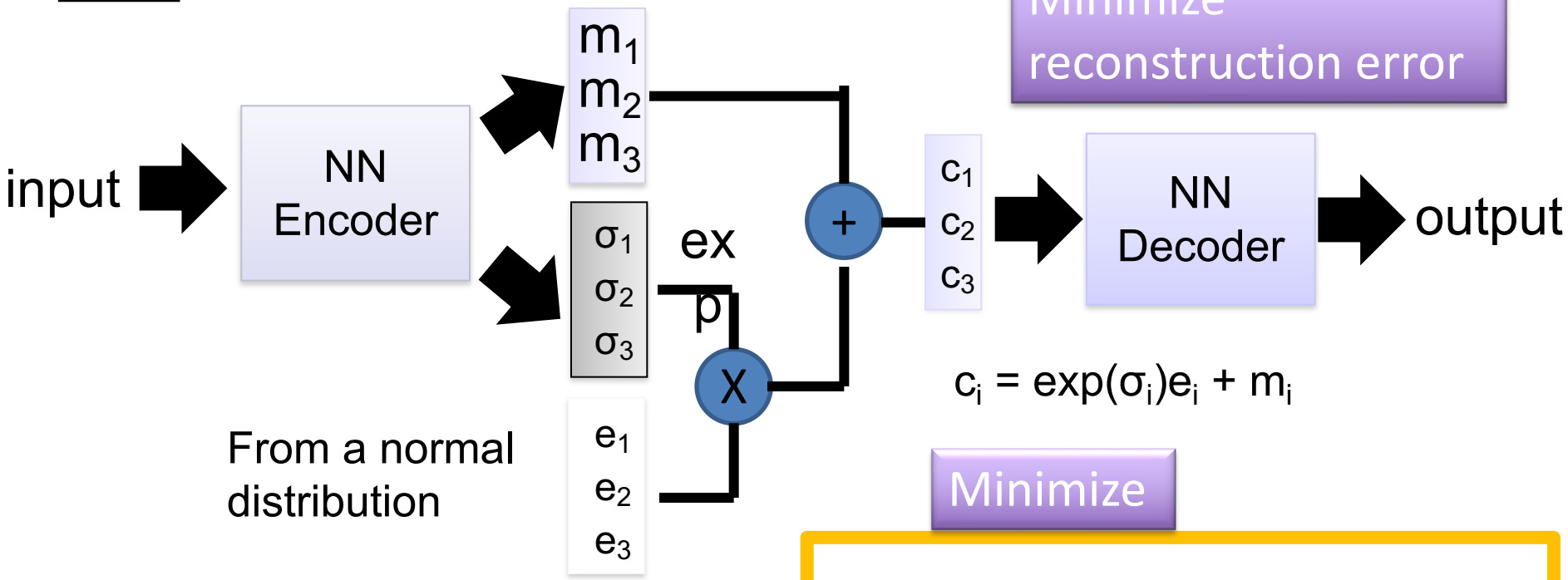


Auto-encoder



code

VAE



$c_i = \exp(\sigma_i)e_i + m_i$

Minimize

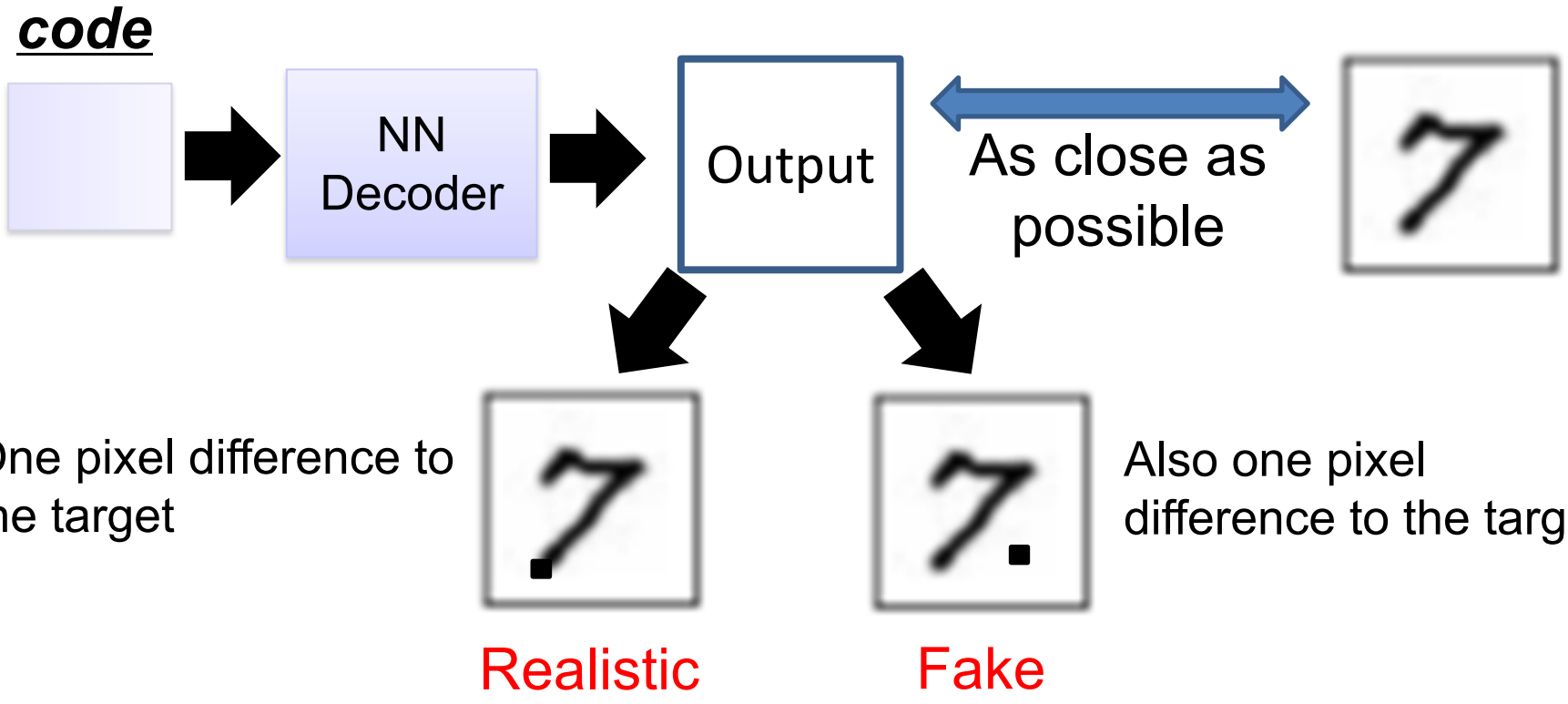
$$\Sigma_{i=1..3} [\exp(\sigma_i) - (1+\sigma_i) + (m_i)^2]$$

This constrains σ_i approacing 0 is good

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

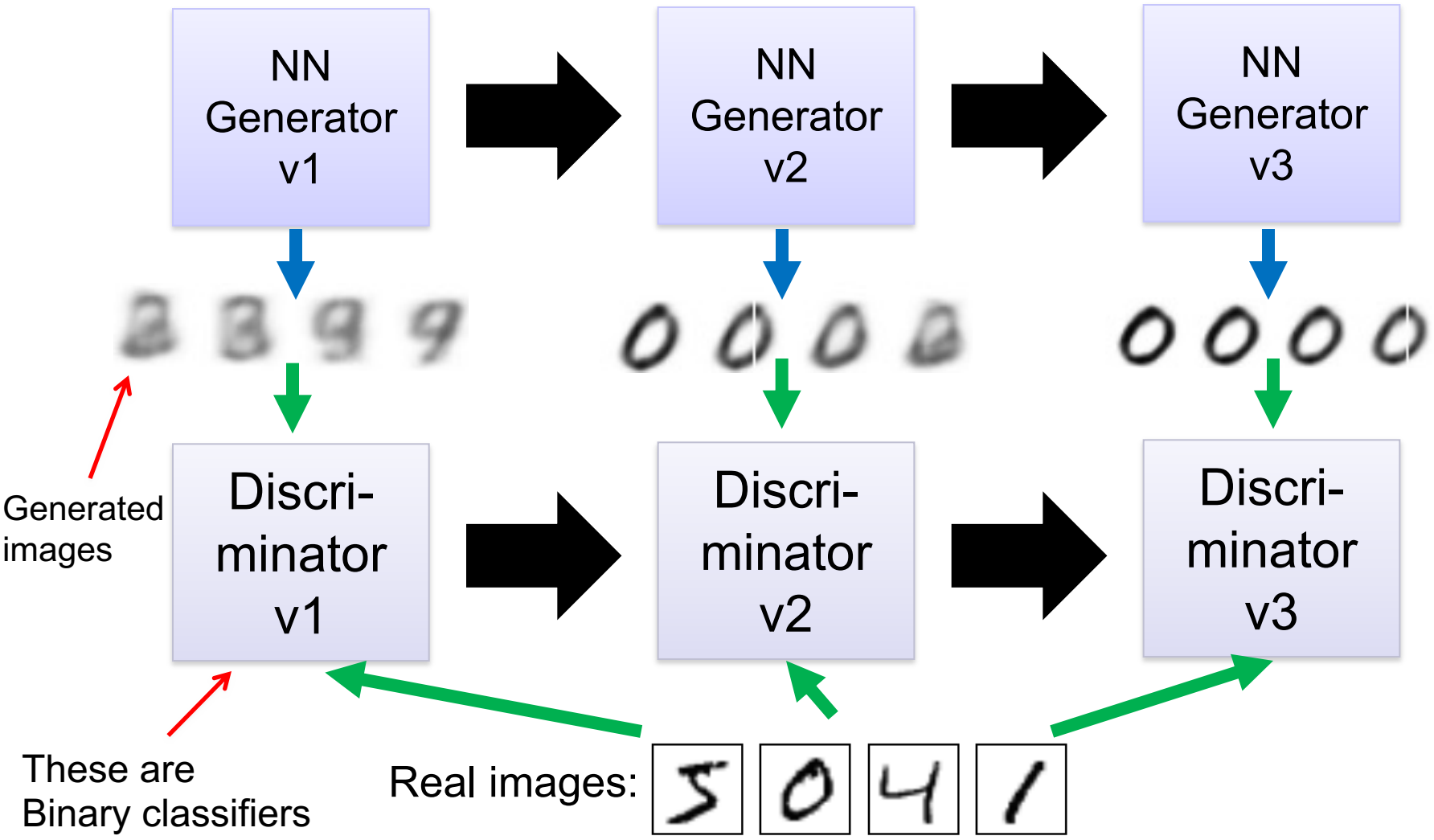
Problems of VAE

- It does not really try to simulate real images



VAE treats these the same

Gradual and step-wise generation

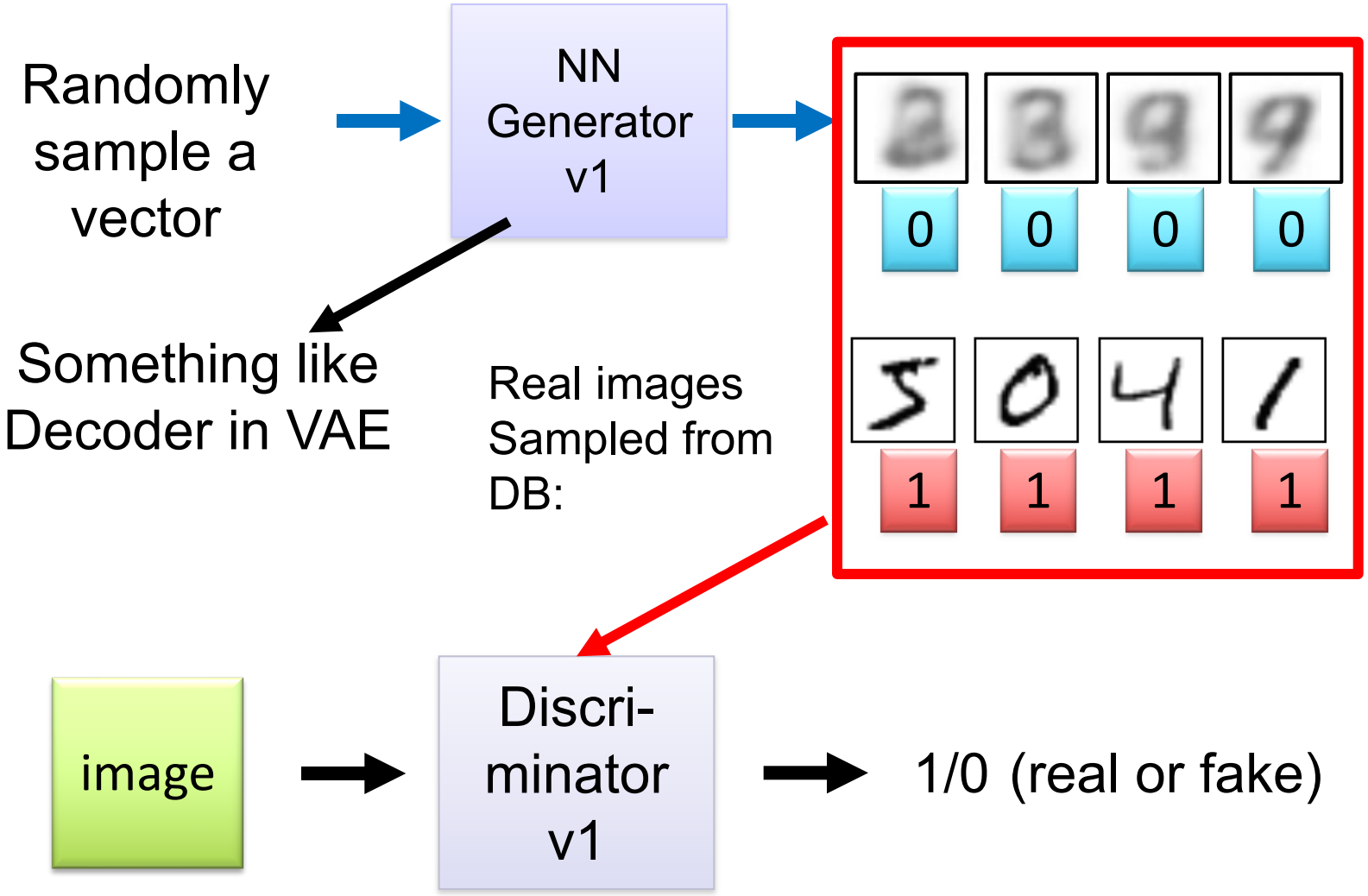


Generative Adversarial Networks

- GAN was first introduced by Ian Goodfellow et al in 2014
- Have been used in generating images, videos, poems, some simple conversation.
- Note, image processing is easy (all animals can do it), NLP is hard (only human can do it).
- This co-evolution approach might have far-reaching implications. Bengio: this may hold the key to making computers a lot more intelligent.

- Tips for training GAN: https://github.com/soumith/ganhacks

GAN – Learn a discriminator



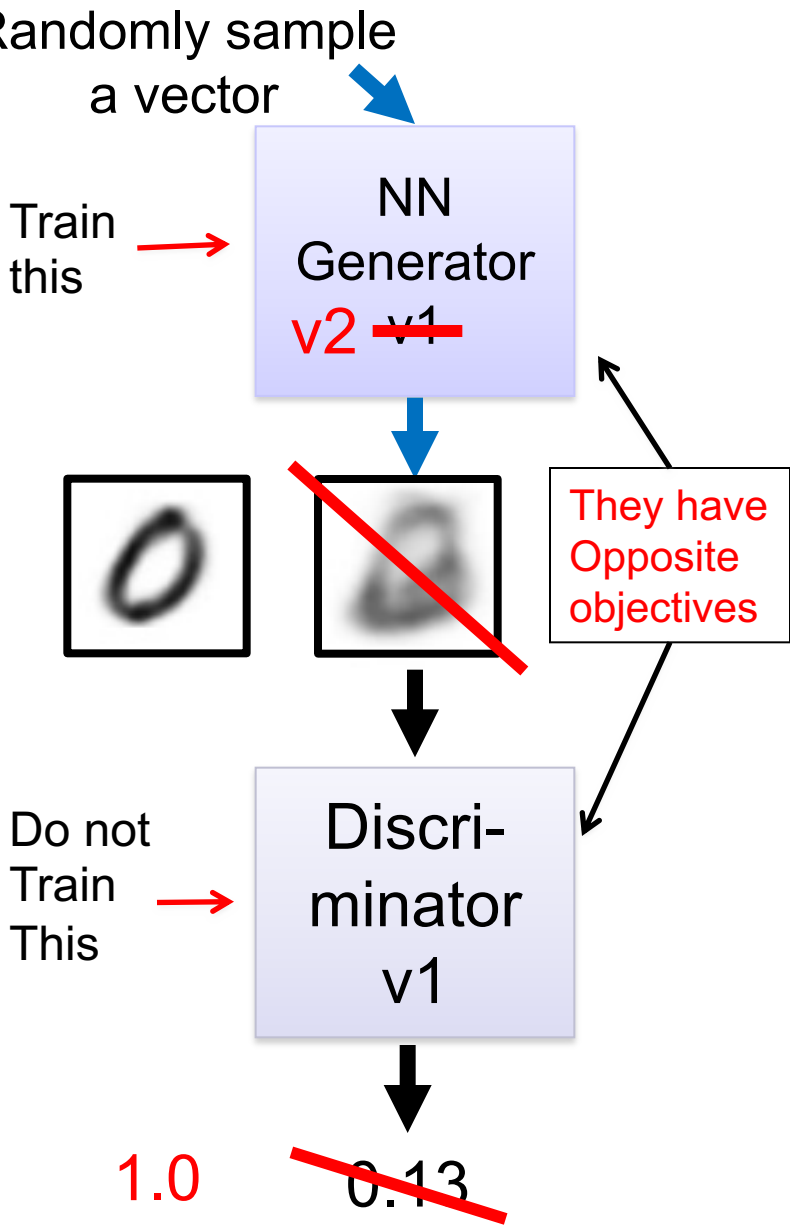
GAN – Learn a generator

Updating the parameters of generator

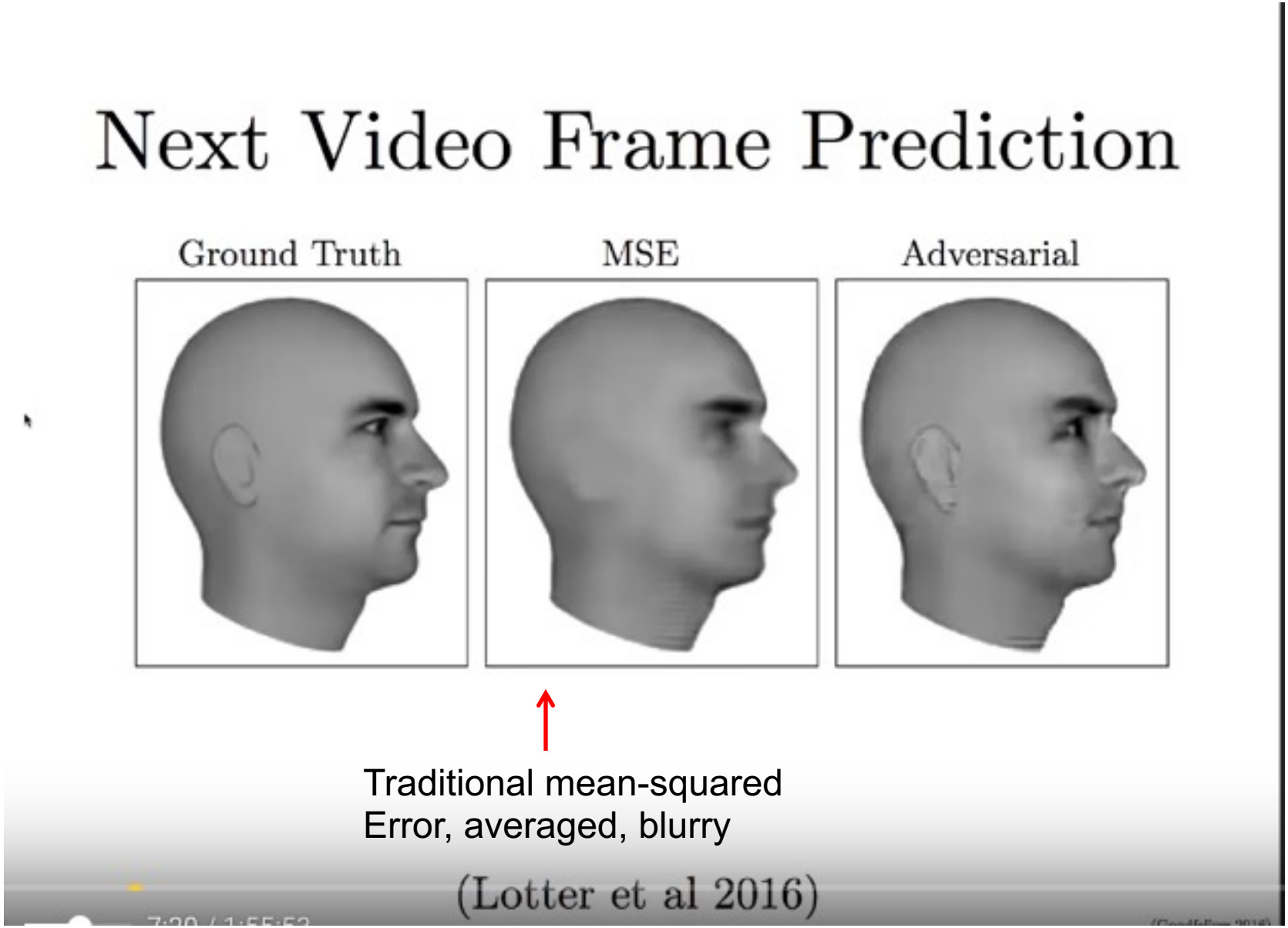
The output be classified as “real” (as close to 1 as possible)

Generator + Discriminator = a network

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



Next few images

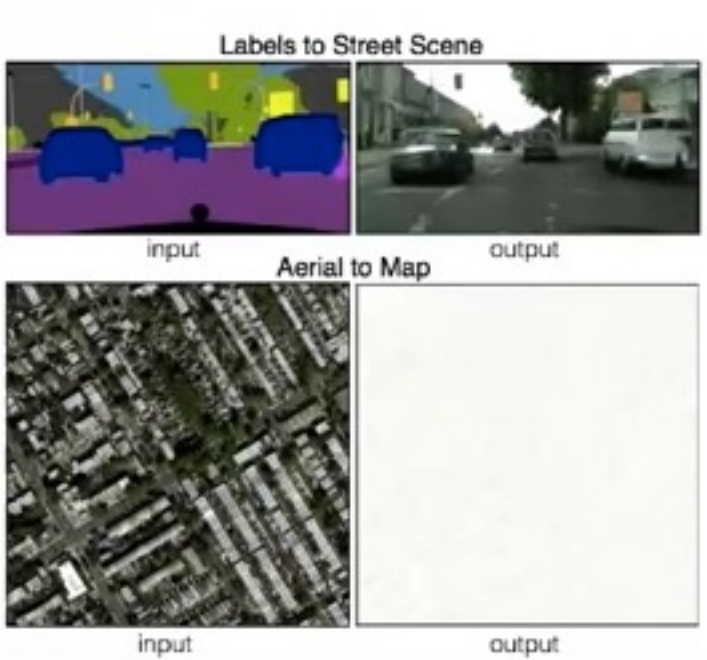


Single Image Super-Resolution

(Ledig et al 2016)

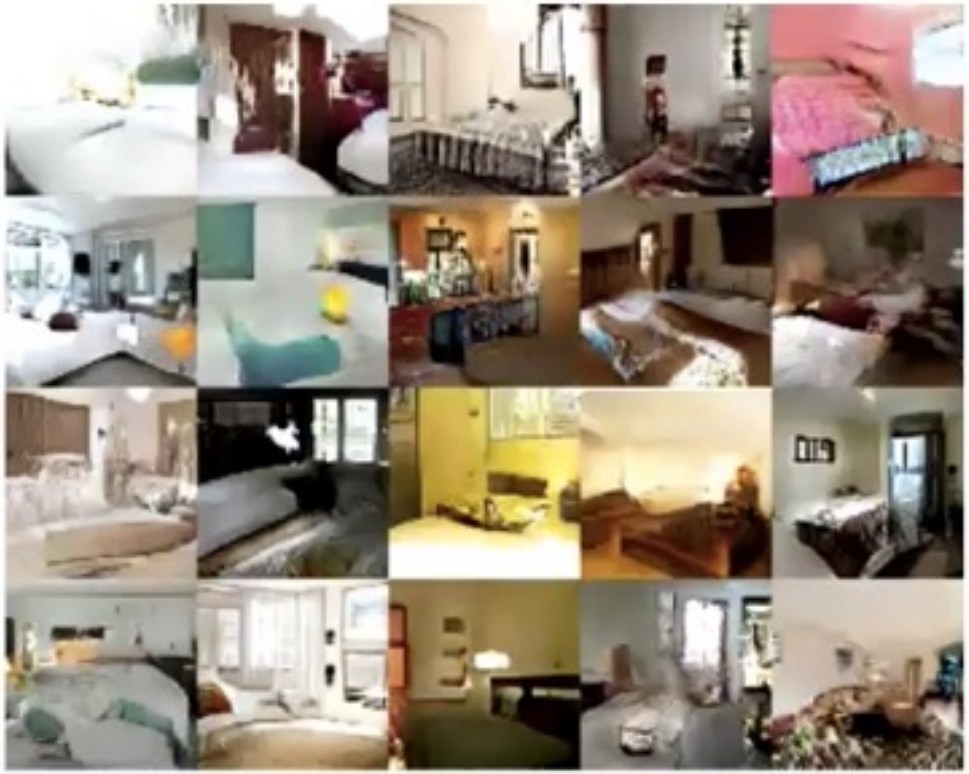
Last 2 are by deep learning approaches.

Image to Image Translation



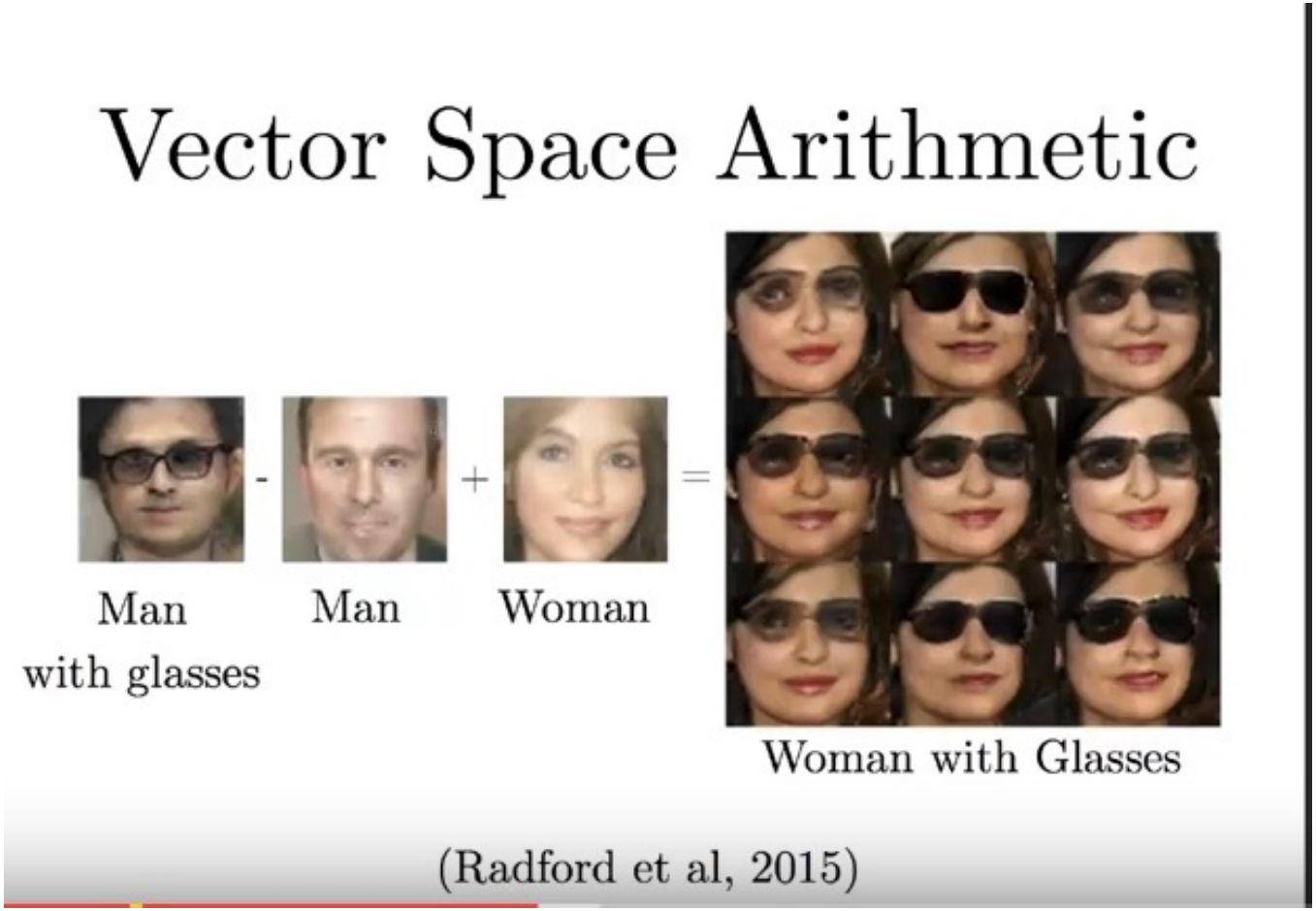
(Isola et al 2016)

DCGANs for LSUN Bedrooms

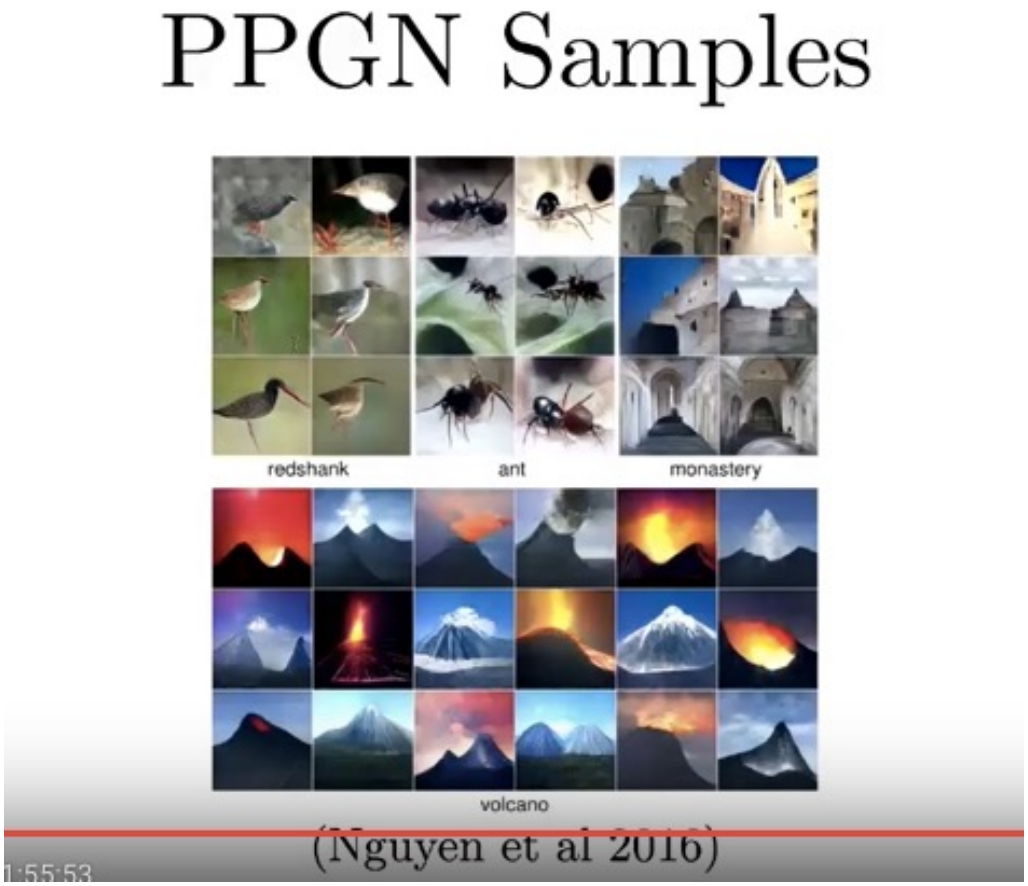


(Radford et al 2015)

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

- How to derive and train GAN
- I will avoid the continuous case and stick to simple explanations.

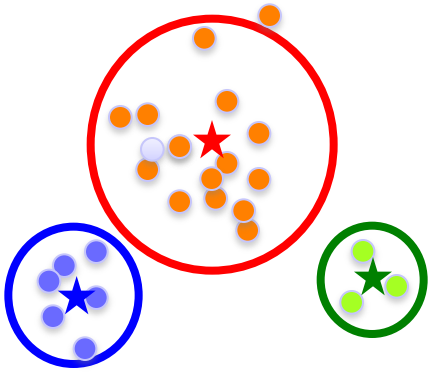
Maximum Likelihood Estimation

- 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

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

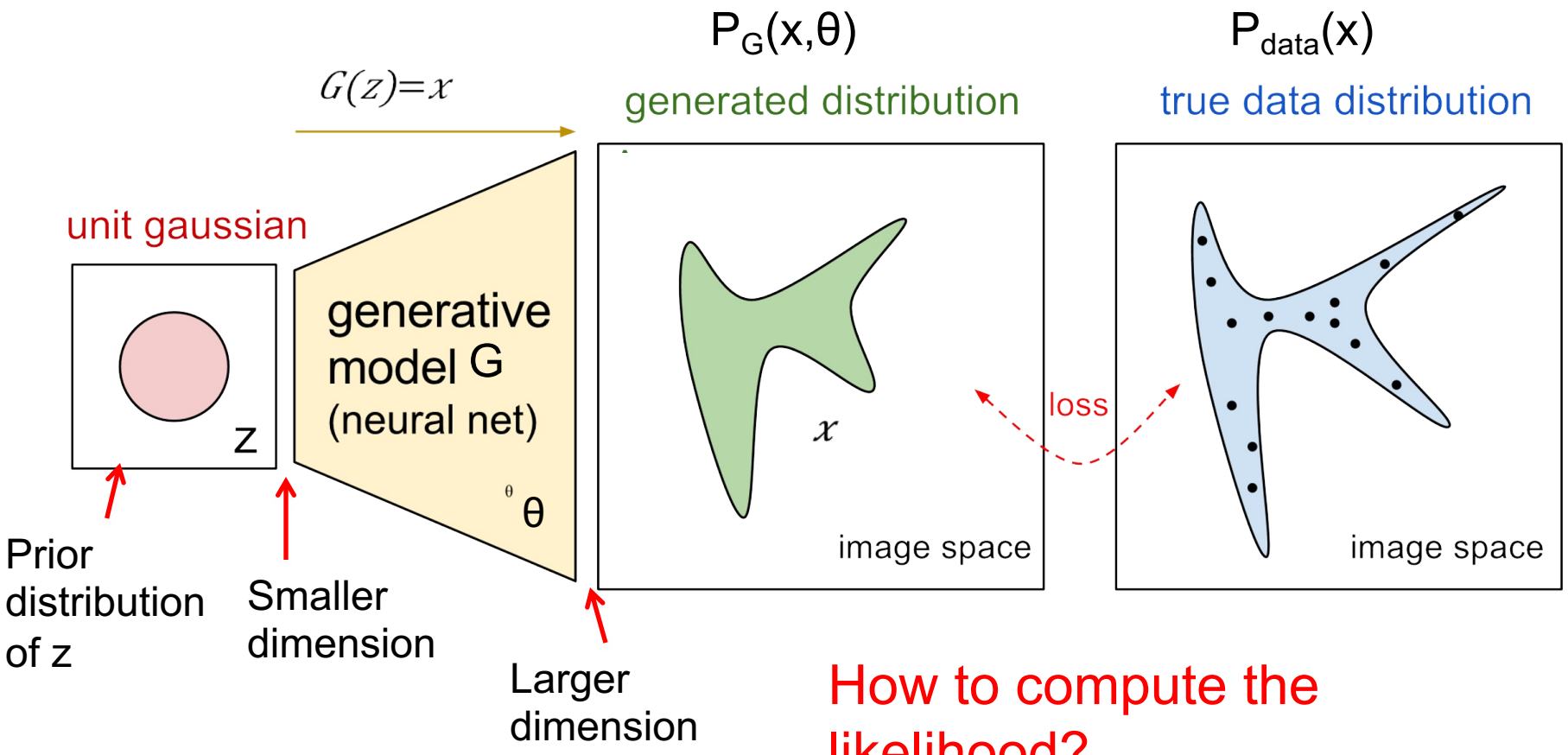
$\cong \arg\max_\theta \Sigma_{i=1..m} P_{data}(x^i) \log P_G(x^i; \theta) - \Sigma_{i=1..m} P_{data}(x^i) \log P_{data}(x^i)$

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



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

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):

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

Note, we are changing distribution G, not just update its parameters (as in the max likelihood case).

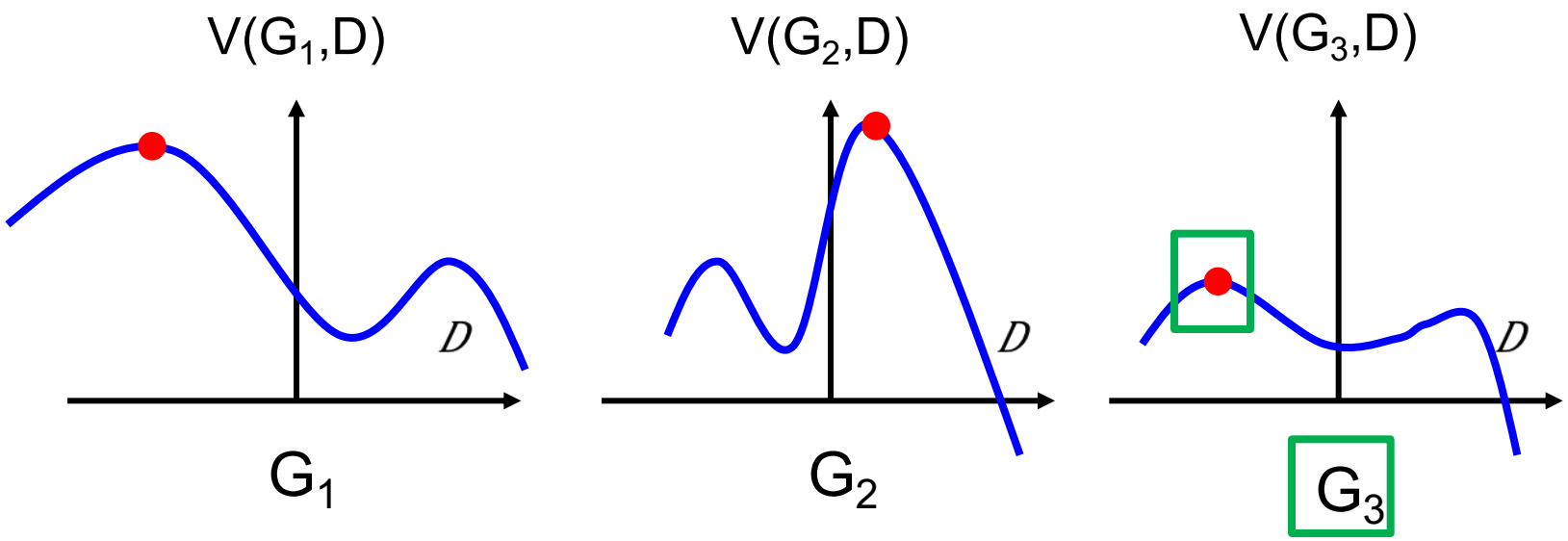
Basic Idea

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

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}



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

- 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

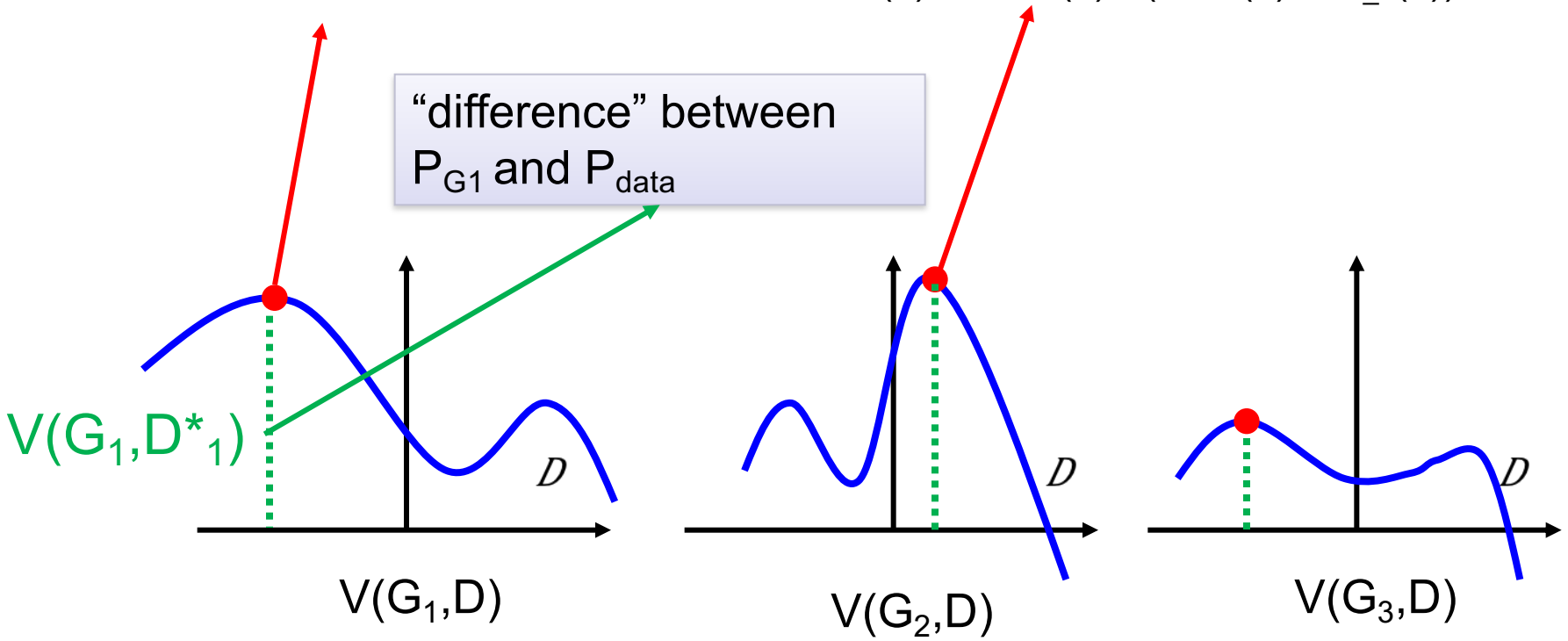
- Given x, the optimal D* maximizing is:

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

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

$D_1^*(x) = P_{data}(x) / (P_{data}(x)+P_{G\_1}(x))$

$D_2^*(x) = P_{data}(x) / (P_{data}(x)+P_{G\_2}(x))$



$\max_D V(G,D)$

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

$\max_D V(G,D)$

$= V(G,D^*)$, where $D^*(x) = P_{data} / (P_{data} + P_G)$, and

$1-D^*(x) = P_G / (P_{data} + P_G)$

$= E_{x\sim P\_data} \log D^*(x) + E_{x\sim P\_G} \log (1-D^*(x))$

$\approx \Sigma [P_{data}(x) \log D^*(x) + P_G(x) \log (1-D^*(x))]$

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

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

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

L(G), this is the loss function

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

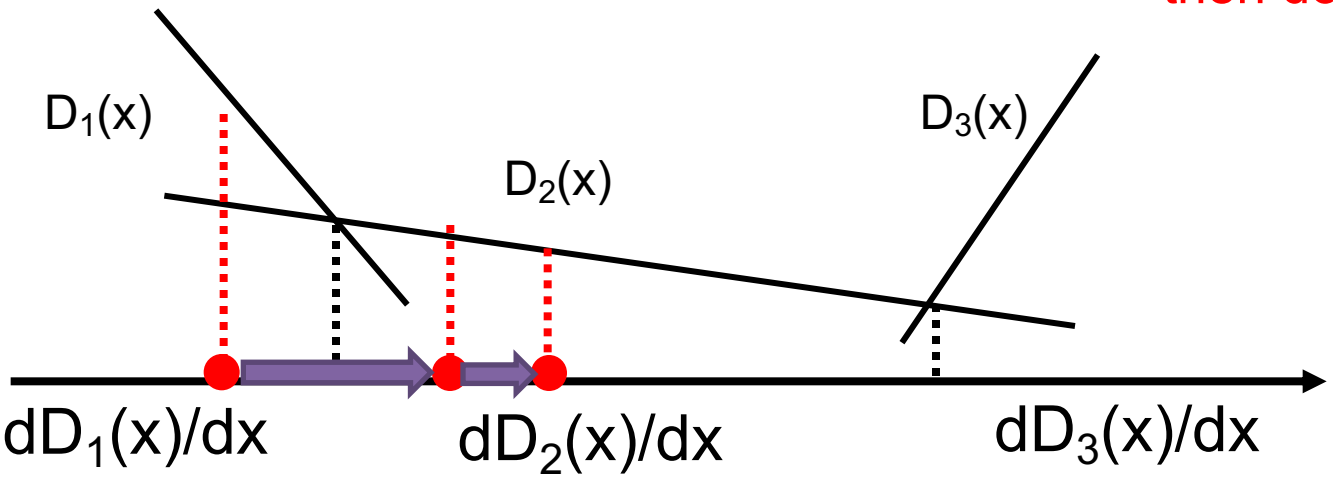
  $\theta_G \leftarrow \theta_G - \eta \partial L(G)/\partial \theta_G$, θ_G defines G

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

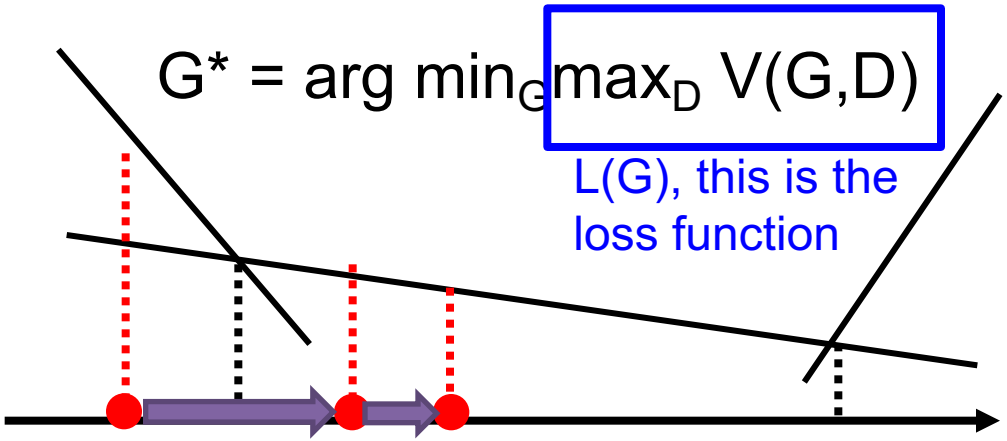
  $f(x) = \max\{D_1(x), D_2(x), D_3(x)\}$

$df(x)/dx = ?$

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



Algorithm



- 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 $\{x^1, \dots, x^m\}$ from P_{data}
  - Sample $\{x^{*1}, \dots, x^{*m}\}$ 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

Learning D

Repeat k times

- 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
  - $V' \approx 1/m \sum_{i=1..m} \log D(x^i) + 1/m \sum_{i=1..m} \log(1-D(x^{*i}))$
  - $\theta_d \leftarrow \theta_d + \eta \Delta V'(\theta_d)$ (gradient ascent)

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

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

$$\theta_g \leftarrow \theta_g - \eta \Delta V'(\theta_g) \quad \text{(gradient descent)}$$

Objective Function for Generator in Real Implementation

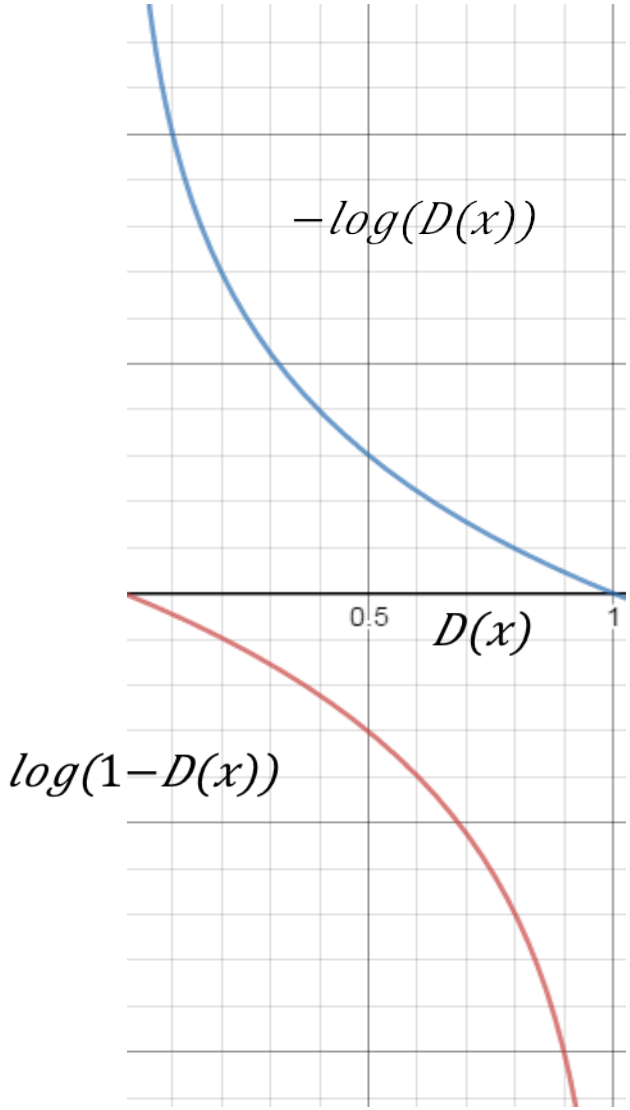
$V = E_{x \sim P\_data}[\log D(x)$

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

Training slow at the beginning

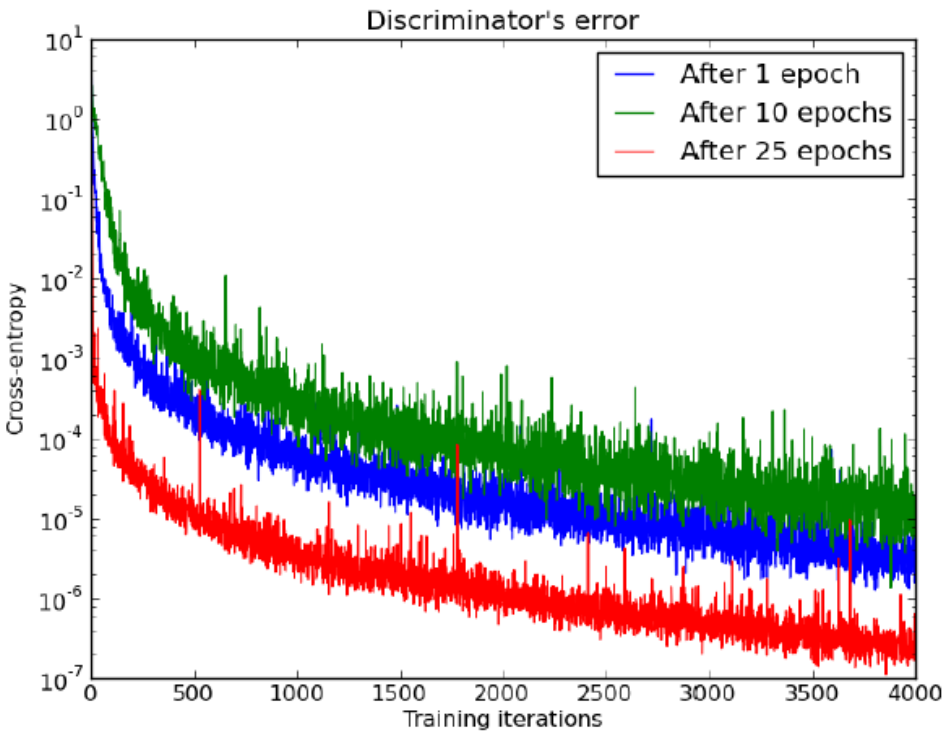
$V = E_{x \sim P\_G}[-\log(D(x))]$

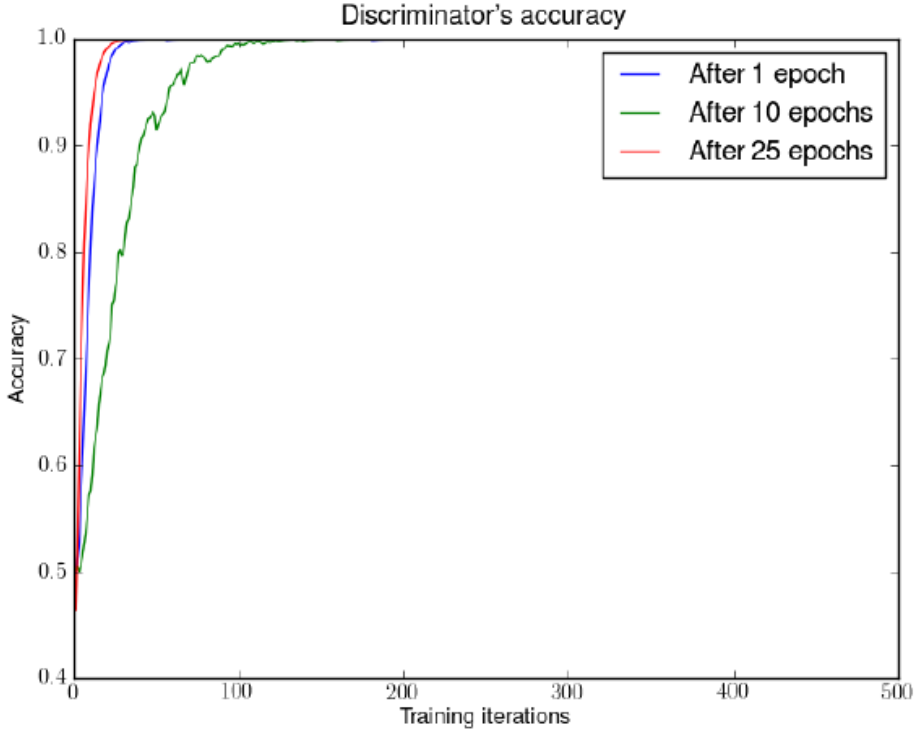
Real implementation:
label x from P_G as positive



Evaluating JS divergence

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

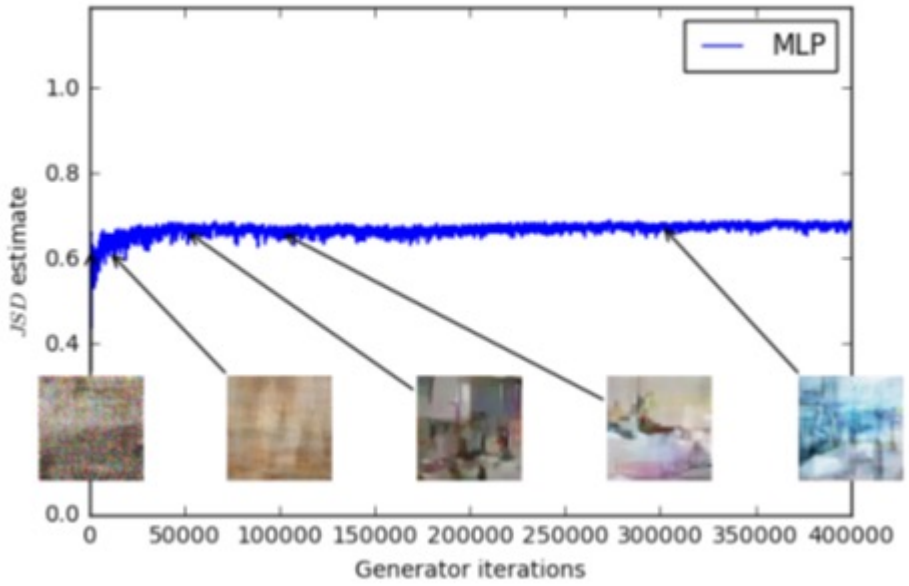




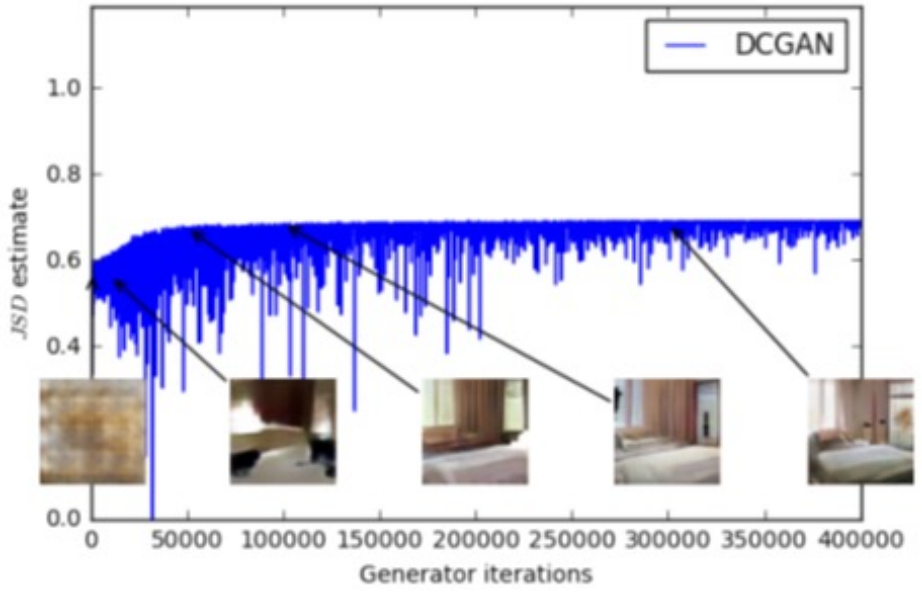
Martin Arjovsky, Léon Bottou, Towards Principled Methods for Training Generative Adversarial Networks, 2017, arXiv preprint

Evaluating JS divergence

- JS divergence estimated by discriminator telling little information



Weak Generator



Strong Generator

Discriminator

1 for all positive examples

0 for all negative examples

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

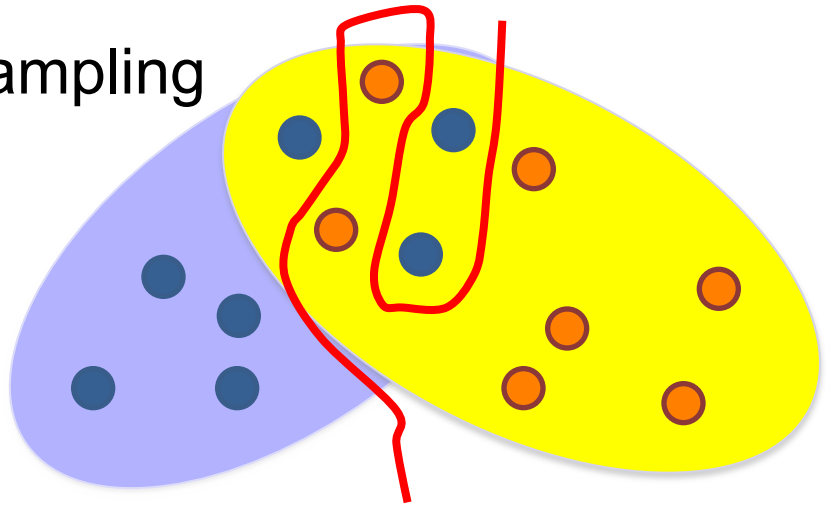
$$\max_D V(G,D) = -2\log 2 + 2\, JSD(P_{data} \,||\, P_G) \quad = 0$$

log 2 when P_{data} and P_G differ completely

Reason 1. Approximate by sampling

Weaken your discriminator?

Can weak discriminator compute JS divergence?



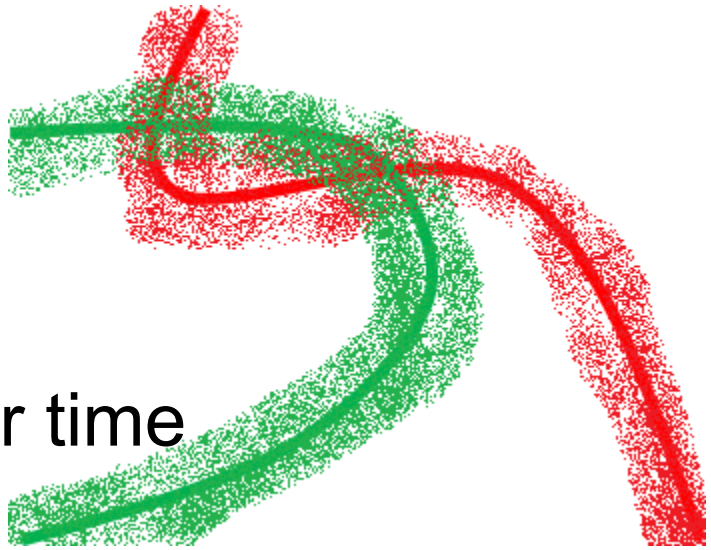
One simple solution: add noise

- 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

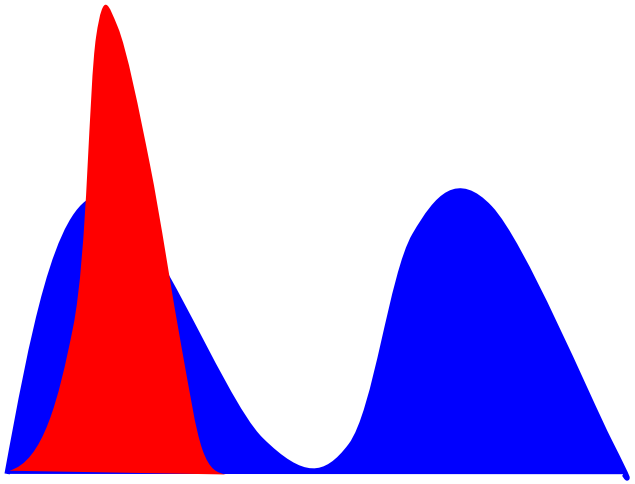
Noises need to decay over time



Mode Collapse

Converge to same faces

Generated Distribution



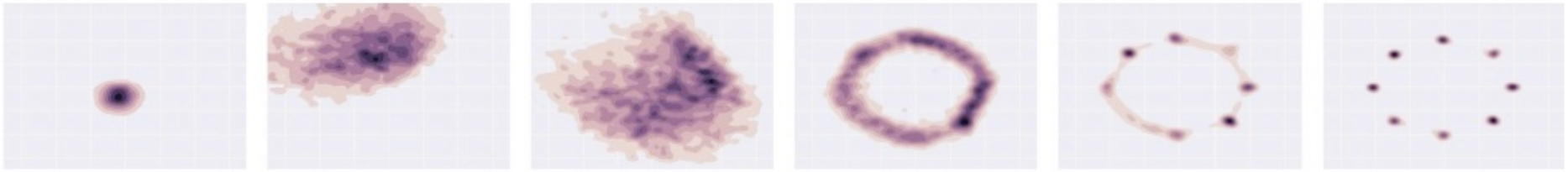
Data Distribution

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

Mode Collapse Example

8 Gaussian distributions:

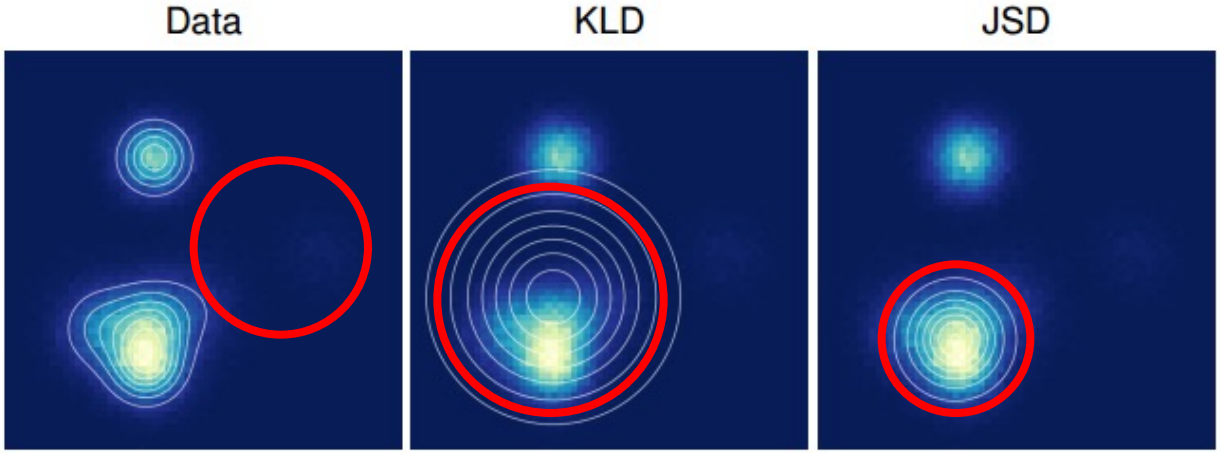
What we want …



In reality …

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**

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

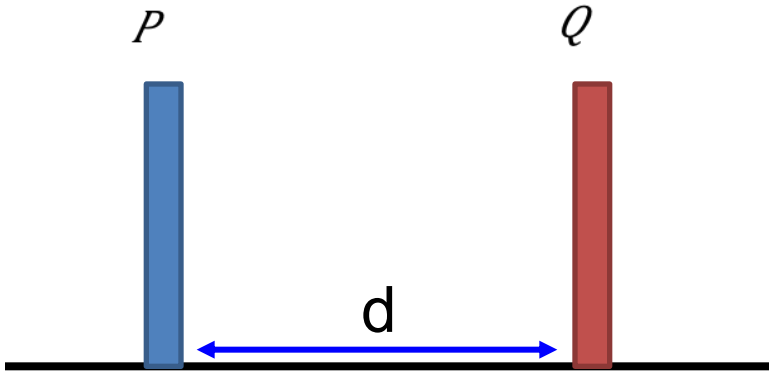
Caption	Image
this flower has white petals and a yellow stamen	
the center is yellow surrounded by wavy dark purple petals	
this flower has lots of small round pink petals	

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

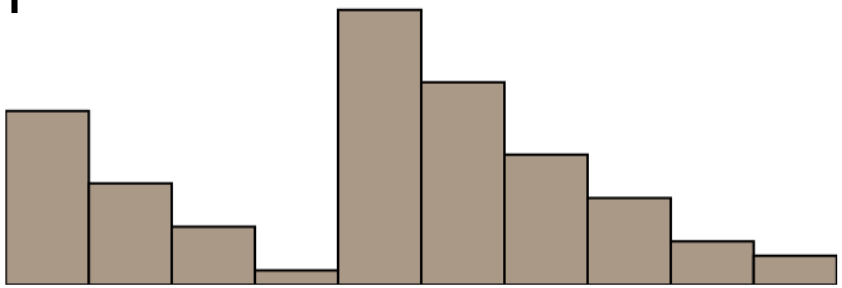
- Considering one distribution P as a pile of earth (total amount of earth is 1), and another distribution Q (another pile of earth) as the target
- The "earth mover's distance" or "Wasserstein Distance" is the average distance the earth mover has to move the earth in an optimal plan.

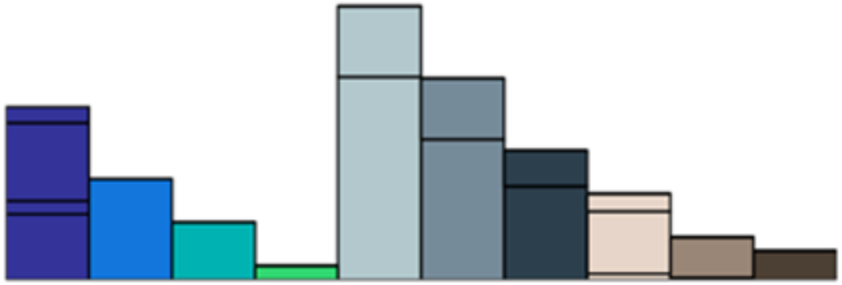


$$W(P,Q)=d$$

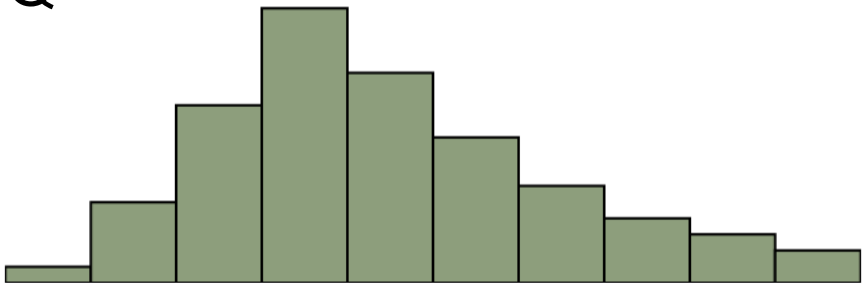
Earth Mover's Distance: best plan to move

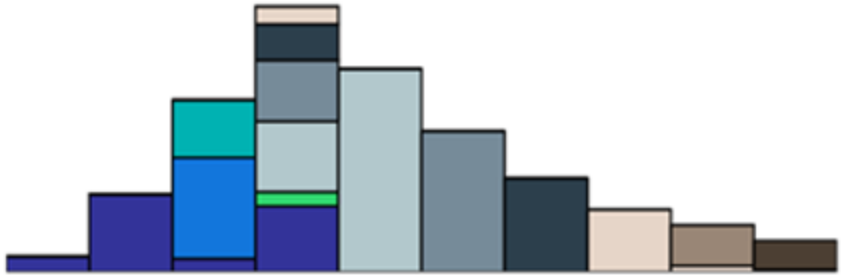
P



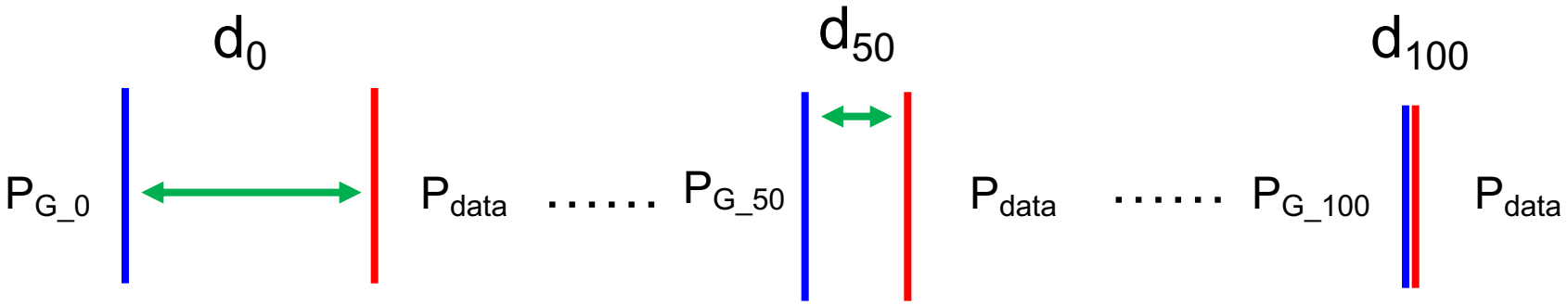


Q





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.

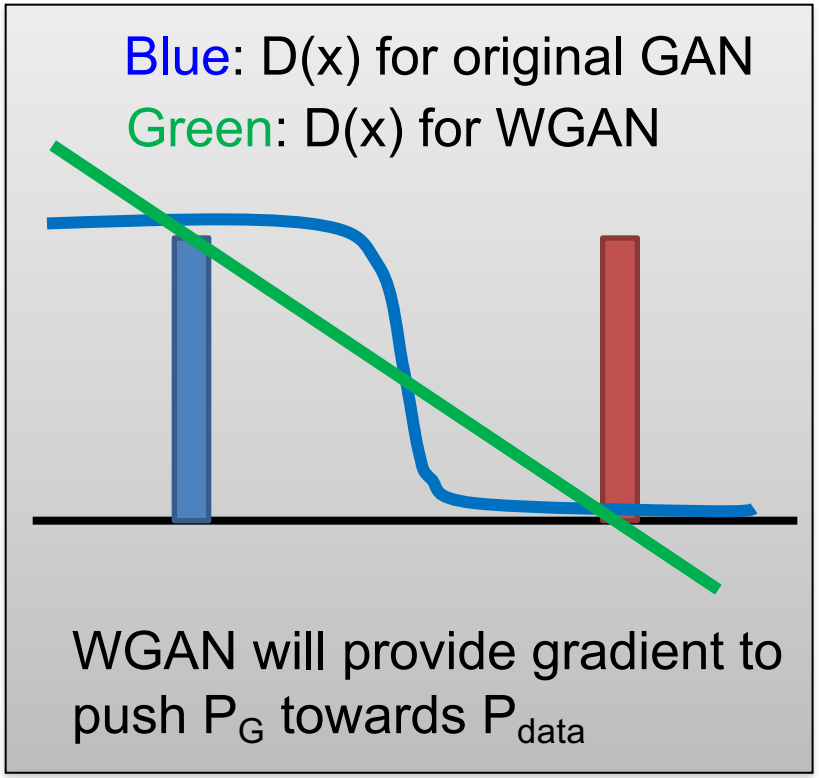
$W(P_{data}, P_G) = \max_{D \text{ is 1-Lipschitz}}[E_{x\sim P\_data} D(x) - E_{x\sim P\_G} D(x)]$

Where a function f is a k-Lipschitz function if

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

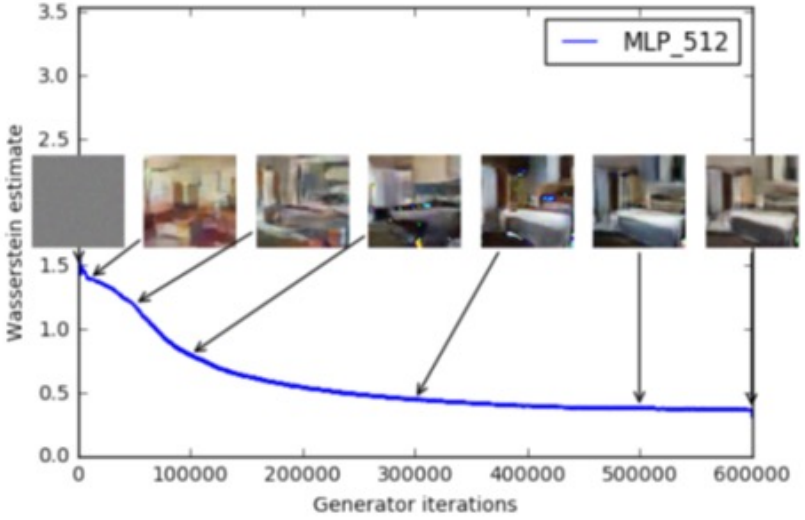
How to guarantee this?

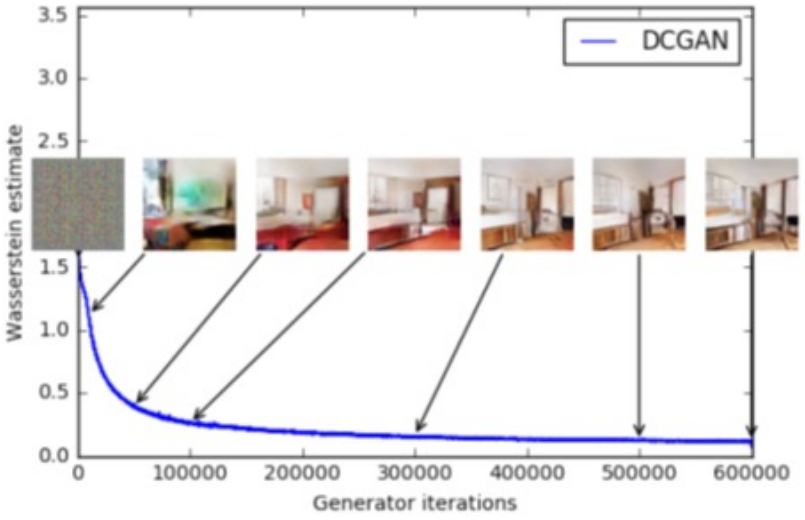
Weight clipping: for all parameter updates, if w>c Then w=c, if w<-c, then w=-c.

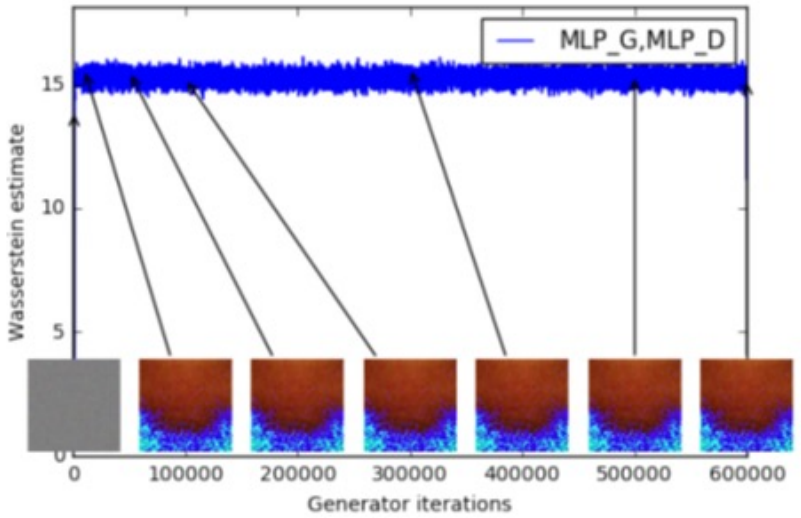


Earth Mover Distance Examples:

Multi-layer perceptron







Lab this week

- T-SNE plots on MNIST, ex.19.2
- Next week: train a simple AE on MNIST (will upload a pdf doc for the AE architecture)