Do you like to be successful?

Able to see the big picture

Are you able to recognise a

scientific GEM

How to recognise good work?
suggestions please

item#1 1st of its kind
item#2 solve problem
item#3 learning process

effort put in

value added

All these works resulted in Nobel Prizes

Description	Reason for rejection
Enrico Fermi's seminal paper on weak interaction, 1933	It contained speculations too remote from reality to be of interest to the reader
Hans Krebs' paper on the citric acid cycle, AKA the Krebs cycle, 1937	Nature publishing had a huge backlog
Murray Gell-Mann's work on classifying the elementary particles, 1953	He did not use a name editor like for his discovery
The invention of the radioimmunoassay, 1955	reviewers were skeptical that humans could make antibodies small enough to bind to things like insulin
The discovery of quasicrystals, 1984	It was rejected on the grounds that it will not interest physicists
Paper outlining nuclear magnetic resonance (NMR) spectroscopy, 1966	rejected twice by the J. Chem. Phys. to be finally accepted and published in the Review of Scientific Instruments

- ✤Is your idea so radical that your professor thinks you are out of your mind?
- ✤Do you discover nature?
- ✤Are you trying to break conventional wisdom?
- ✤If you are doing deep learning for healthcare, are you listening to good doctors?

  engineers building a tennis racket must ask the tennis players

Generative Adversarial Networks

This (GAN), . . . is the most interesting idea in the last 10 years in ML, in my opinion.

…Yann LeCun

Generative

Generative

sampled noise input

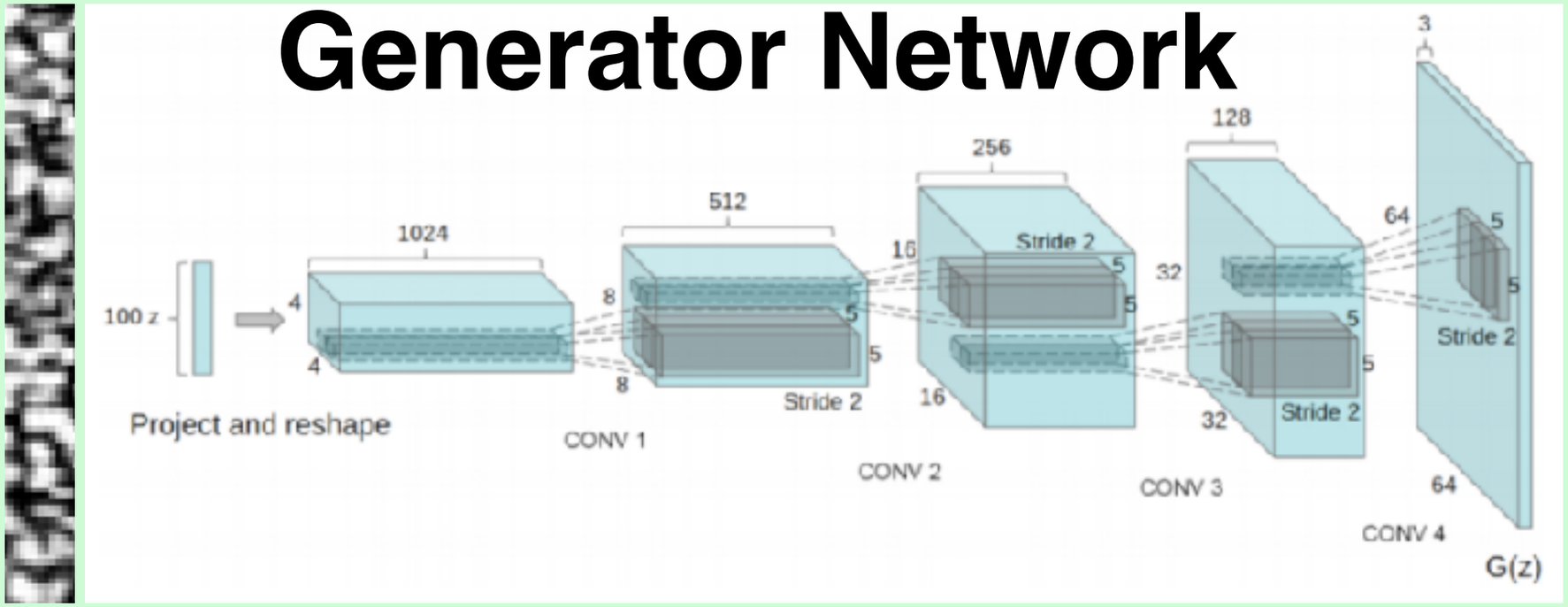
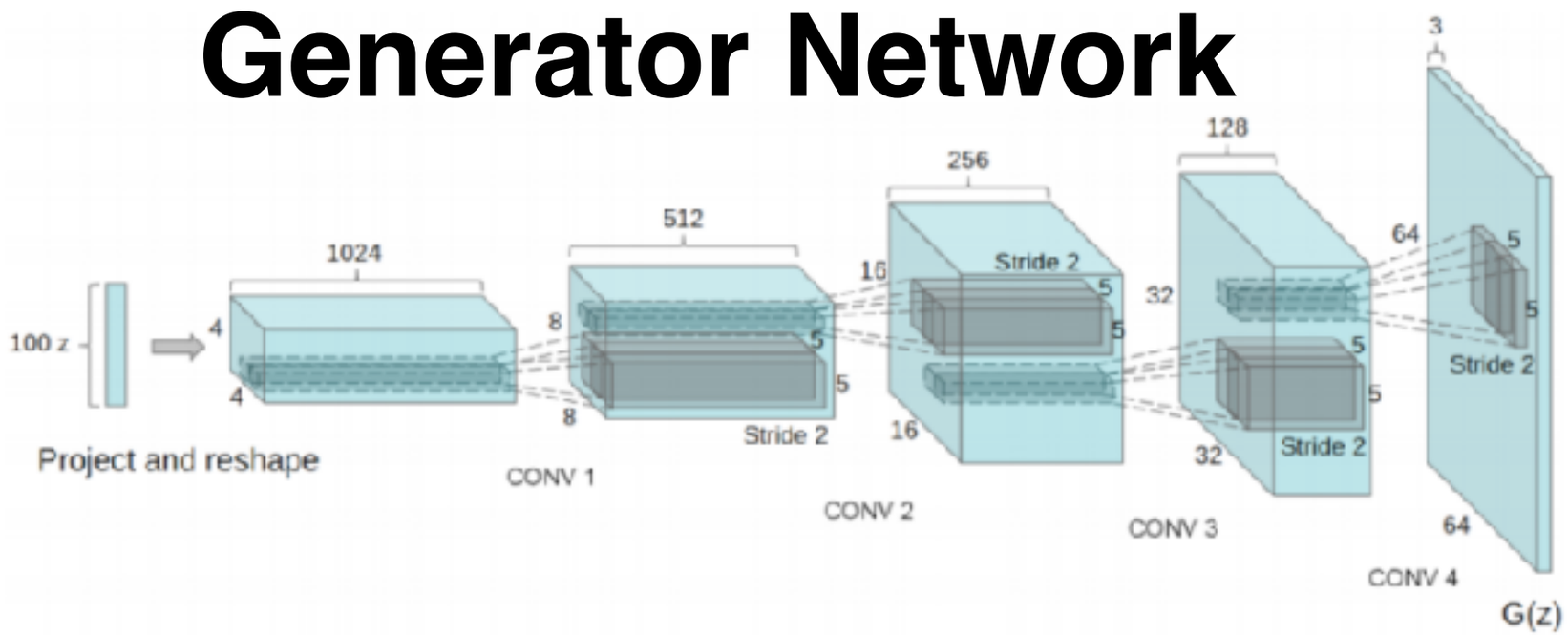


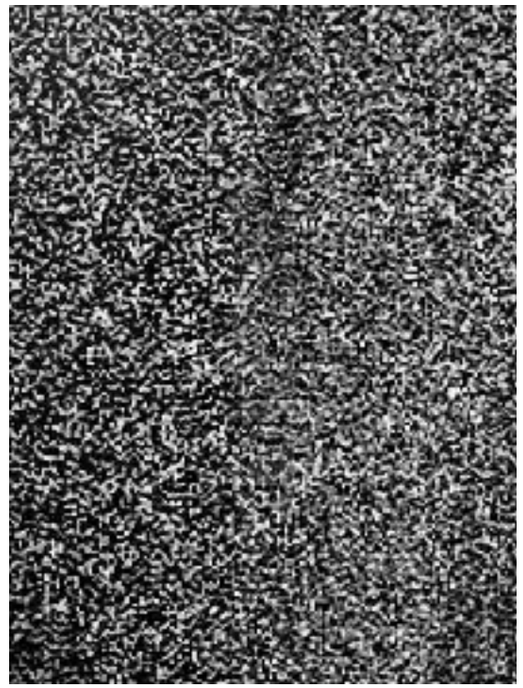
image generated

https://www.linkedin.com/pulse/unsupervised-representation-learning-deep-generative-diego

Adversarial

When the generator network is not trained well

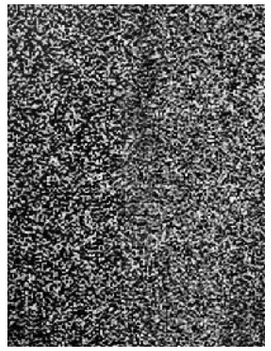


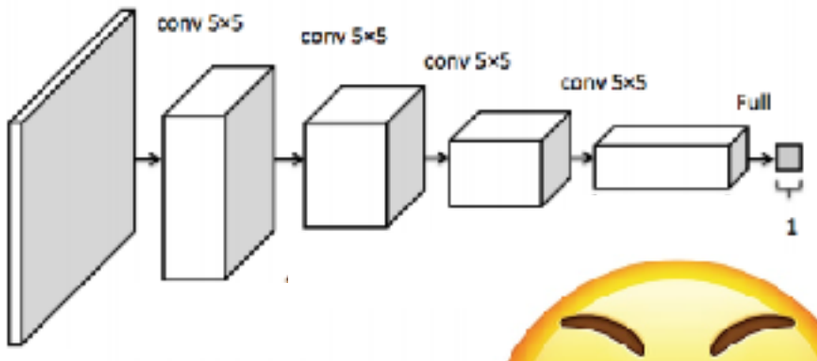


get nonsense instead of a nice picture

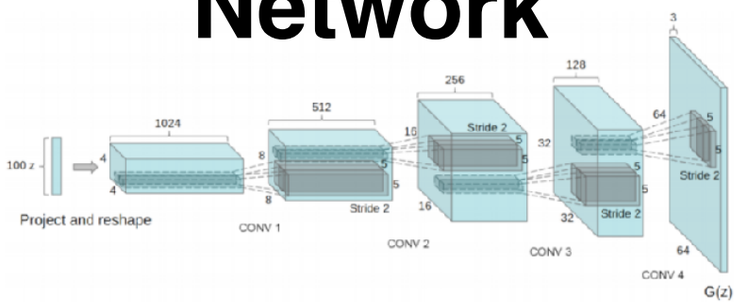
Training the generator via an **adversarial** the discriminator network

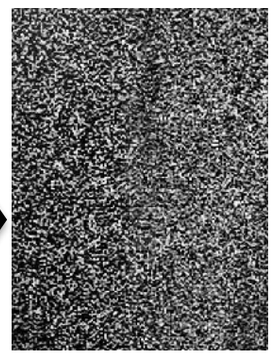
Discriminator Network





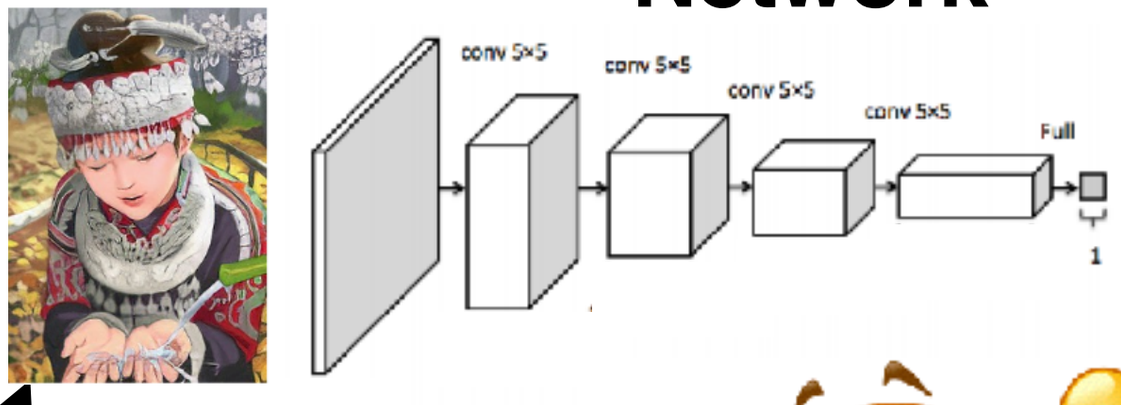
Generator Network



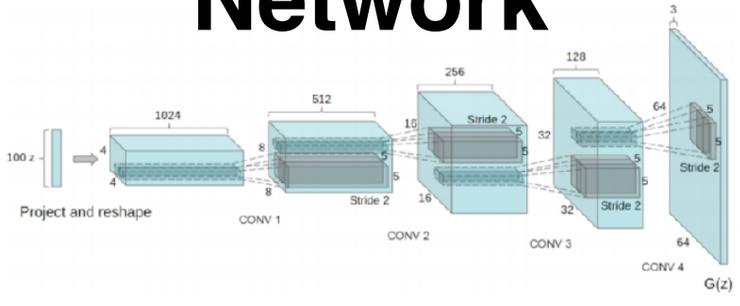


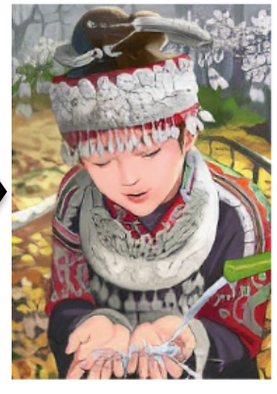
Training the generator via an **adversarial** the discriminator network

Discriminator Network

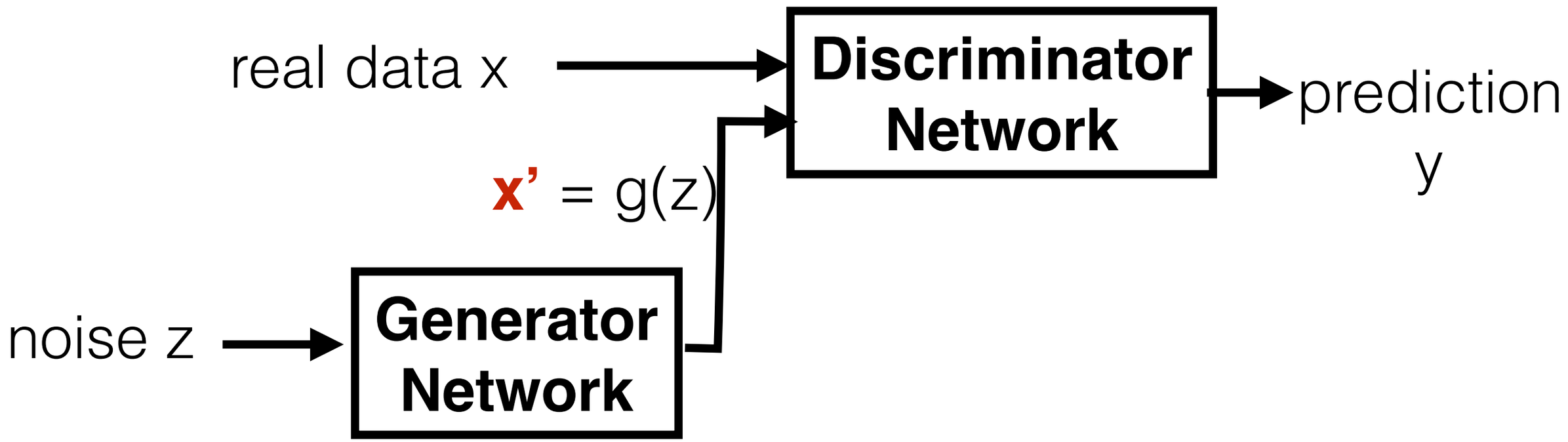


Generator Network

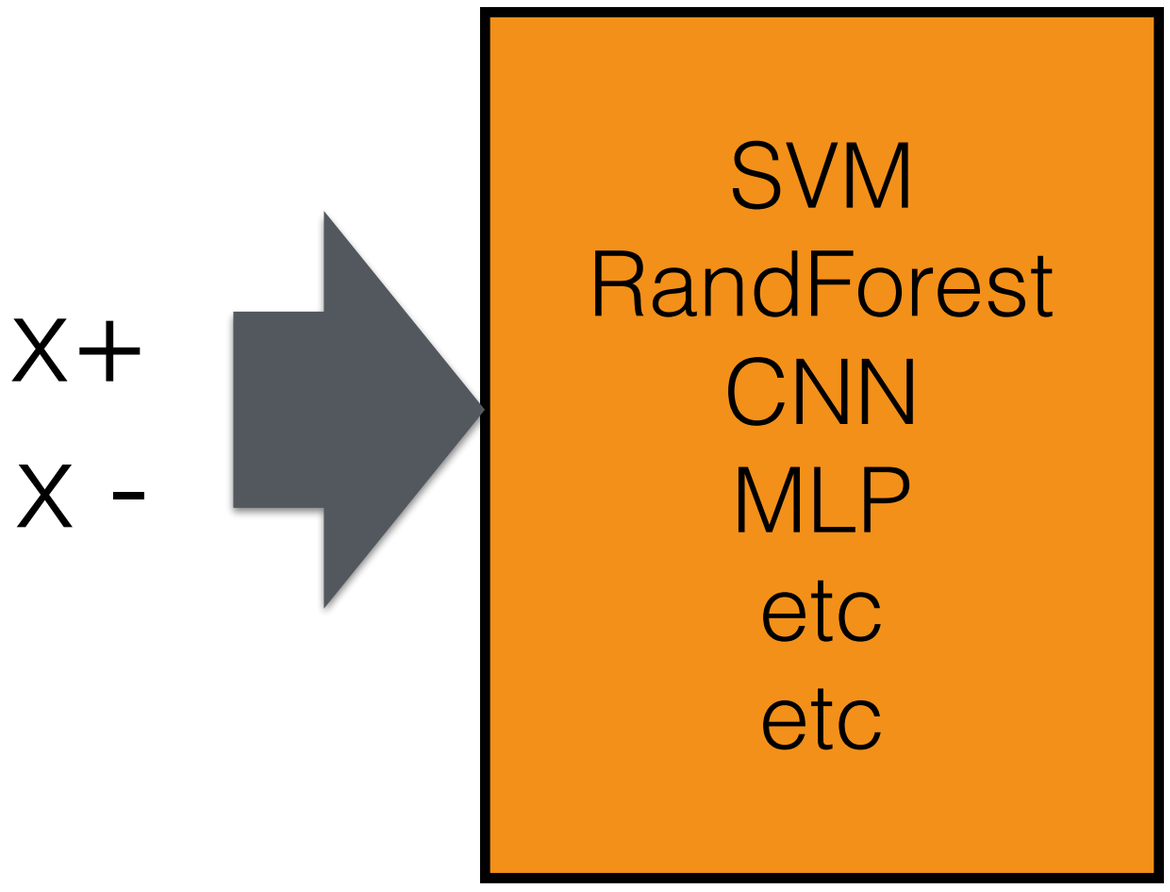




The original GAN diagram



“Classical Supervised Learning”
The passive learner



Training . . .

“Classical Supervised Learning”
The passive learner

- ✤ It ‘eats’ what you gave to it
- ✤ No data = ‘dead learner’
- ✤ Bad data = ‘bad learner’
- ✤ Has totally no control over data
- ✤ Retrain when data change a little bit

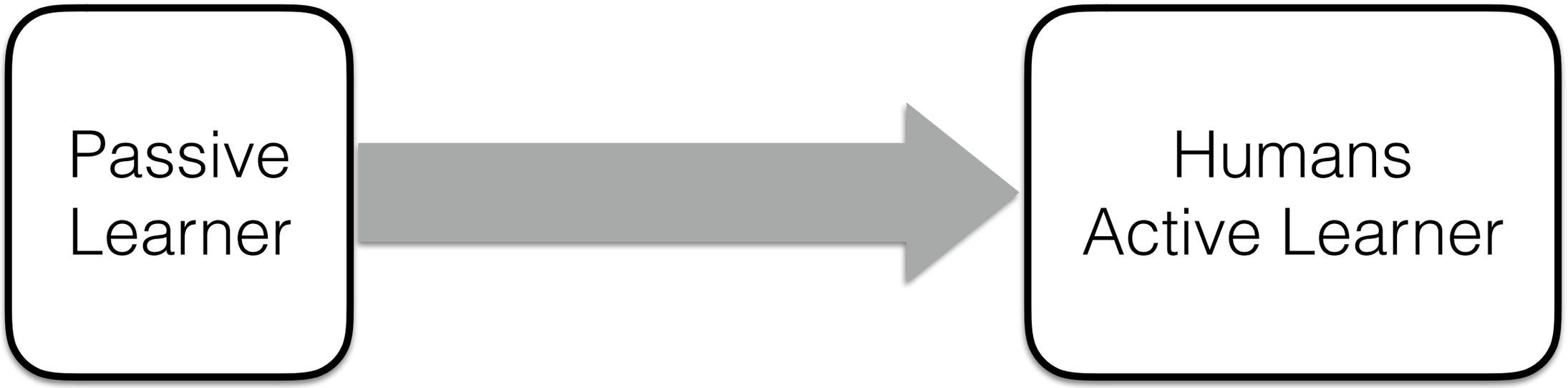
Is this an intelligent machine?

Humans

The active learner

- ✤ Select to pay attention or not pay attention
- ✤ Creative and perform thought experiments
- ✤ Generalises well, just little bit of data is ok
- ✤ We perform experiments to generate data

Learn by analogy :: 举一反三



where is GAN?

❌ Select to pay attention or not pay attention

❌ Creative and perform thought experiments

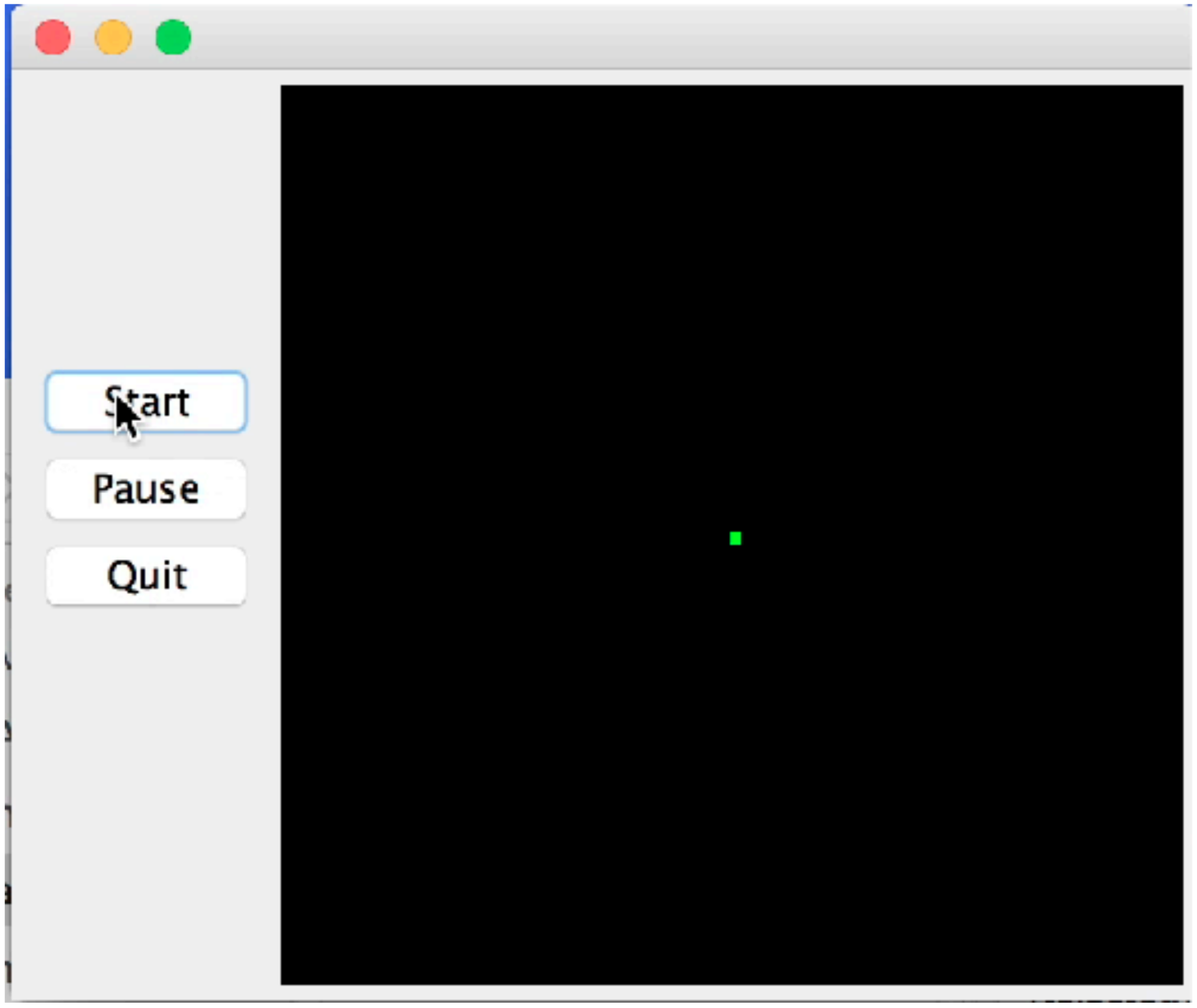
❓ Generalises well, just little bit of data is ok

✅ Performs experiments to generate data

Generative models and their problems

All generative models boils down to sampling data from a distribution

$$(x_1, x_2) \sim a_1 N(\mu_1, \sigma_1) + a_2 N(\mu_2, \sigma_2)$$



Sampling black / white pixels

$$(x_1, x_2, \cdots x_N) \sim \frac{\exp(-E(x_1, \cdots x_N)/T)}{Z} \qquad x_i = \pm 1$$

$$Z = \sum_{x_1, x_2, \cdots x_N}^{2^N} \exp(-E(x_1, \cdots x_N)/T)$$

$$E(x_1, \cdots x_N) = \sum \text{number of neighboring } x_i \text{ with the same sign}$$

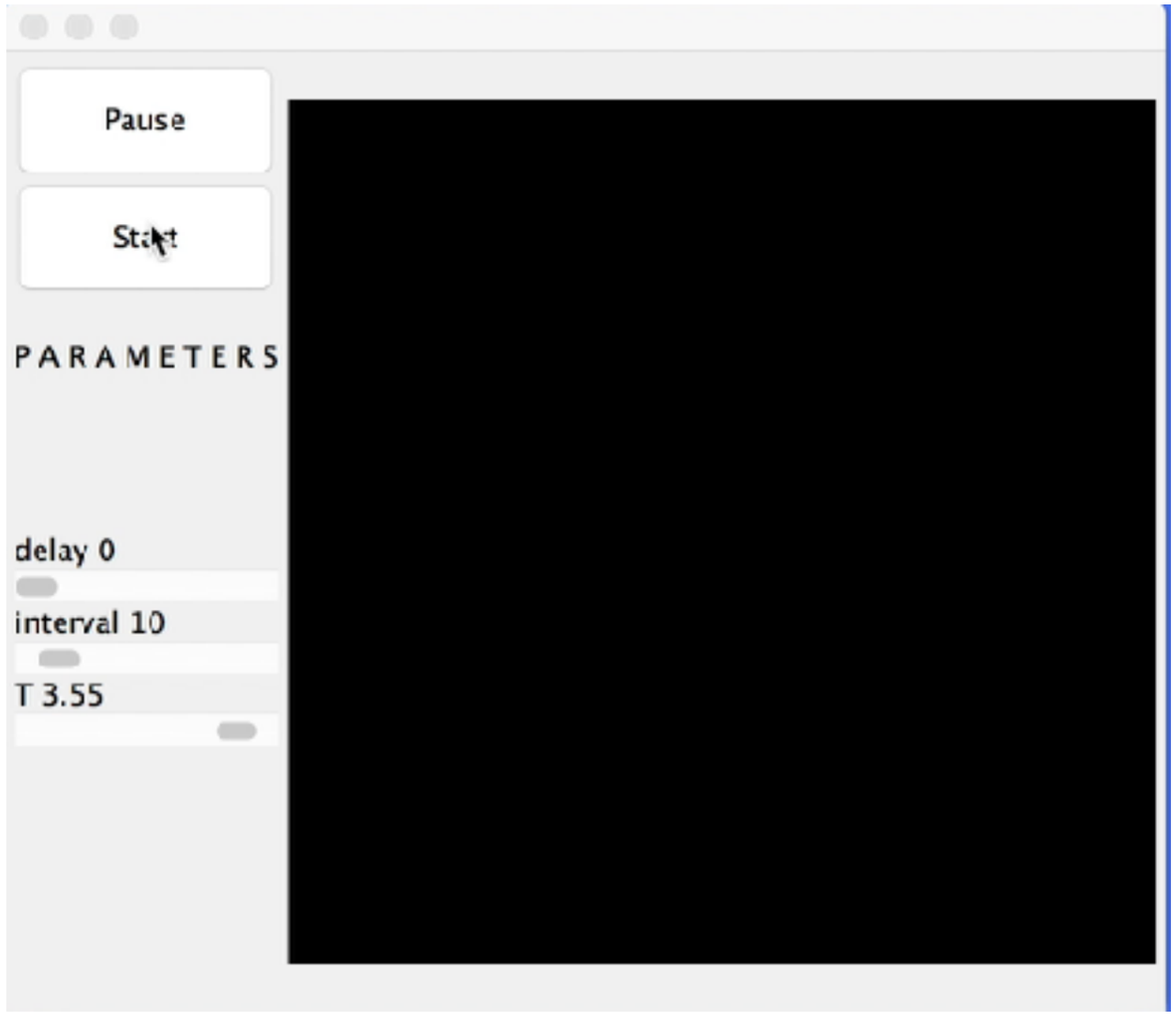
x1	*x6*	*x11*	*x16*	*x21*	*x26*
x2	*x7*	*x12*	*x17*	*x22*	*x27*
x3	*x8*	*x13*	*x18*	*x23*	*x28*
x4	*x9*	*x14*	*x19*	*x24*	*x29*
x5	*x10*	*x15*	*x20*	*x25*	*x30*

Sampling black / white pixels

$$(x_1, x_2, \cdots x_N) \sim \frac{\exp(E(x_1, \cdots x_N)/T)}{Z} \qquad x_i = \pm 1$$

$$Z = \sum_{x_1, x_2, \cdots x_N}^{2^N} \exp(E(x_1, \cdots x_N)/T)$$

$$E(x_1, \cdots x_N) = \sum \text{number of neighboring } x_i \text{ with the same sign}$$



How to sample is the key problem in generative models

This one is easy

$$(x_1, x_2) \sim a_1 N(\mu_1, \sigma_1) + a_2 N(\mu_2, \sigma_2)$$

This one is harder but can do with advanced methods

$$(x_1, x_2, \cdots x_N) \sim \frac{\exp(-E(x_1, \cdots x_N)/T)}{Z}$$

In general if we can build a distribution, theoretical or empirical distribution will do, we can find a way to sample

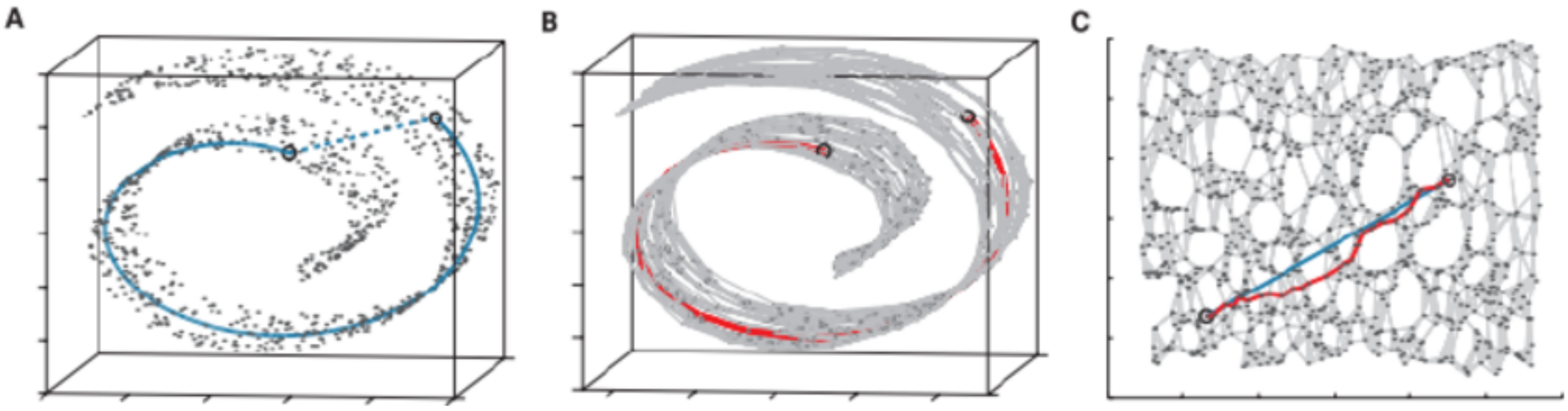
The difficulty is building a distribution
We don’t have the slightest idea how the distribution looks like

How can a distribution be build in a high dimension space where data is very sparse?

e.g. images with one mega pixels of greyscale [0,255], is embedded in a volume 255^{1,000,000}

Image data are certainly always very very sparse, it is believed that they lie in some very complex manifold.

Many real world data too occupy infinitesimal volume of feature space and form very complex manifold.



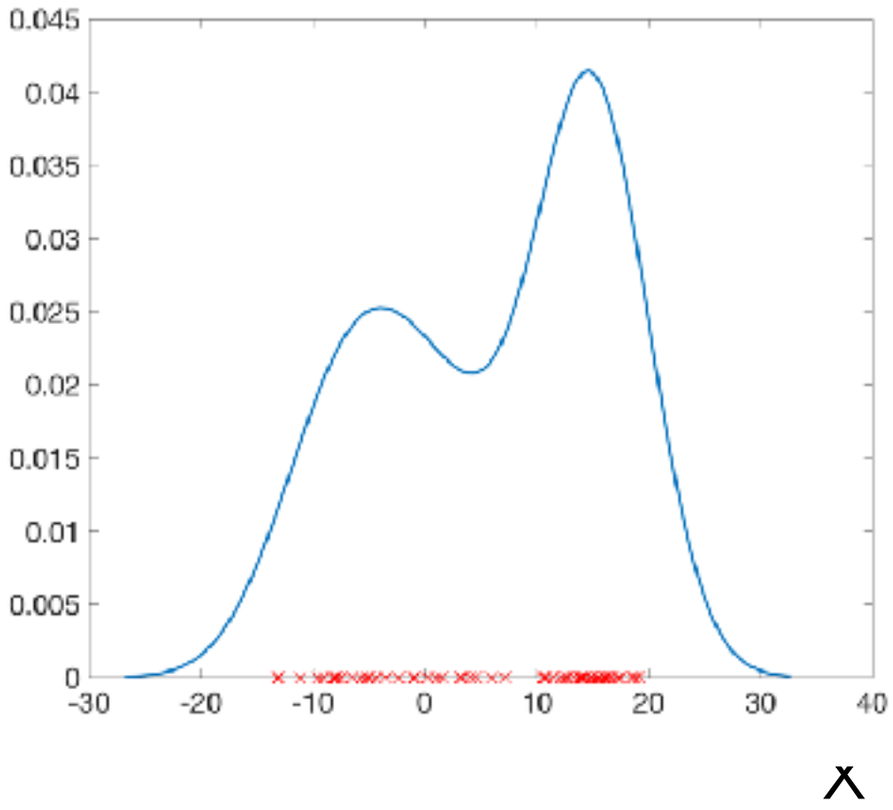
https://www.researchgate.net/figure/286513346_fig1_Figure-10-The-swiss-roll-dataset-a-The-euclidean-distance-blue-dashed-line-of-two

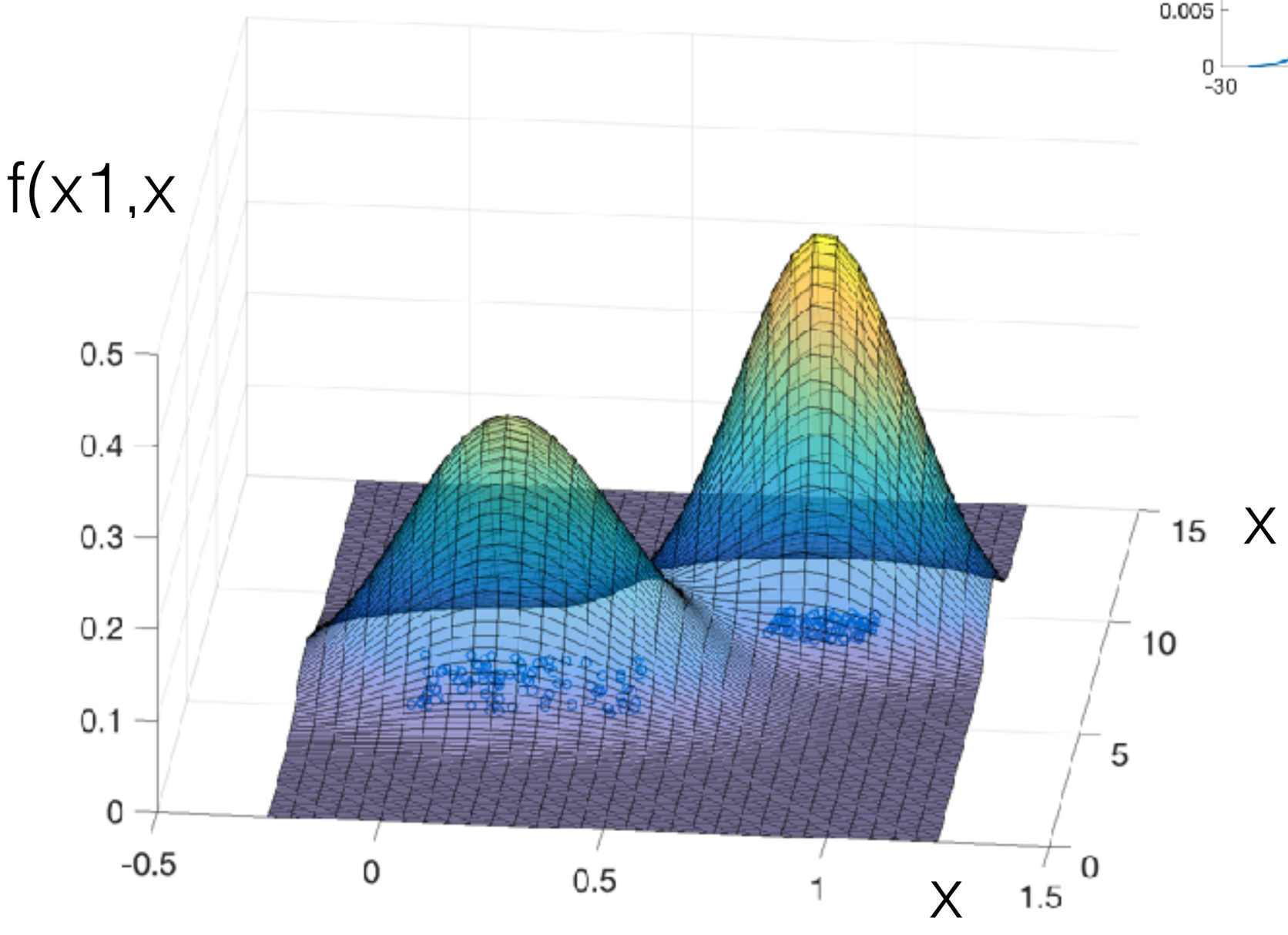
Given a data set x_i

Distribution can be constructed by finding a function $f(x)$ that is large `near' data points and small elsewhere by minimising a cost function over $f(x)$

$$C = -\int_{x \text{ `near' } x_i} f(x')dx' + \int_{x \text{ everywhere else}} f(x')dx'$$

$$C = -\int_{x \text{ 'near' } x_i} f(x')dx' + \int_{x \text{ everywhere else}} f(x')dx'$$





Given a data set x_i

Distribution can be constructed by finding a function $f(x)$ that is large `near' data points and small elsewhere by minimising a cost function over $f(x)$

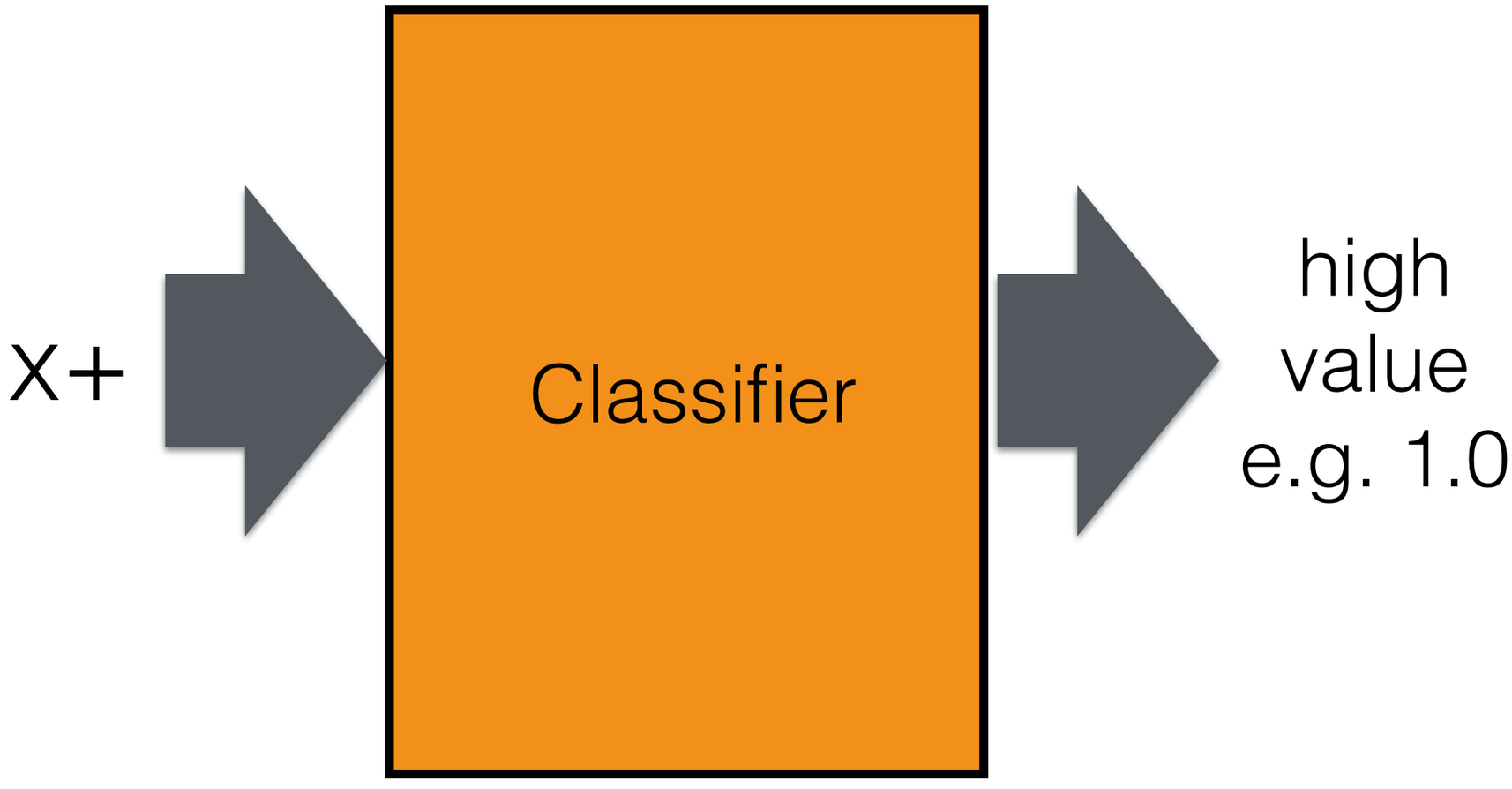
$$C = -\int_{x \text{ `near' } x_i} f(x')dx' + \int_{x \text{ everywhere else}} f(x')dx'$$

Two difficulties here:

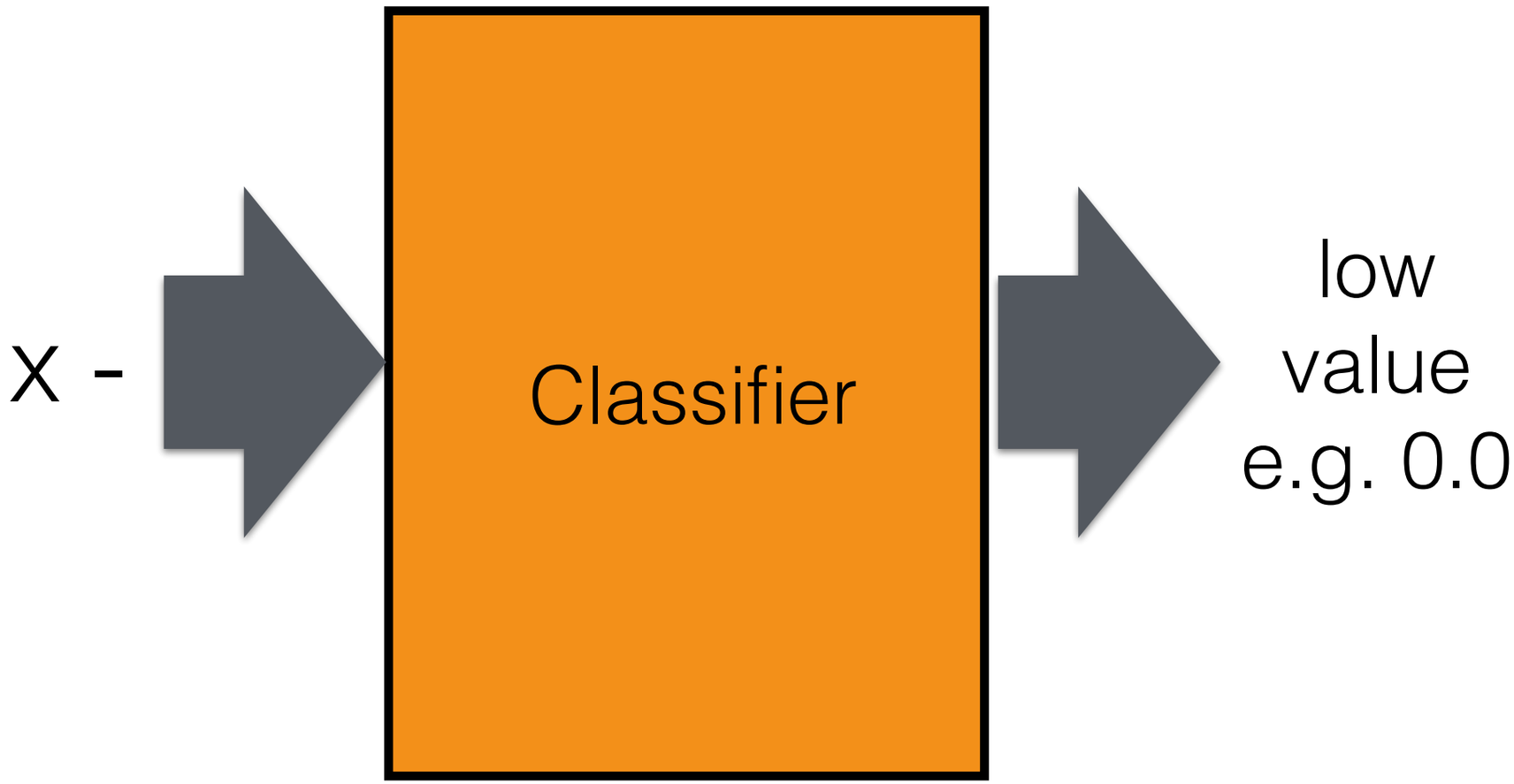
- Defining `near' - regularisation issue
- High dimensional integrals, 1,000,000 dimension for image data

Neural networks offer a pseudo-solution

Lets start by reviewing the classical supervise learning on two classes



Lets start by reviewing the classical supervise learning on two classes



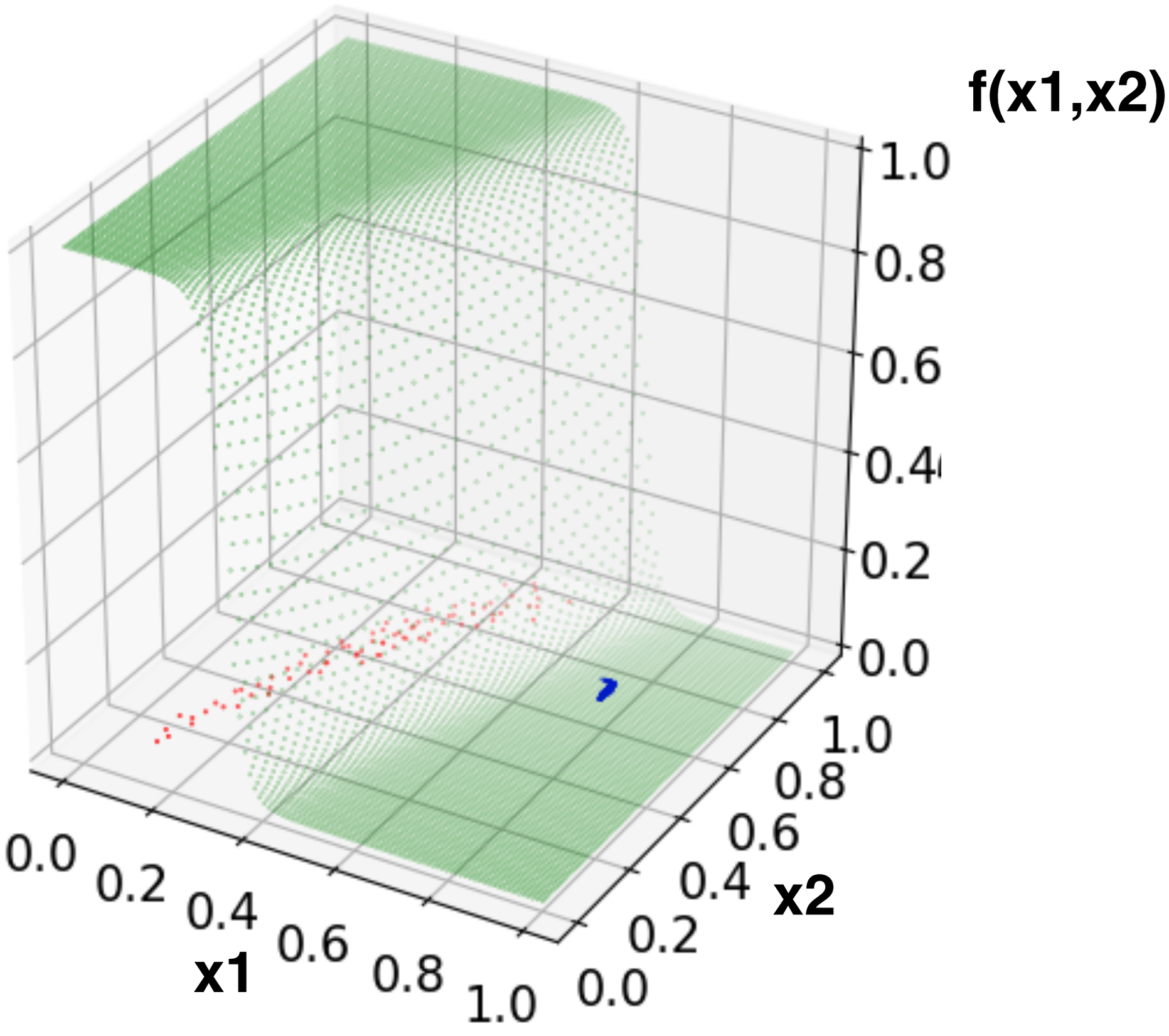
Minimize a cost function with respect to w

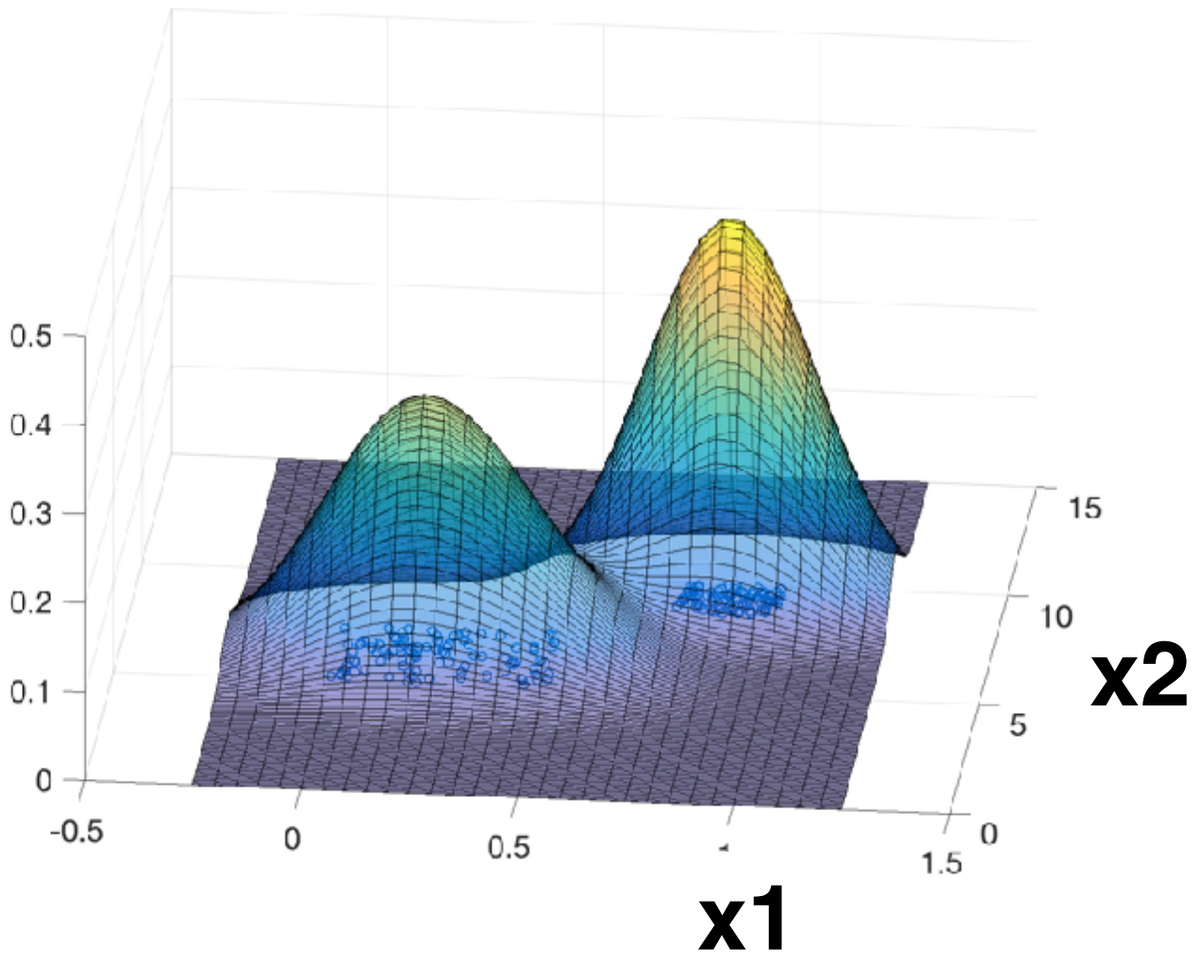
$$C(w) = -\frac{1}{N^+}\sum_i^{N^+} f_w(x_i) + \frac{1}{N^-}\sum_j^{N^-} f_w(x_j)$$

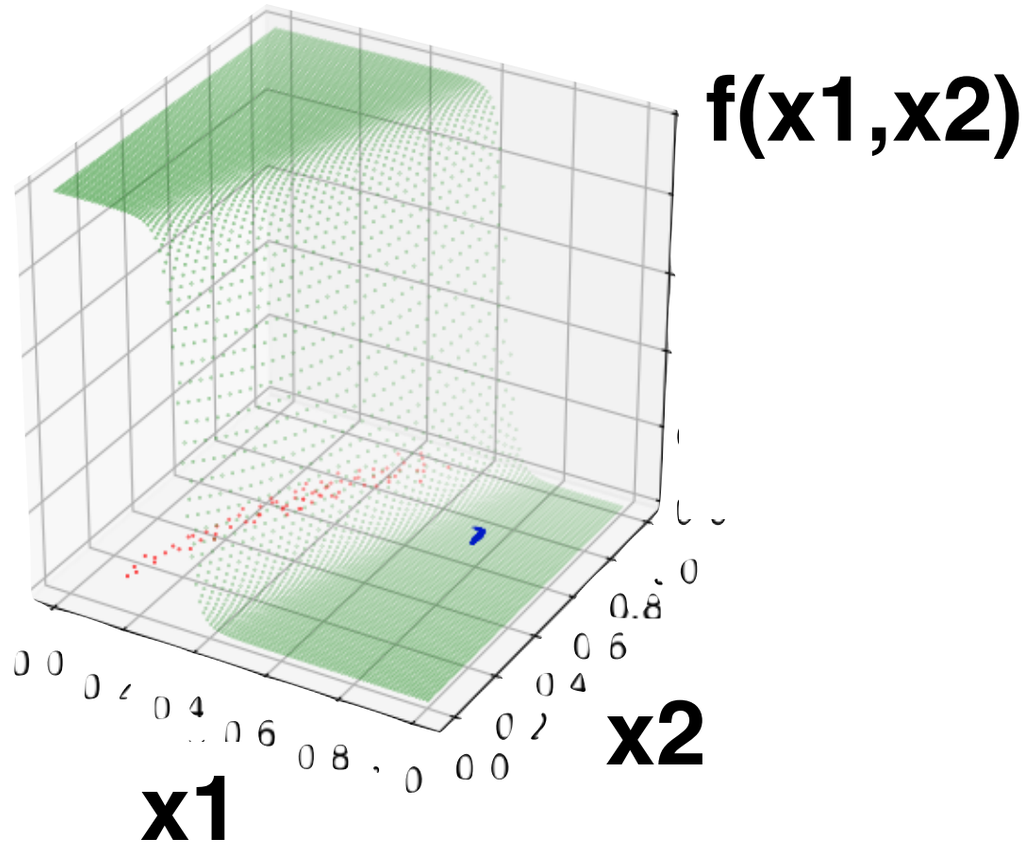
An example

$$C(w) = -\left(\frac{1}{N^+}\sum_i \log(D_w(x_i)) + \frac{1}{N^-}\sum_j \log(1 - D_w(x_j))\right)$$

$$D_w(x_i) \in (0, 1)$$


f(x1,x2)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x1
0.0
0.2
0.4
0.6
0.8
1.0
x2


0.5
0.4
0.3
0.2
0.1
0
-0.5
0
0.5
1
1.5
0
5
10
15
x2
x1


f(x1,x2)
x2
x1

Idea of GAN (Generative Adversarial Networks)

- Given a set of example real data
- Generate “fake” data that looks like read data

Generator is a neural network parametrised by θ

Sample a set of random noise

$$z_i \qquad i = 1, \cdots N^-$$

Generate set of fake data from sampled noise

$$g_\theta(z_i) \qquad i = 1, \cdots N^-$$

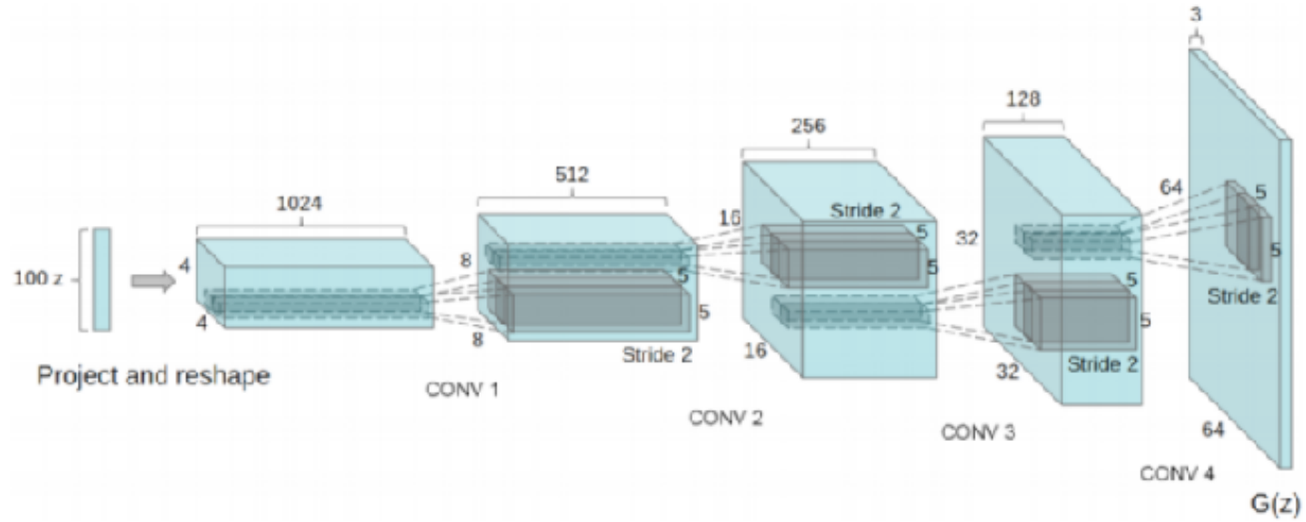
Generator Network

z_1

z_2

$\vdots$

z_{N-}



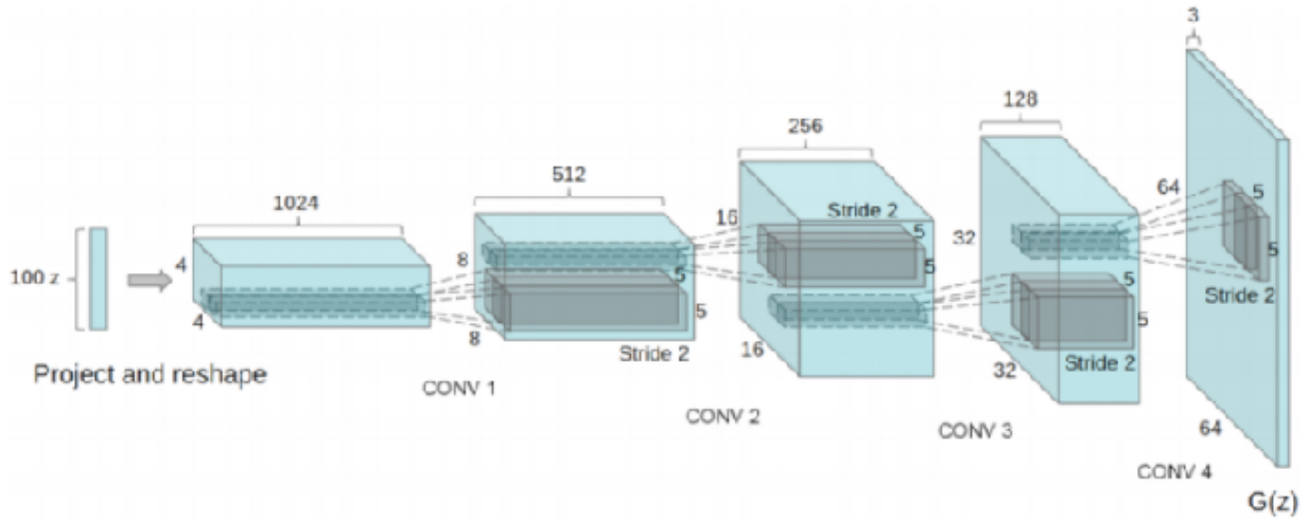
$g_\theta(z_1)$

$g_\theta(z_2)$

$\vdots$

$g_\theta(z_{N-})$

If we adjust θ well, $g_\theta(z_i)$ will look like this



Training Generator network using Discriminator network

Discriminator network needs positive and negative training examples

$$C(w) = -\frac{1}{N^+}\sum_i^{N^+} f_w(x_i) + \frac{1}{N^-}\sum_j^{N^-} f_w(x_j)$$

put real data examples here

put “fake” generated data here

First train the Discriminator network

$$C(w,\theta) = -\frac{1}{N^+}\sum_{i}^{N^+} f_w(x_i) + \frac{1}{N^-}\sum_{j}^{N^-} f_w(g_\theta(z_j))$$

put real data examples here

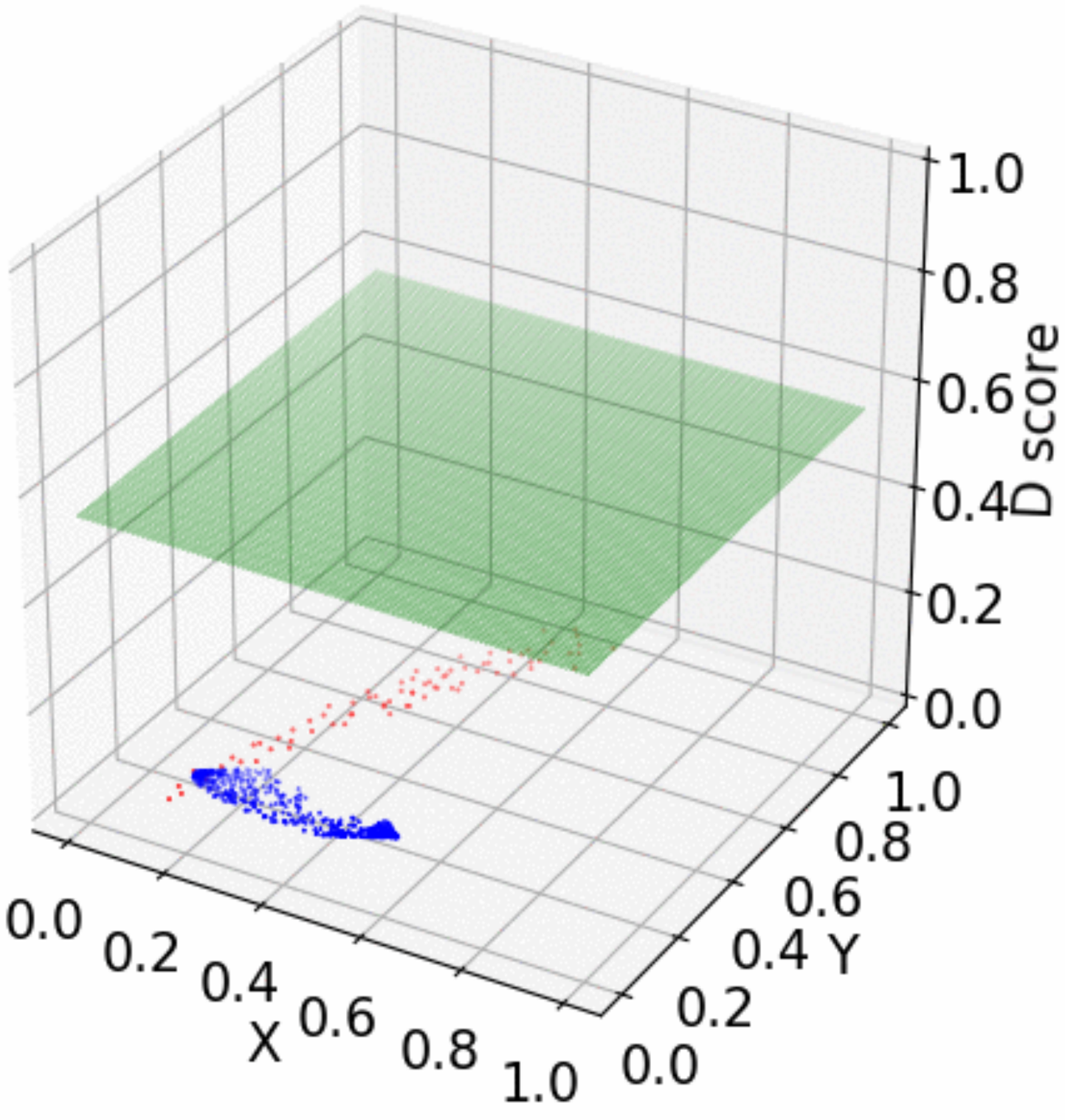
put “fake” generated data here

$$w^* = \arg\min_w C(w,\theta) \qquad \text{keeping } \theta \text{ constant}$$

Discriminator network is trained to grade generated data as negative

It criticised any imperfections of the generator

Discriminator : x -> 10 -> 10 -> 10 -> out



Next train the Generator network

$$C(w,\theta) = -\frac{1}{N^+}\sum_{i}^{N^+} f_w(x_i) + \frac{1}{N^-}\sum_{j}^{N^-} f_w(g_\theta(z_j))$$

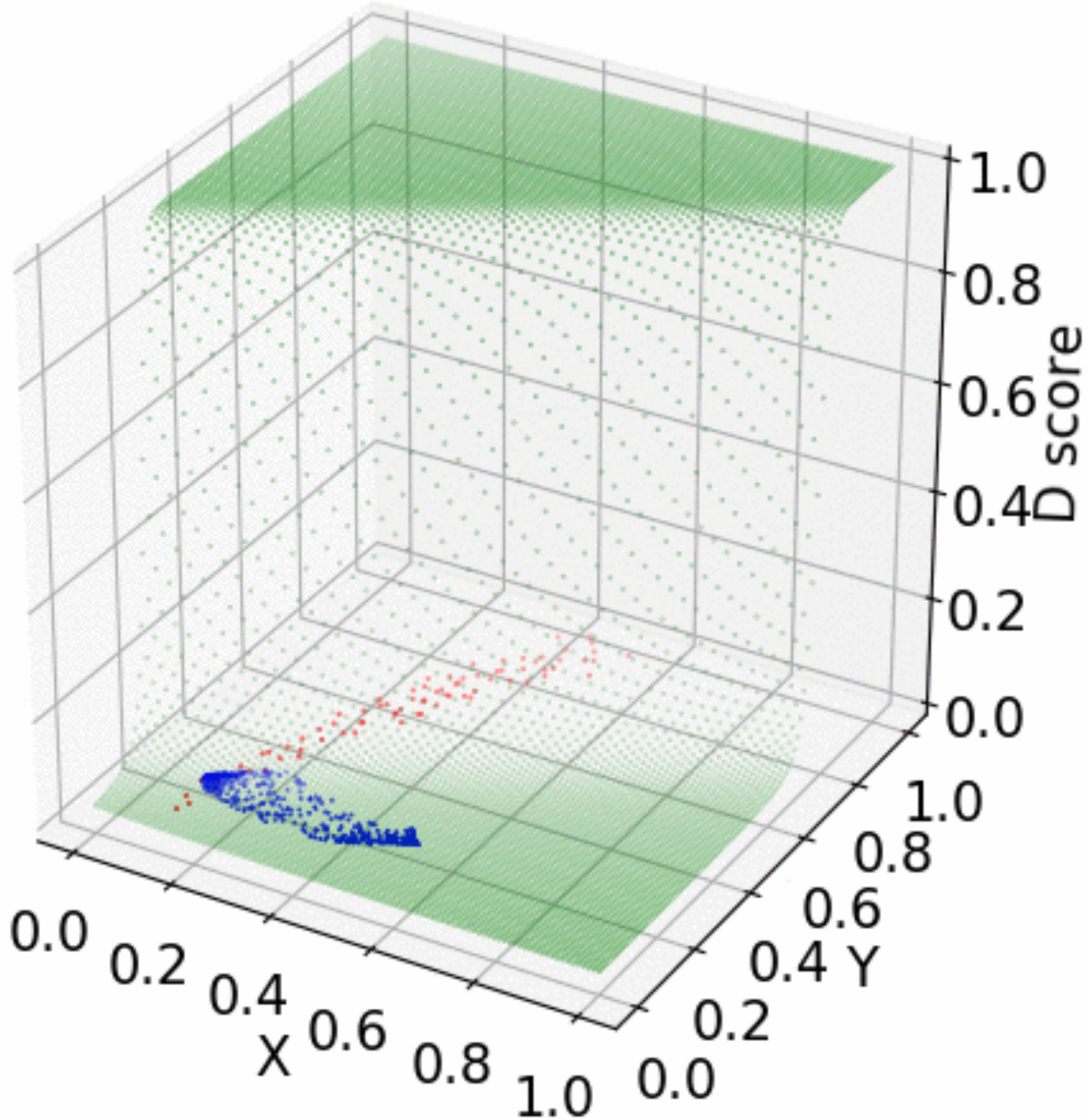
put real data examples here

put "fake" generated data here

$$w^* = \arg\min_w C(w,\theta) \qquad \text{keeping } \theta \text{ constant}$$

$$\theta^* = \arg\max_\theta C(w,\theta) \qquad \text{keeping } w \text{ constant}$$

Generator : x -> 5 -> 5 -> 5 -> out



Cost function use in original GAN paper

$$C(w,\theta) = -\left(\frac{1}{N^+}\sum_i \log(D_w(x_i)) + \frac{1}{N^-}\sum_j \log(1 - D_w(g_\theta(z_j)))\right)$$

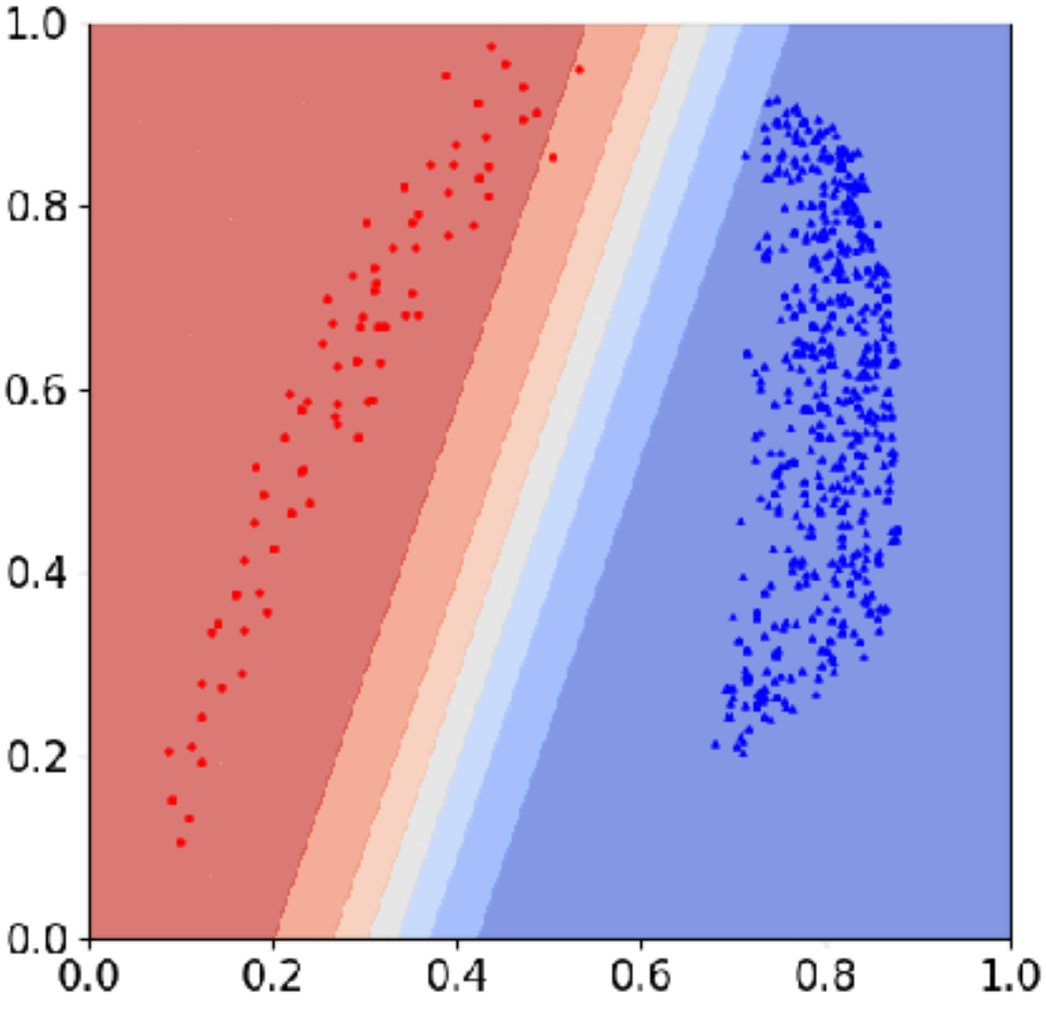
$$w^* = \arg\min_w C(w,\theta) \qquad \text{keeping } \theta \text{ constant}$$

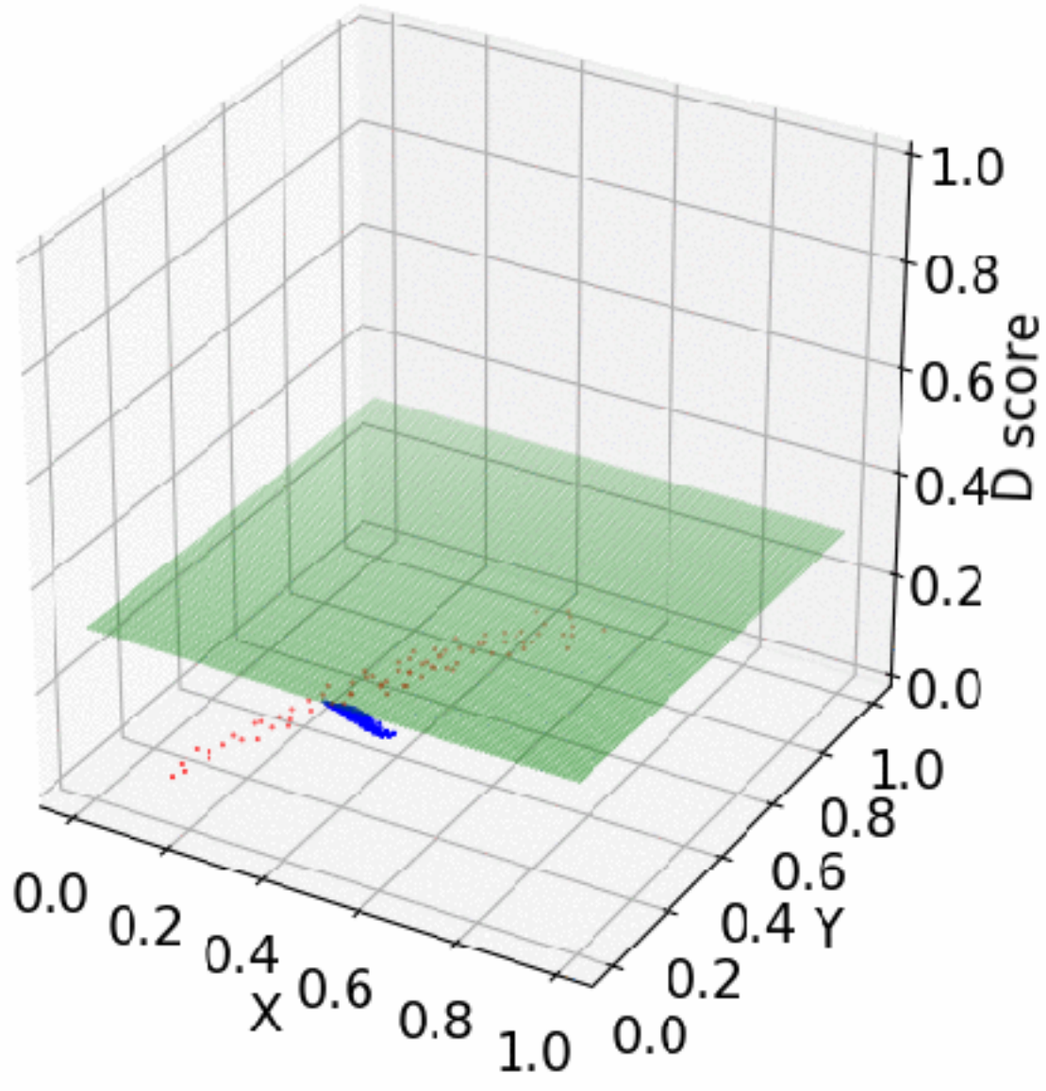
$$\theta^* = \arg\max_\theta C(w,\theta) \qquad \text{keeping } w \text{ constant}$$

1. Randomly initialise θ and w
2. Use $g_\theta(z_i)$ to generate "fake" data
3. Keep θ constant and optimise w for some n iterations
4. Keep w constant and optimise θ for some k iterations

Discriminator : x -> 10 -> 10 -> 10 -> out

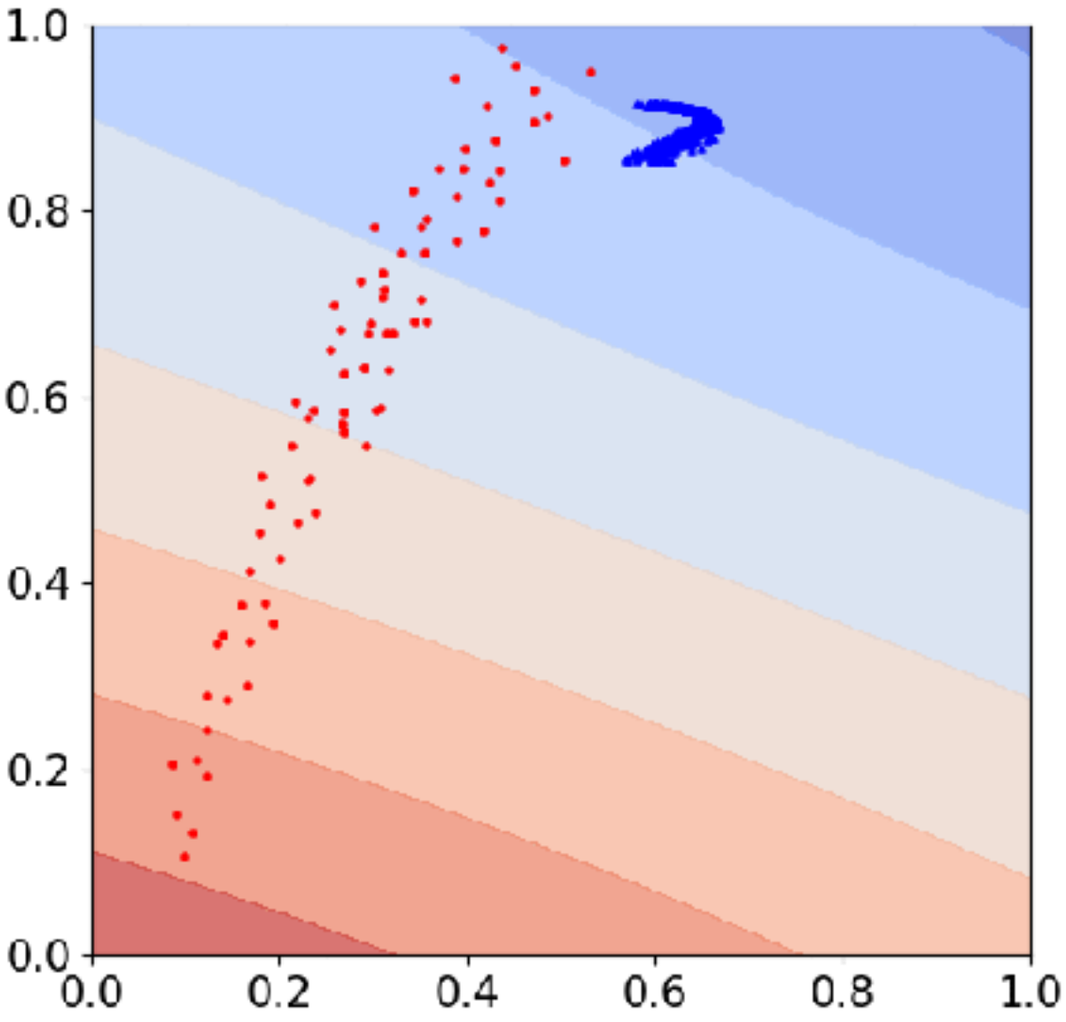
Generator : x -> 5 -> 5 -> 5 -> out

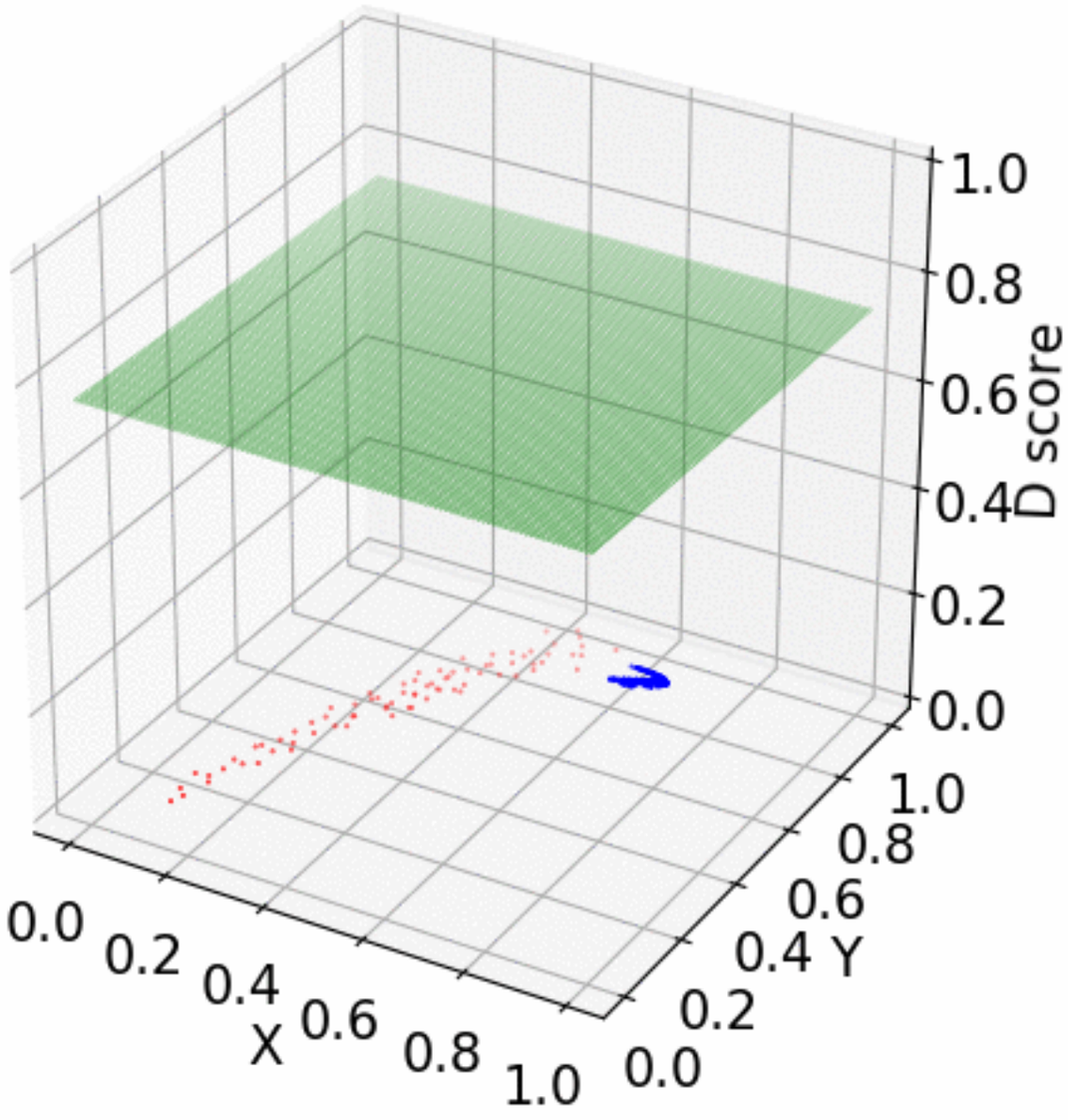




Discriminator : x -> 10 -> 10 -> 10 -> out

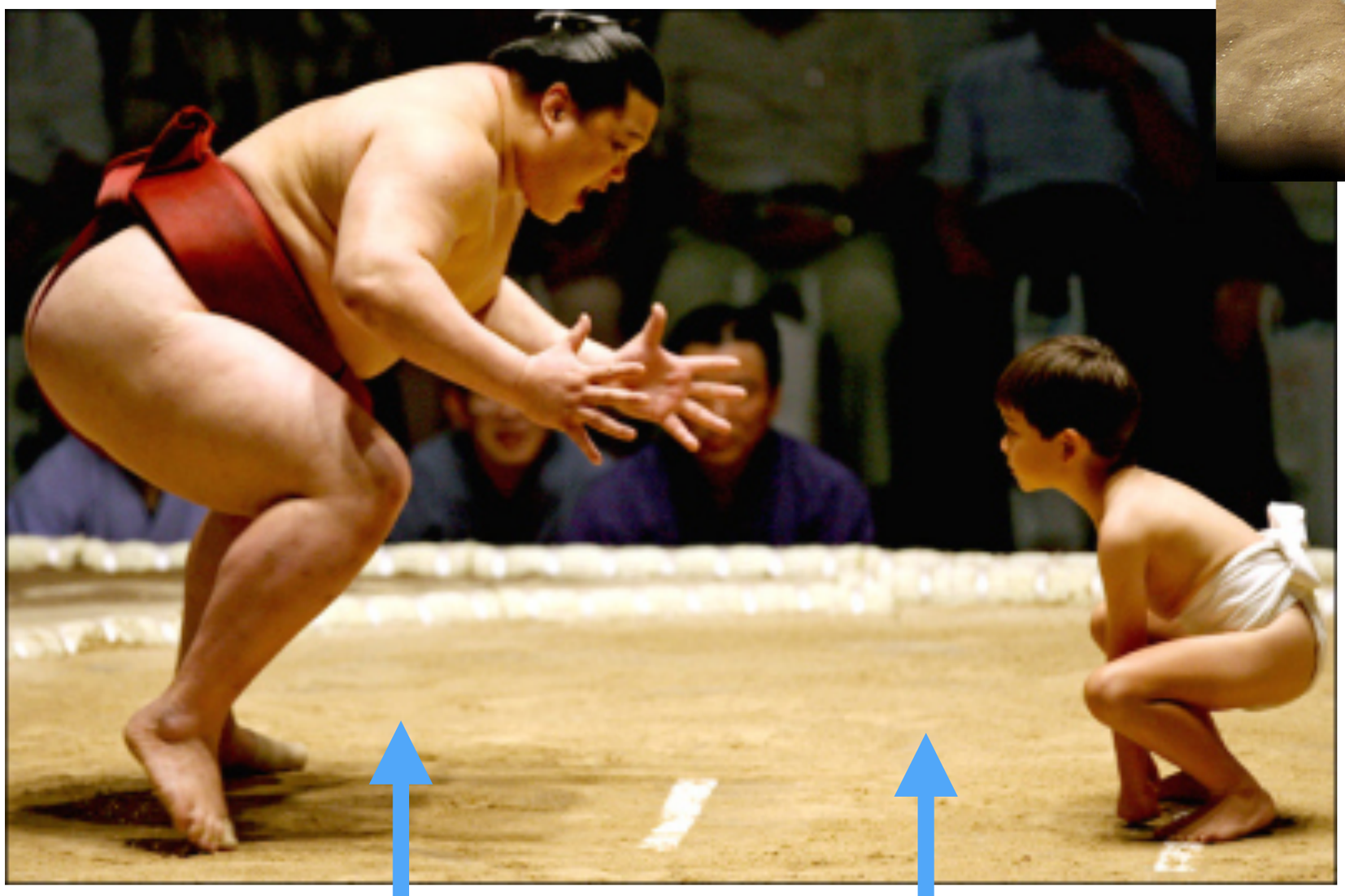
Generator : x -> 5 -> 5 -> 5 -> out





Shortcomings of GAN

When two adversarial of equal strength competes both can improve

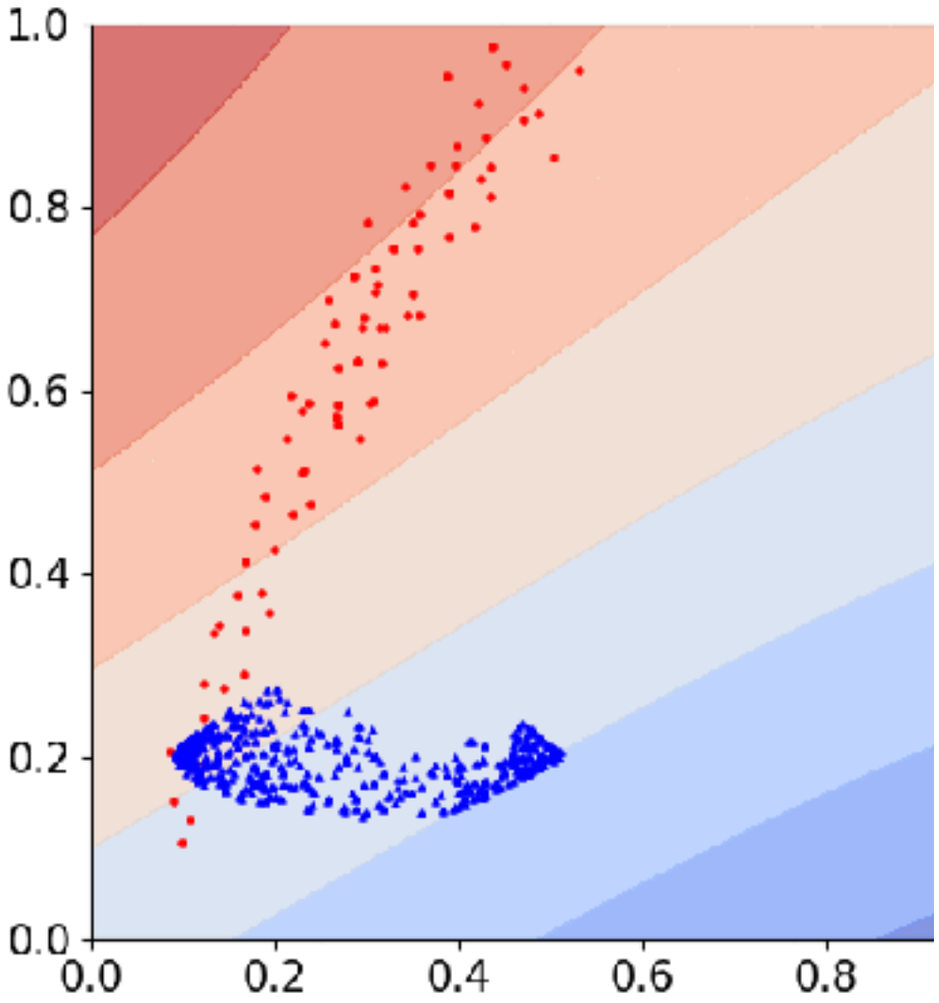


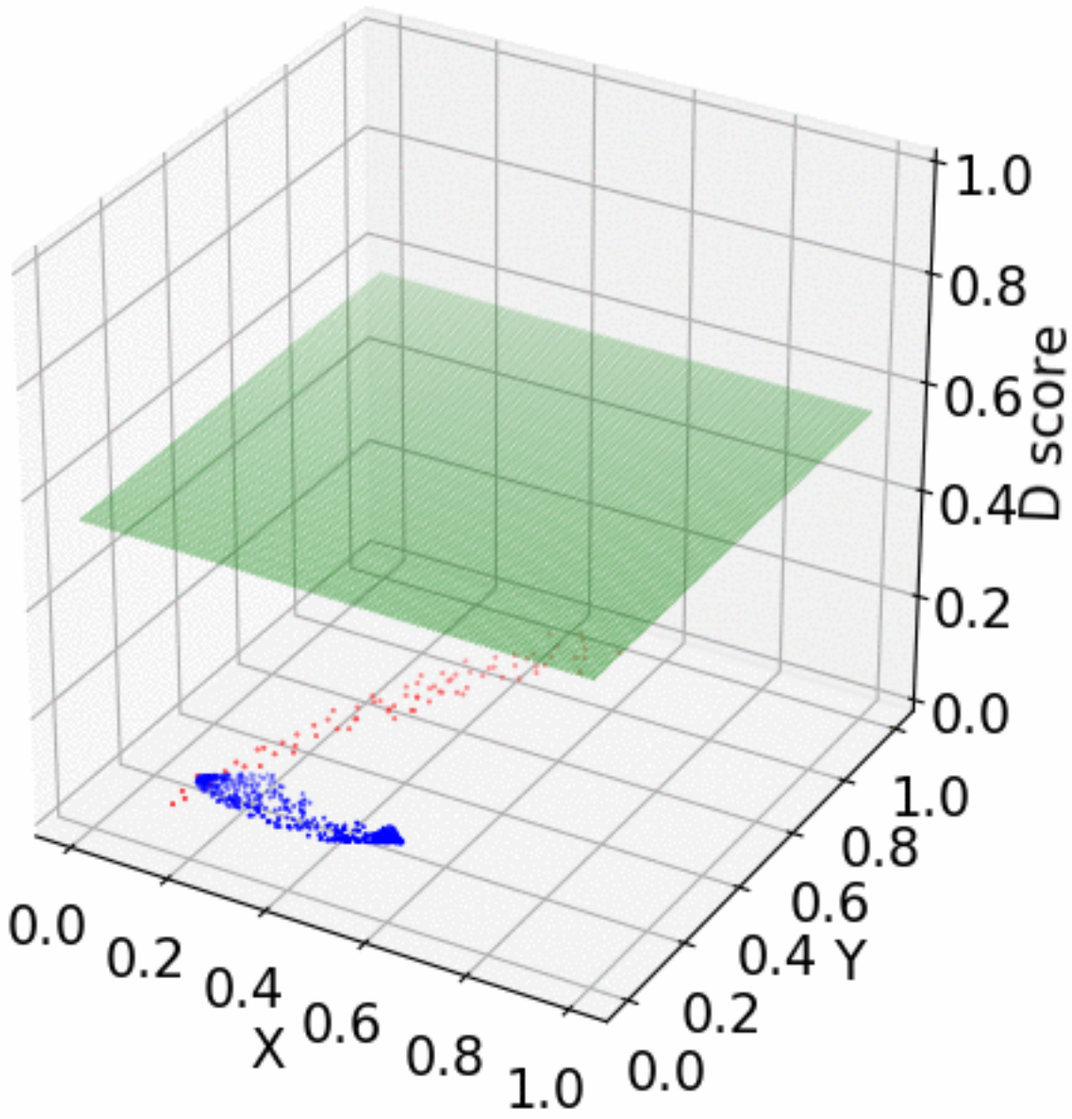
Discriminator

Generator

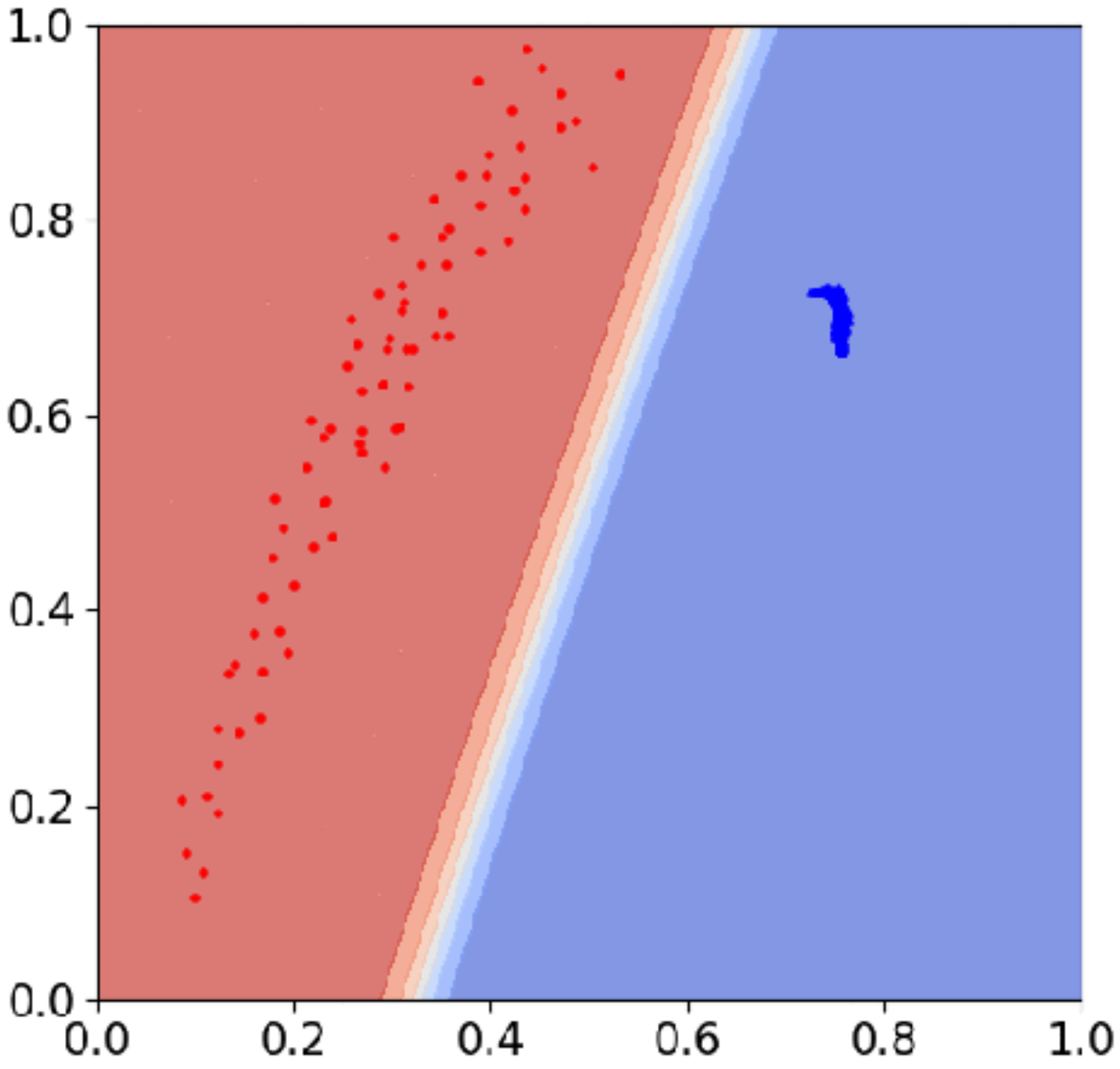
Discriminator fail

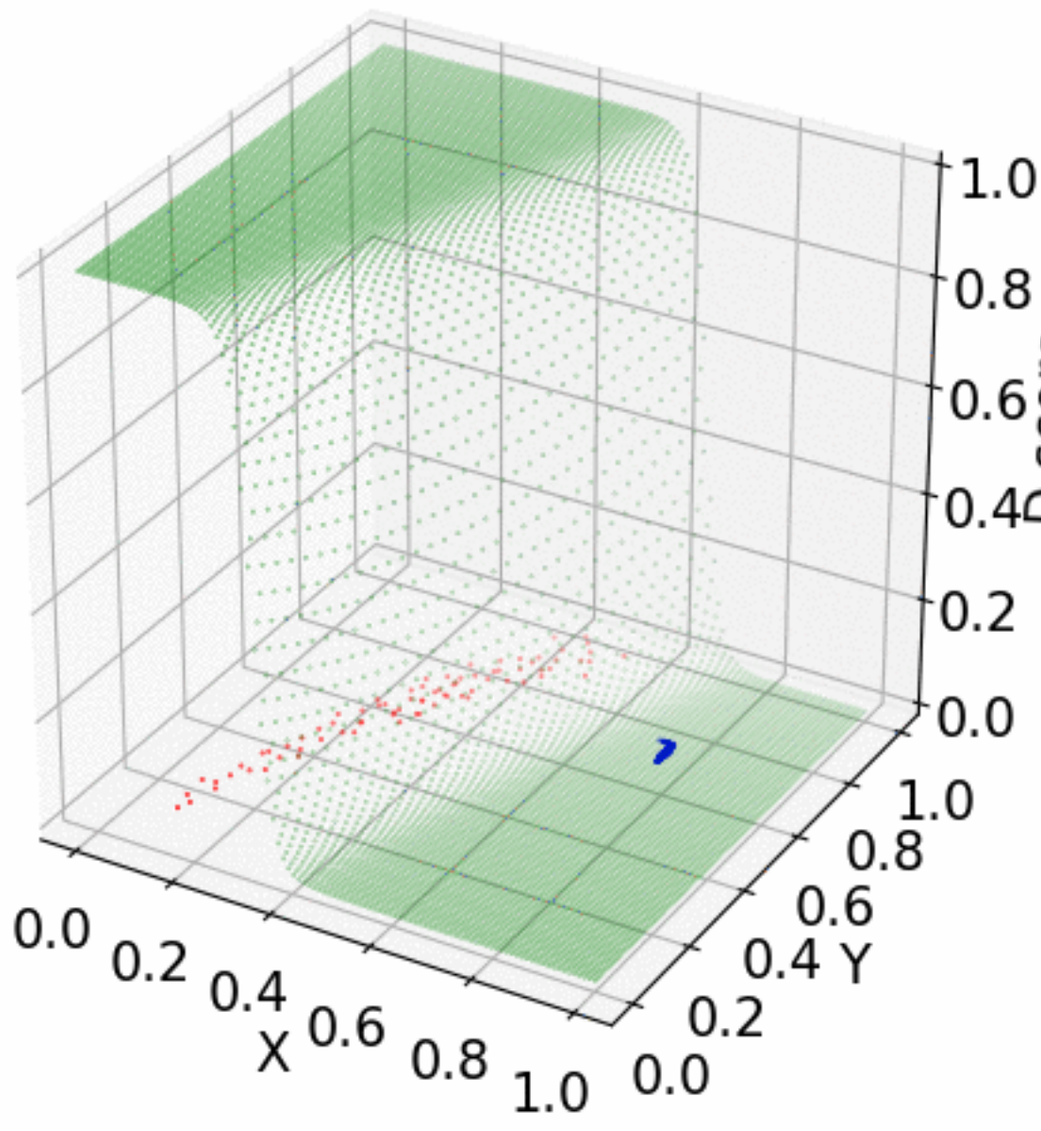
Discriminator : x -> 10 -> 10 -> 10 -> out
Generator : x -> 5 -> 5 -> 5 -> out



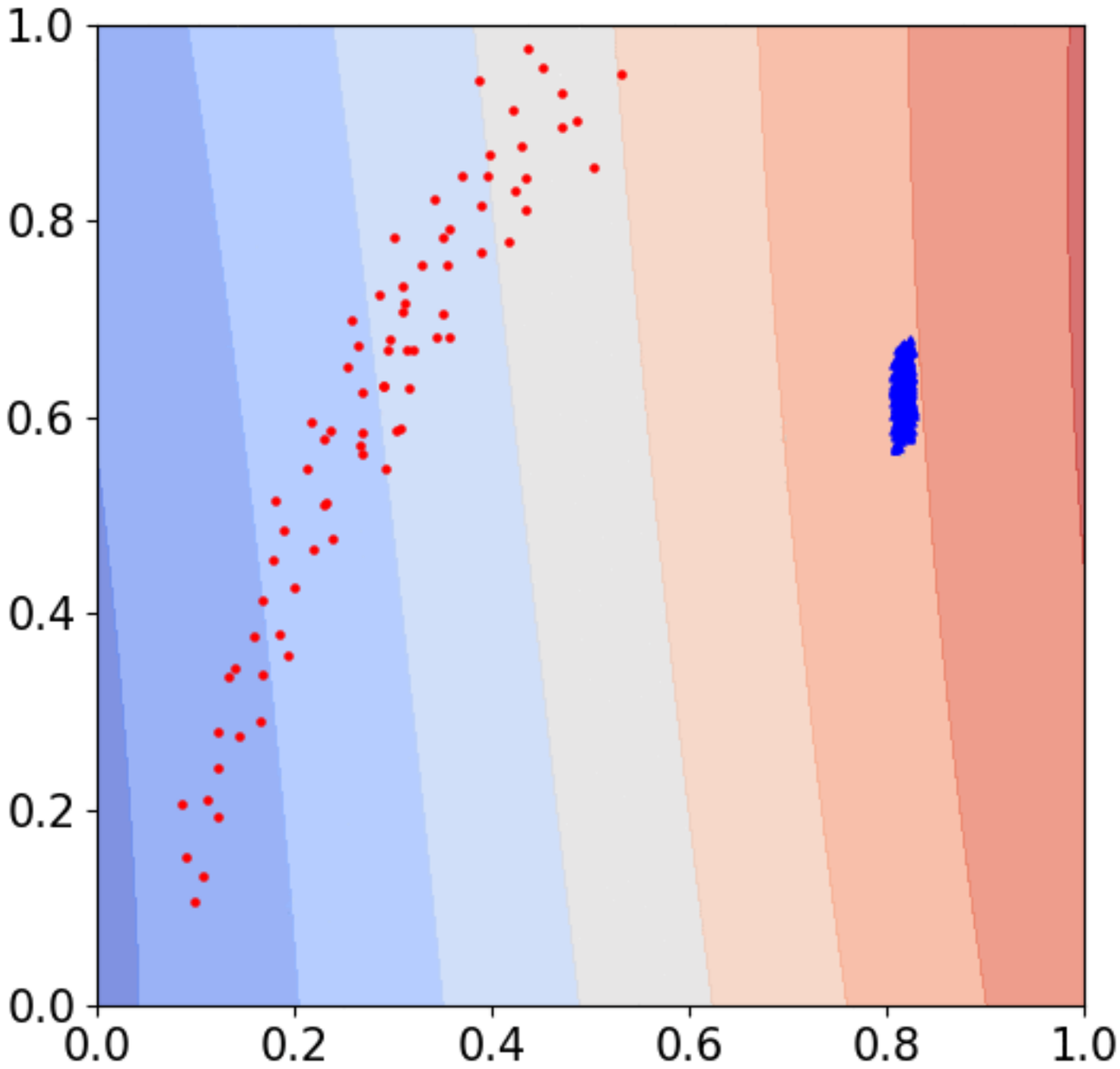


Generator fail due to zero gradient

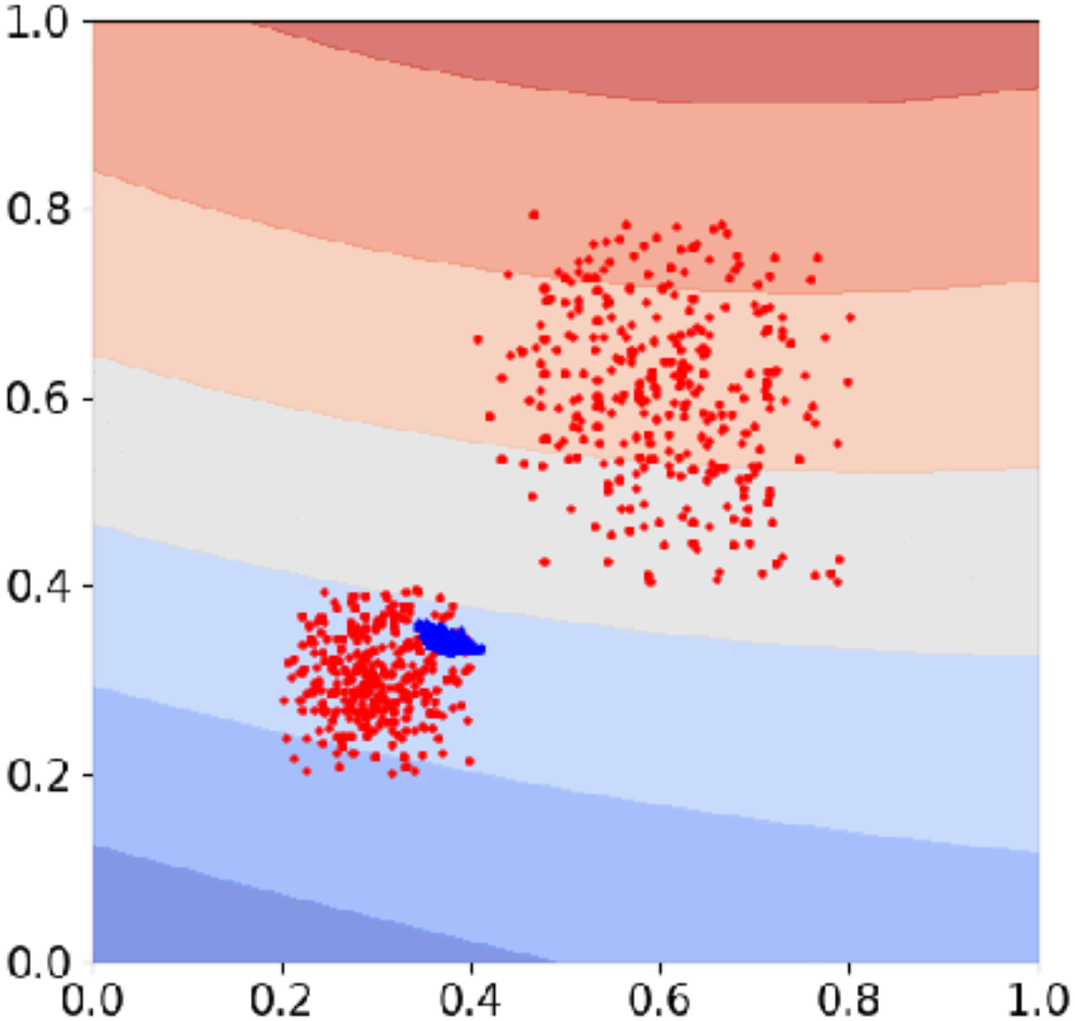


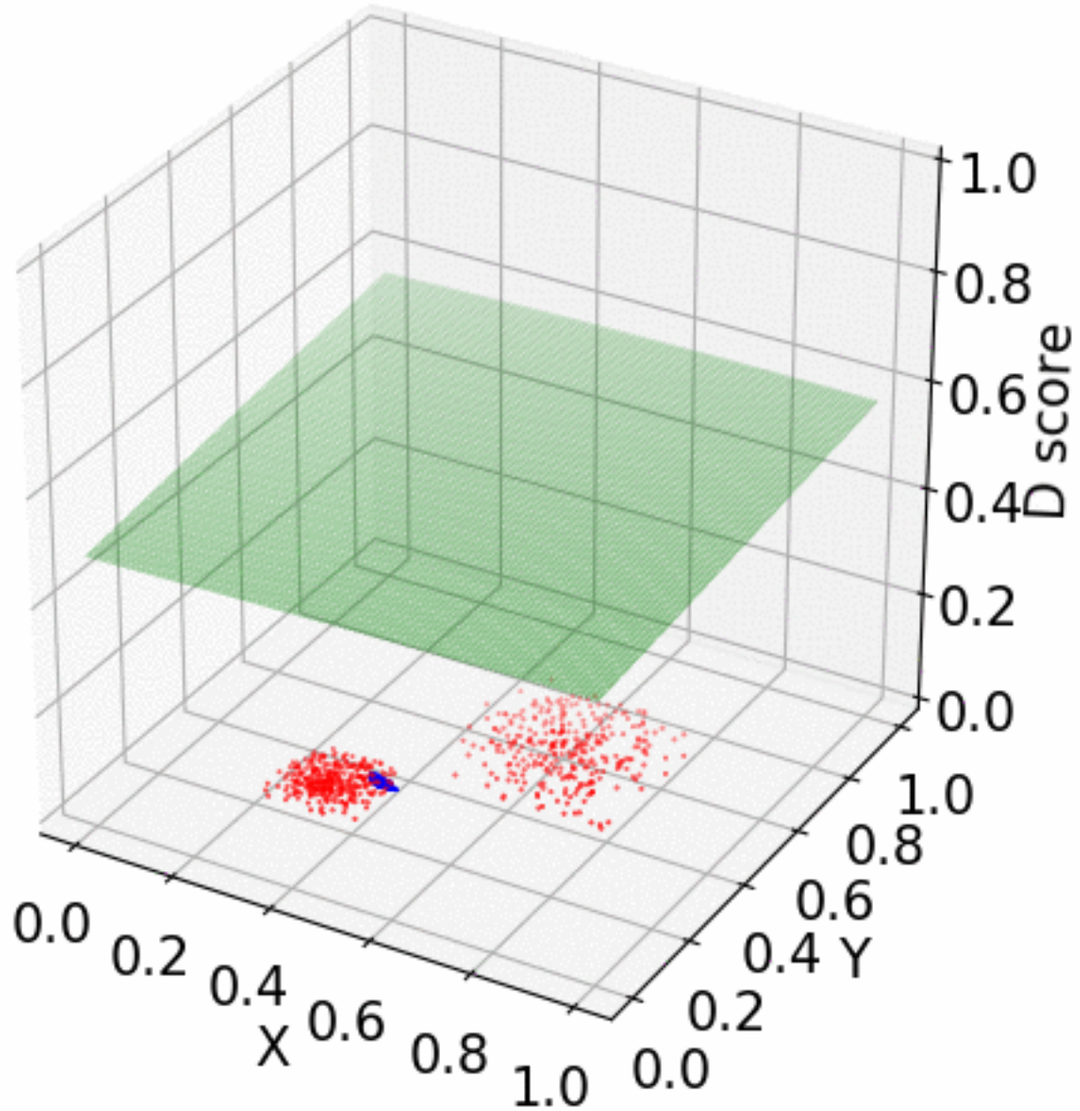


Generator fail due to zero gradient (surrounded)



Mode collapse





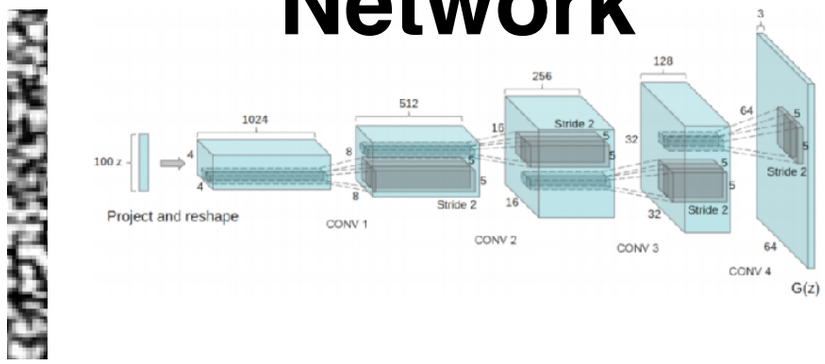
Improved Techniques for Training GANs

T. Salimans, I Goodfellow *et al*

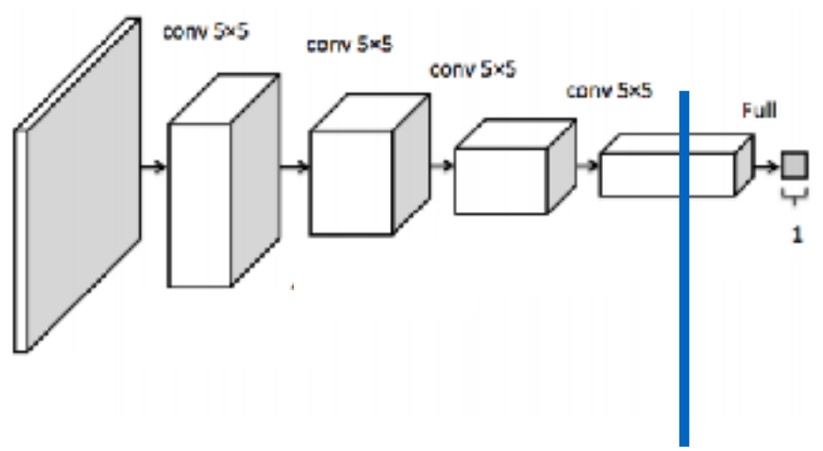
1. Feature matching

for training the generator

Generator Network



Discriminator Network



$$\rightarrow f(x)$$

Cost function for generator

$$C_g(w,\theta) = \left\| \frac{1}{N}\sum_i \vec{f}_w(x_i) - \frac{1}{M}\sum_j \vec{f}_w(g_\theta(z_j)) \right\|_2^2$$

$$\theta^* = \arg\min_\theta C_g(w,\theta)$$

Discriminator cost function use in original GAN paper

$$C(w,\theta) = -\left(\frac{1}{N^+}\sum_i \log(D_w(x_i)) + \frac{1}{N^-}\sum_j \log(1 - D_w(g_\theta(z_j)))\right)$$

$$\theta^* = \arg\max_\theta C(w,\theta) \qquad \text{keeping } w \text{ constant}$$

2. Minibatch discrimination

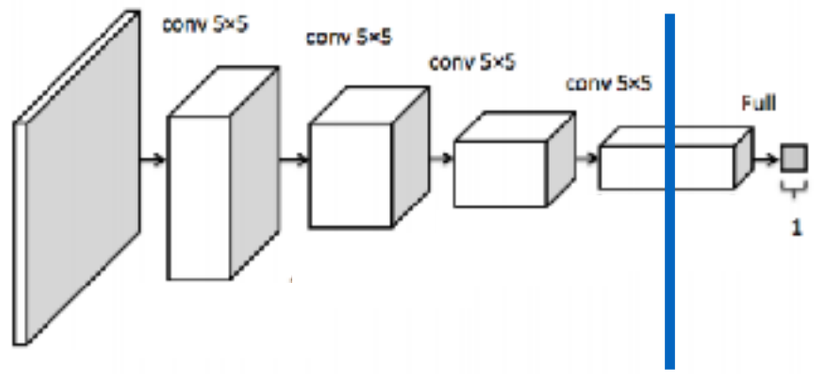
for training the discriminator

$\vec{f}(x_i)$ ⟶ **T** ⟶ $\vec{o}(x_i)$

Read paper for details

o contains information on how this data point cluster with other data points in the minibatch

Discriminator Network



$\vec{f}(x)$

Concatenate features to create a longer feature

$$\left(\vec{f}(x_i), \vec{o}(x_i)\right)$$

Put this new feature back into subsequent layers of discriminator

3. Historical averaging

Add a term in cost function

$$\left\| \theta - \frac{1}{t} \sum_i \theta_i \right\|^2$$

Effect, at every step, theta does not change too much in magnitude and direction

4. One-sided label smoothing

Changes in discriminator training

Instead of asking discriminator to output ‘1’ for real data, ask discriminator to output a softer value like ‘0.9’ for real data.

Fake data still has label ‘0’

Encourage a softer decision boundary, e.g. don’t need to saturate out the sigmoid and make gradient zero

5. Virtual batch normalisation

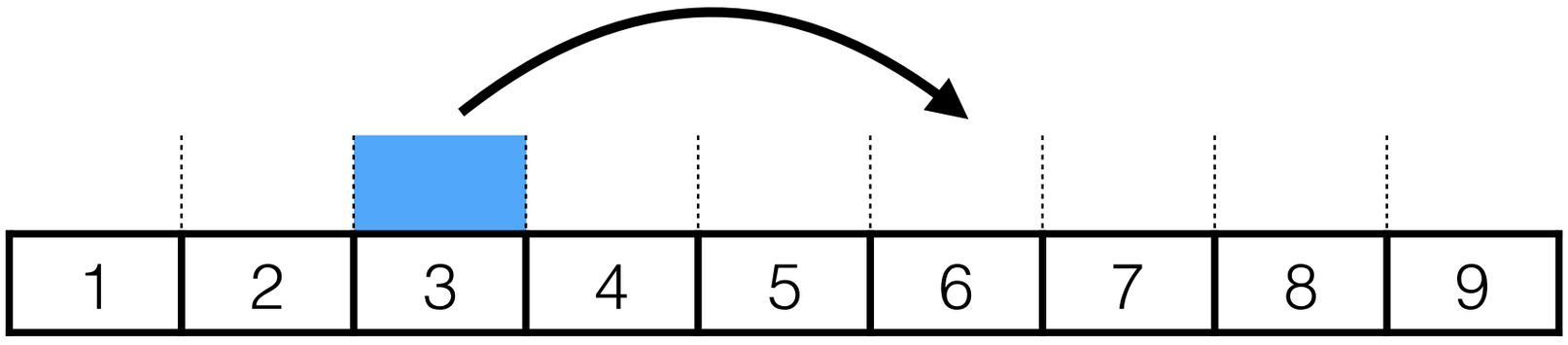
Instead of normalising differently for each minibatch, use a reference batch so that normalisation is consistent across minbatches.

What I would do is to use one statistics for normalisation across all data, e.g. compute z-score using one mean and one variance

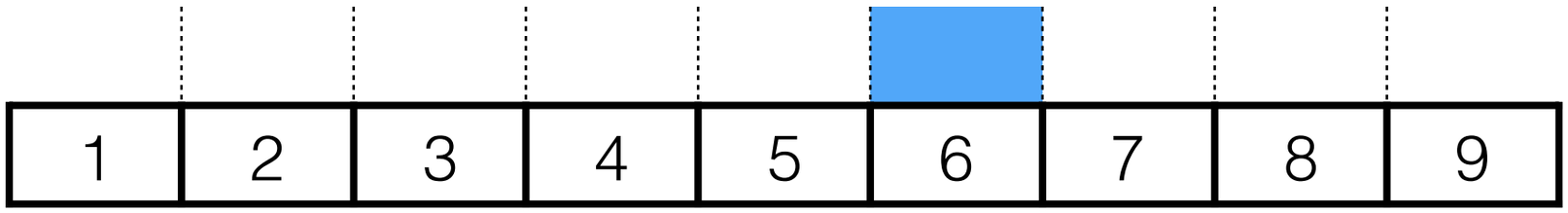
What paper proposed is to recalculate mean and variance for each data point, which will increase computational time a lot

Wasserstein GAN

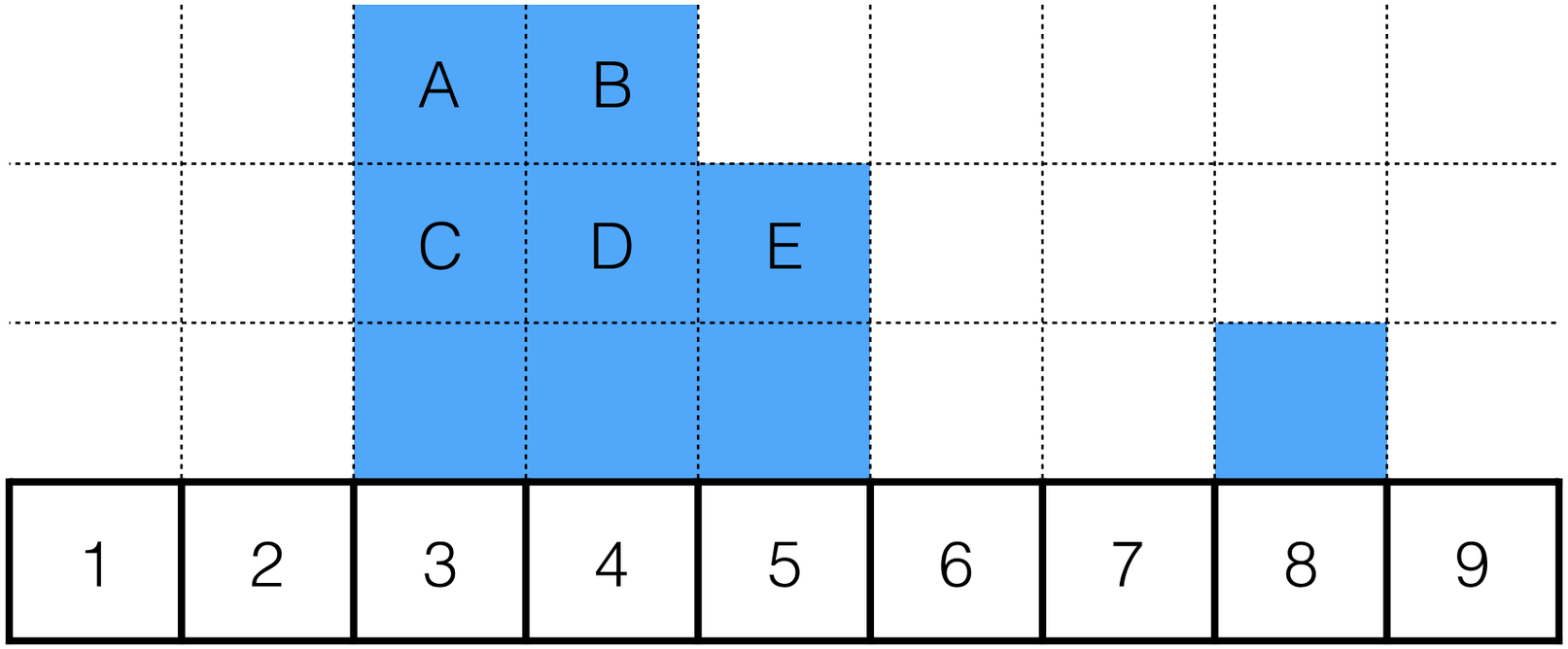
Earth Mover's Distance

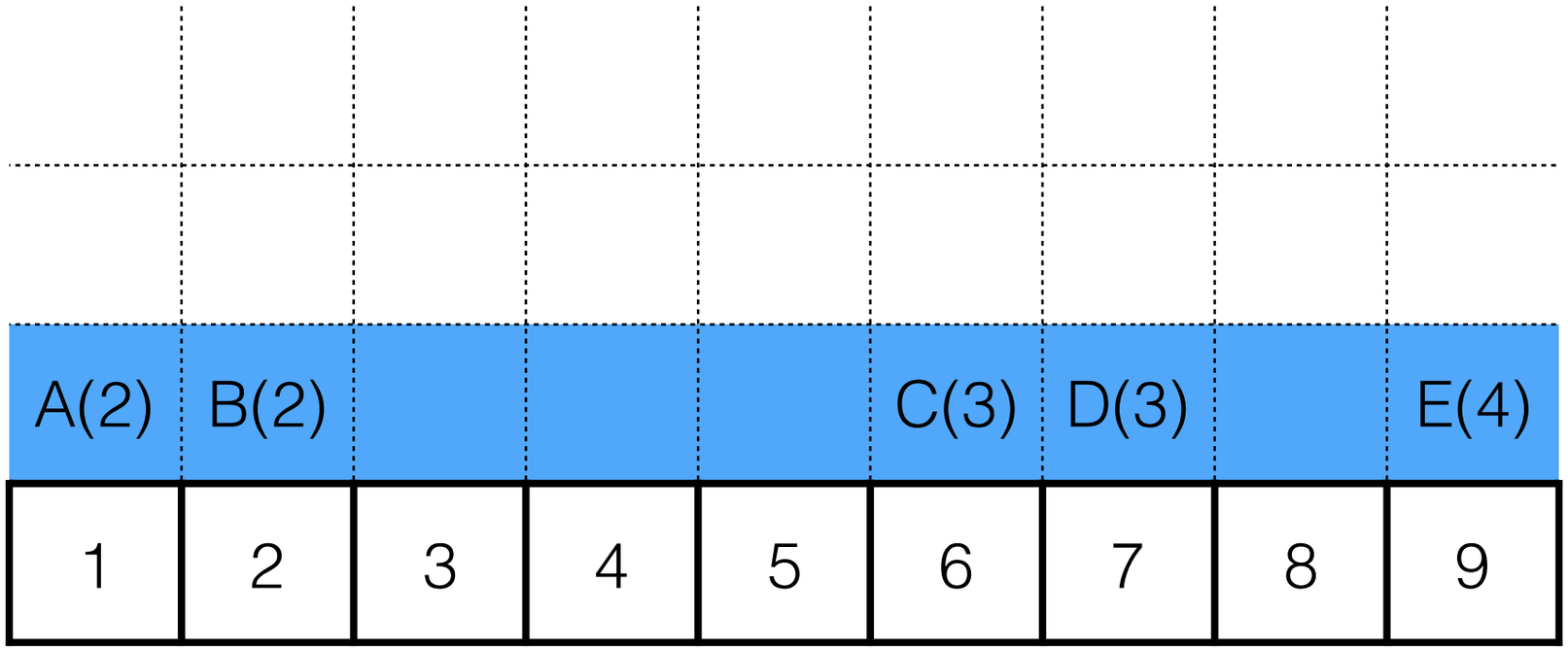


cost = 3



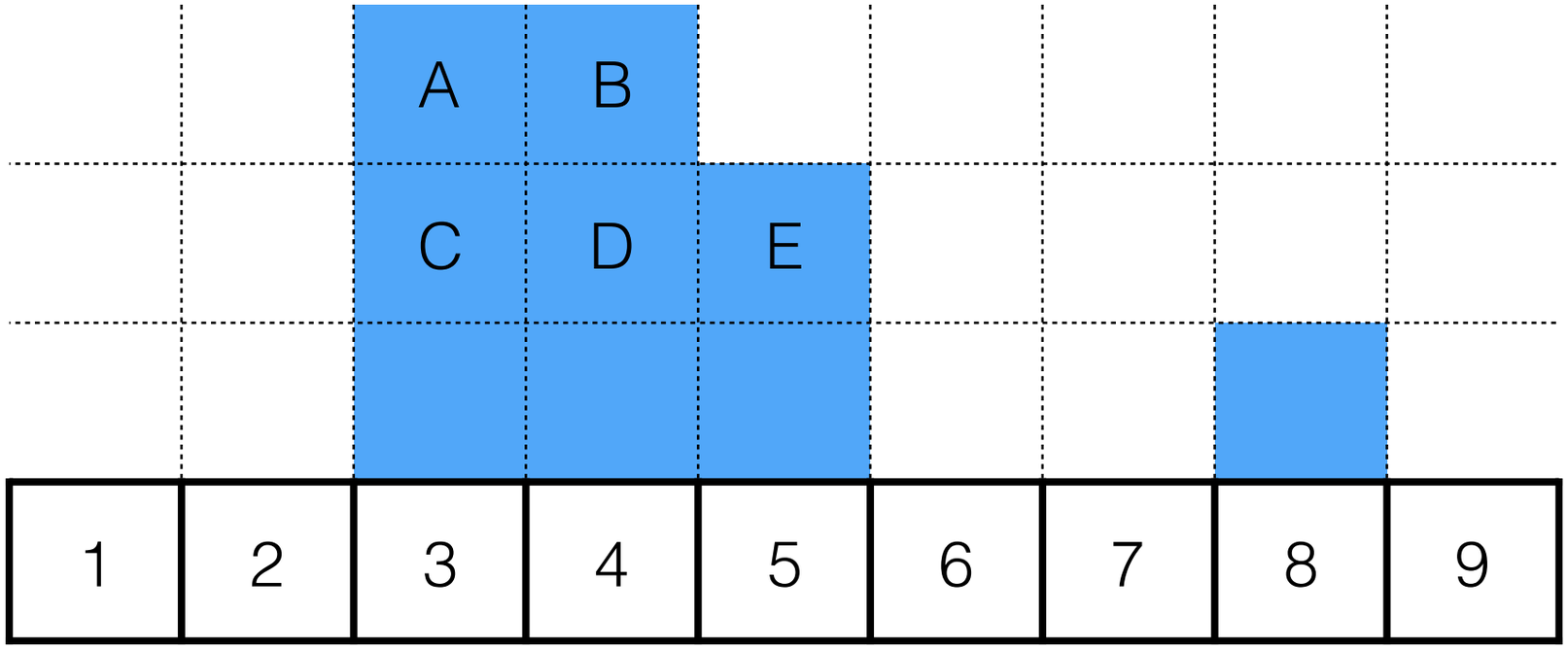
Earth Mover's Distance

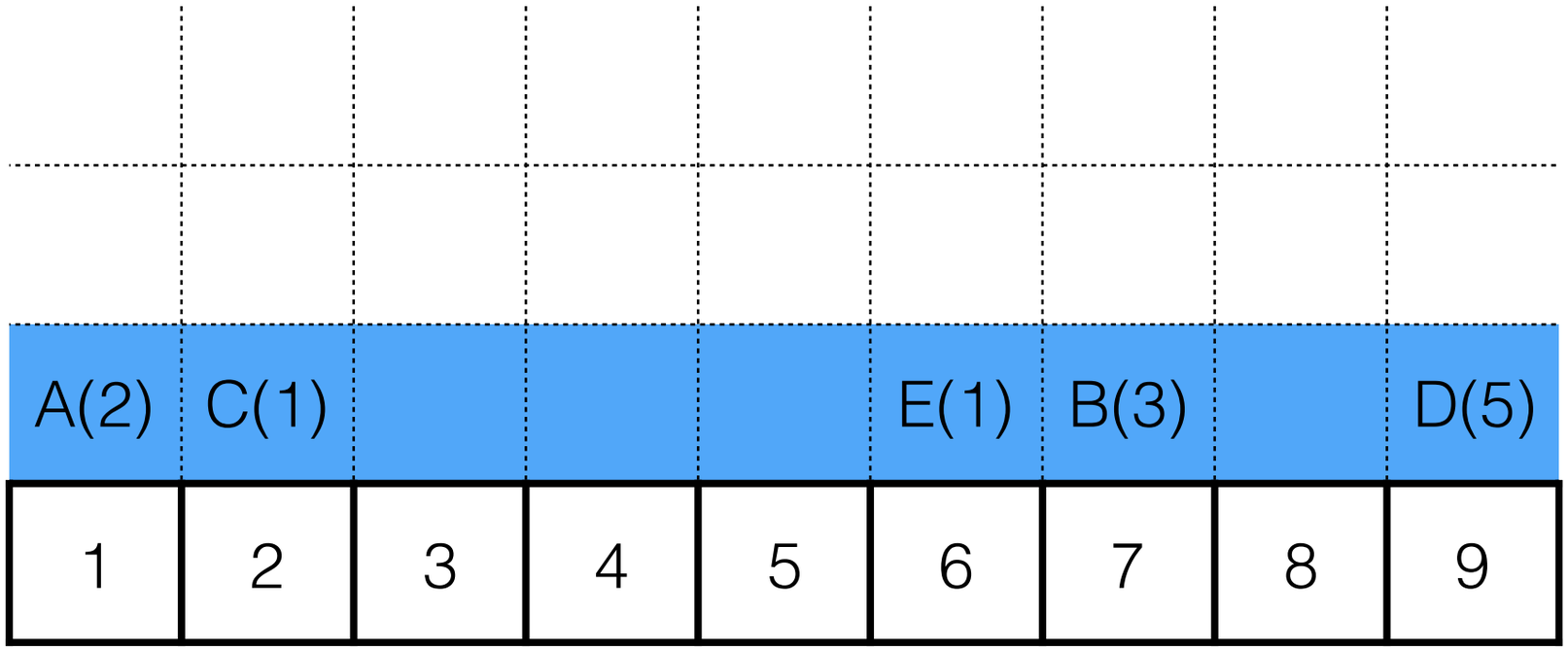




cost = 14

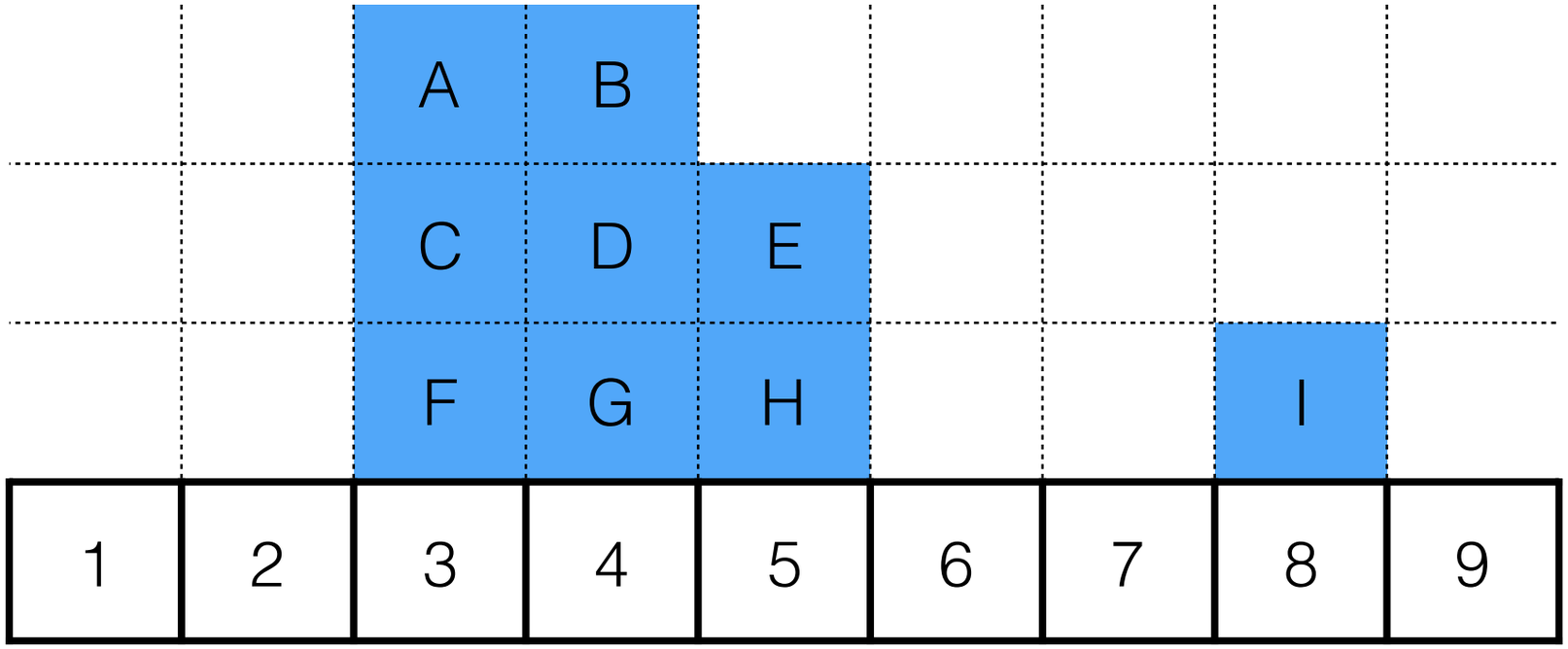
Earth Mover's Distance

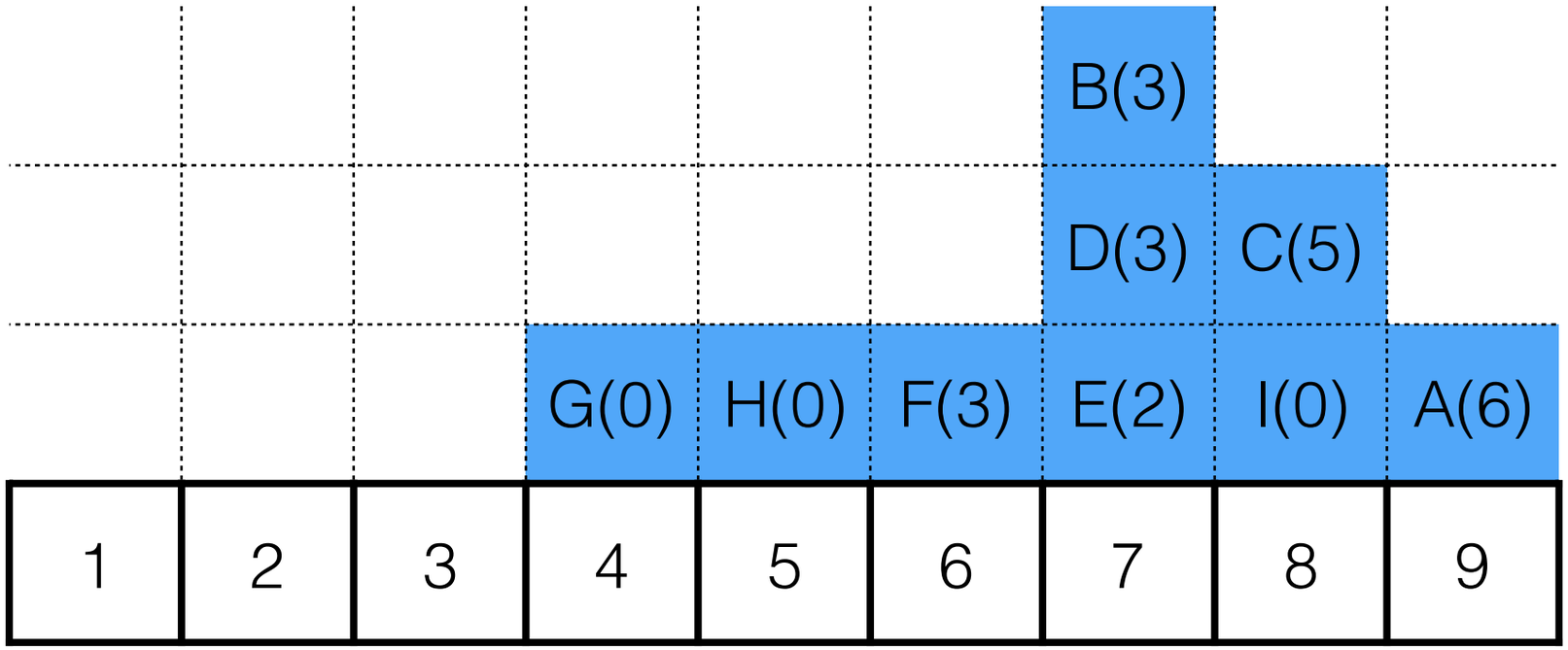




cost = 12

Earth Mover's Distance





cost = 22

Distribution of real and fake data

$P_r(x)$ is the distribution of real data, it is constant

$P_\theta(g_\theta(z))$ is the distribution of generated data
it changes when generator change

$W(P_r(x), P_\theta(g_\theta(z)))$

is the Earth Mover's Distance between real and generated data

Idea of Wasserstein GAN

No discriminator (in the paper they use the word ‘critic’)

Given $P_r(x)$

Initialize $g_\theta(z_i)$ and generate $P_\theta(g_\theta(z))$

Adjust generator such that

$W(P_r(x), P_\theta(g_\theta(z)))$ is minimised

Kantorovich-Rubinstein duality

I did not prove it

$$W(P_r, P_\theta) = \sup_{\|f\|<1} \left(\frac{1}{N} \sum_i f(x_i) - \frac{1}{M} \sum_j f(g(z_j)) \right)$$

over 1-Lipschitz functions

(loosely speaking gradient does not exceed one)

Sample from $P_r(x)$

Sample from $P_\theta(g_\theta(z))$

Kantorovich-Rubinstein duality

I did not prove it

$$W(P_r, P_\theta) = \sup_{\|f\|<1} \left(\frac{1}{N} \sum_i f(x_i) - \frac{1}{M} \sum_j f(g(z_j)) \right)$$

Sample from $P_r(x)$

Sample from $P_\theta(g_\theta(z))$

This is a constrained optimisation problem!

Representation of 1-Lipschitz function

One way to parameterise a function is to use a neural network. By adjusting weights, we create different functions.

When network is small, class of function obtained is limited.

When network is deep, we can represent more functions.

$$W(P_r, P_\theta) = \sup_{\|f\|<1} \left(\frac{1}{N} \sum_i f(x_i) - \frac{1}{M} \sum_j f(g(z_j)) \right)$$

Represent $f(\cdot)$ by a neural network $f_w(\cdot)$

Use gradient ascend to find $W(P_r(x), P_\theta(g_\theta(z)))$

Algorithm

Initialize w and θ

Generates $P_\theta(g_\theta(z))$

(1) Maximize by adjusting w

$$W(P_r, P_\theta) = \sup_{\|f\|<1} \left(\frac{1}{N} \sum_i f_w(x_i) - \frac{1}{M} \sum_j f_w(g(z_j)) \right)$$

(2) Minimize Earth Mover's distance

$$\arg\min_\theta W(P_r, P_\theta)$$

(3) Repest (1) & (2)

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

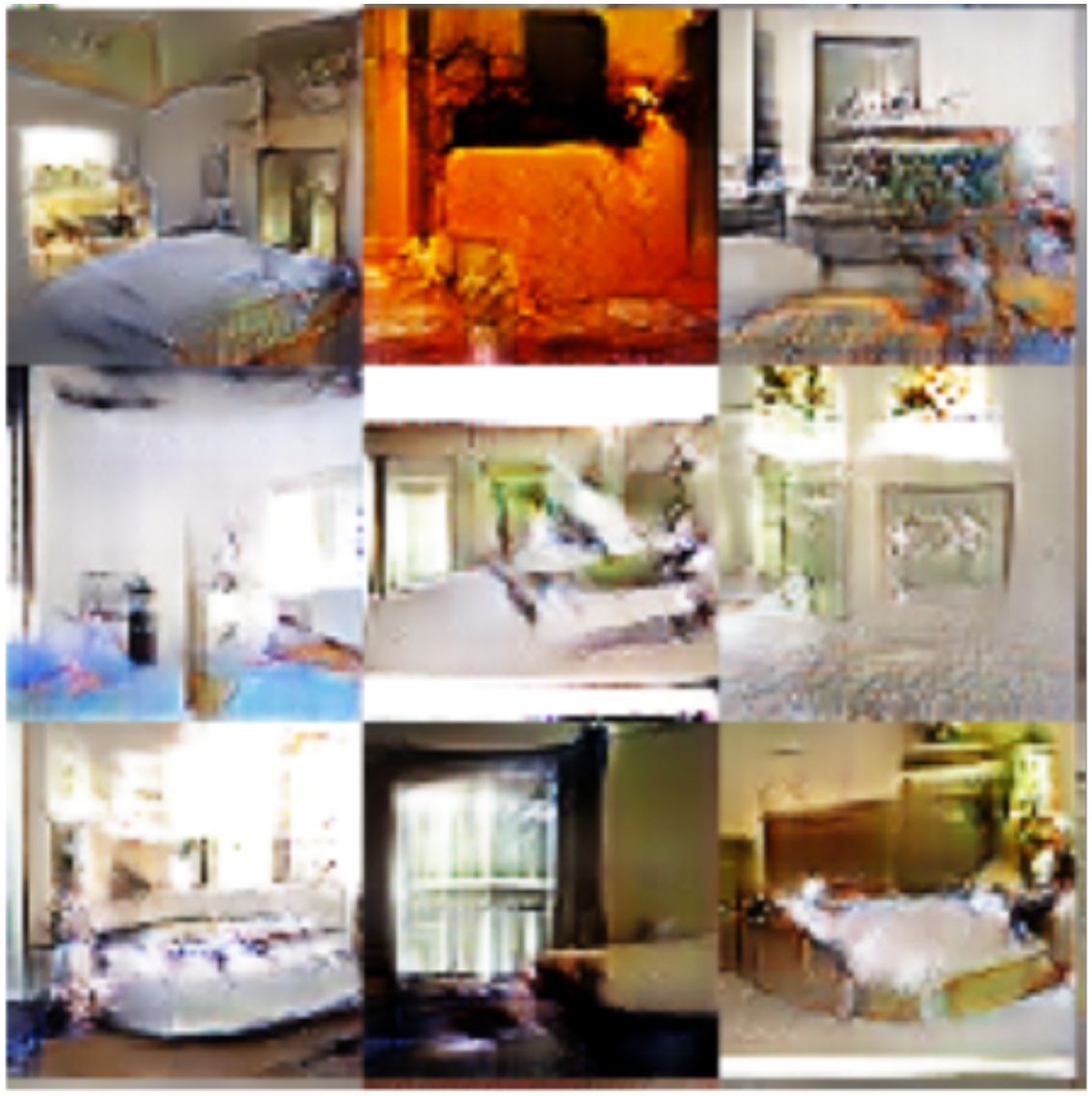
Assignment #3: There is something strange about this algorithm. What is it?

Deep Convolutional GAN

A. Radford, L. Metz, S. Chintala
Unsupervised Representation Learning with Deep Convolutional GAN, ICLR2016

one epoch

5 epoch



one epoch

5 epoch

Generator

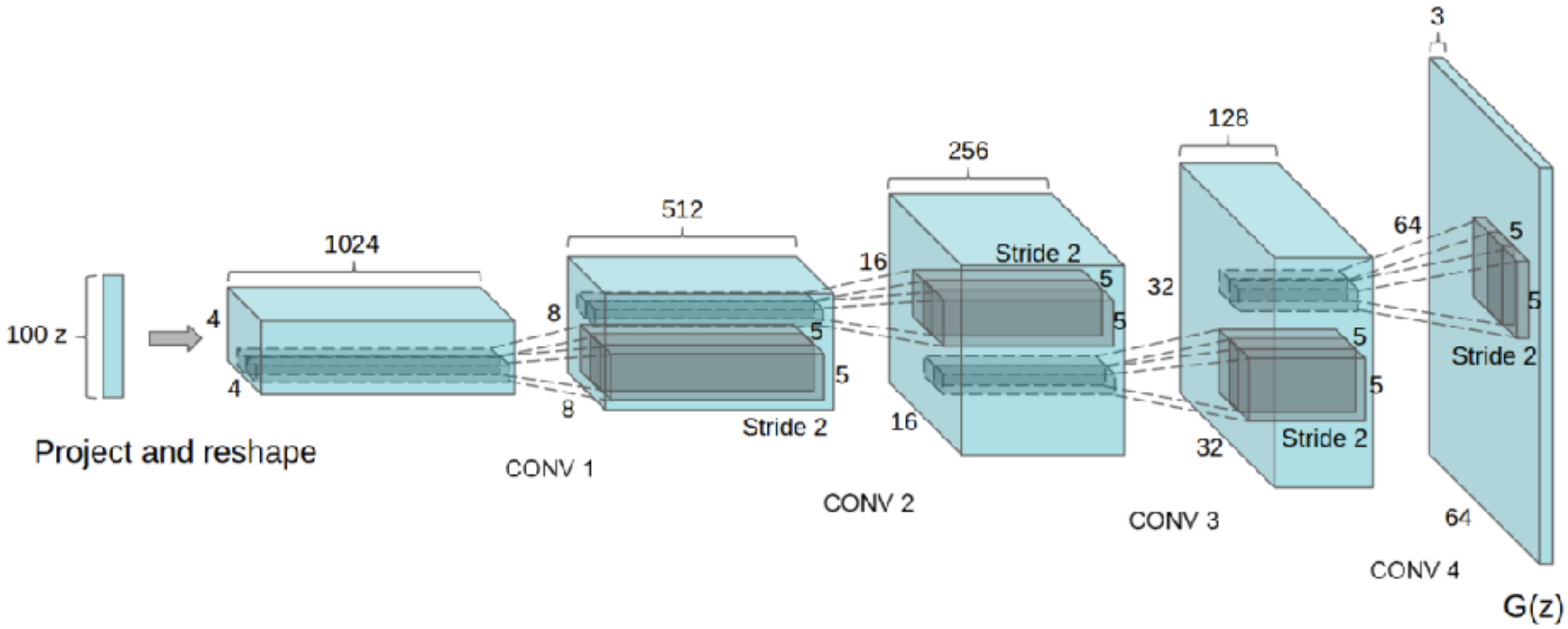
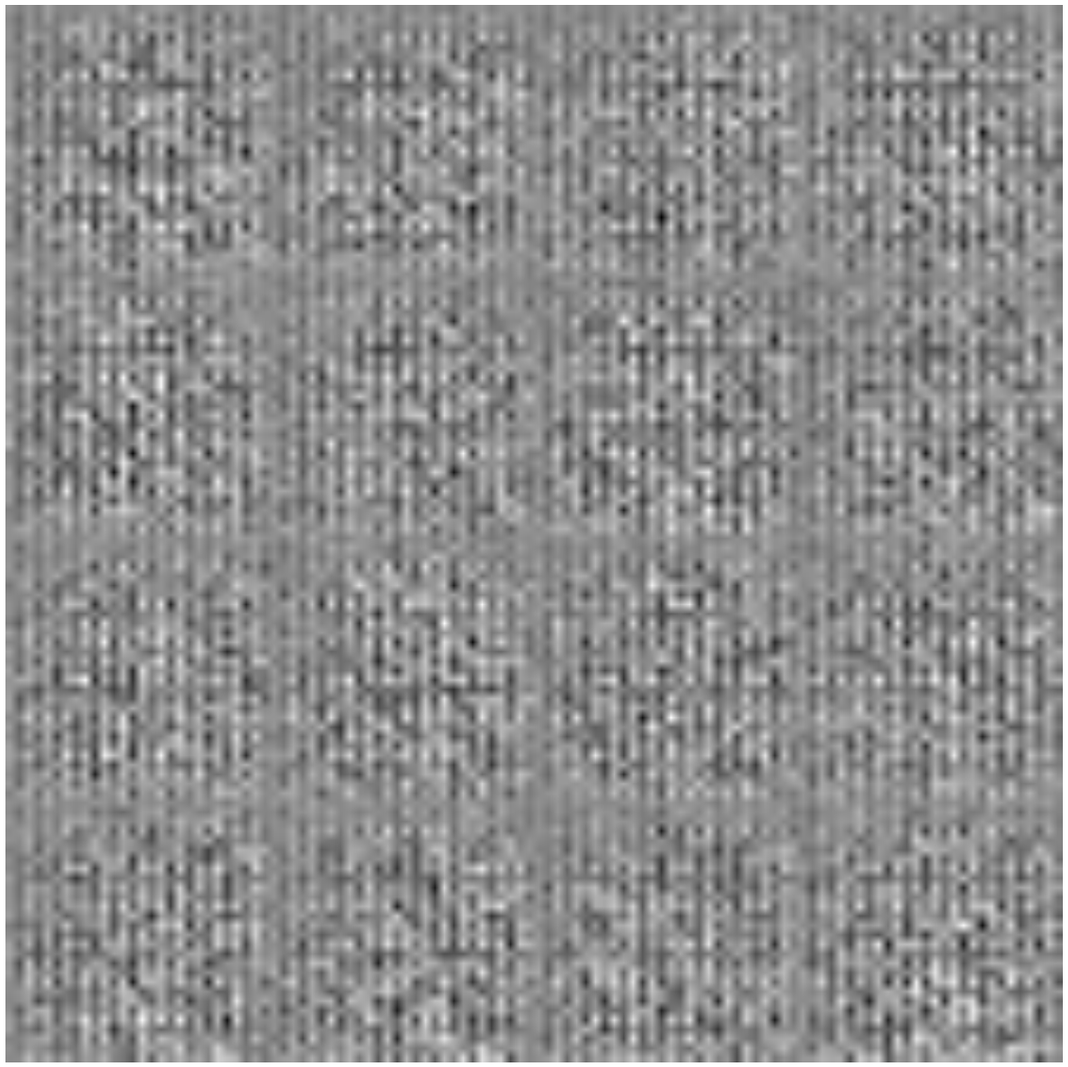


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

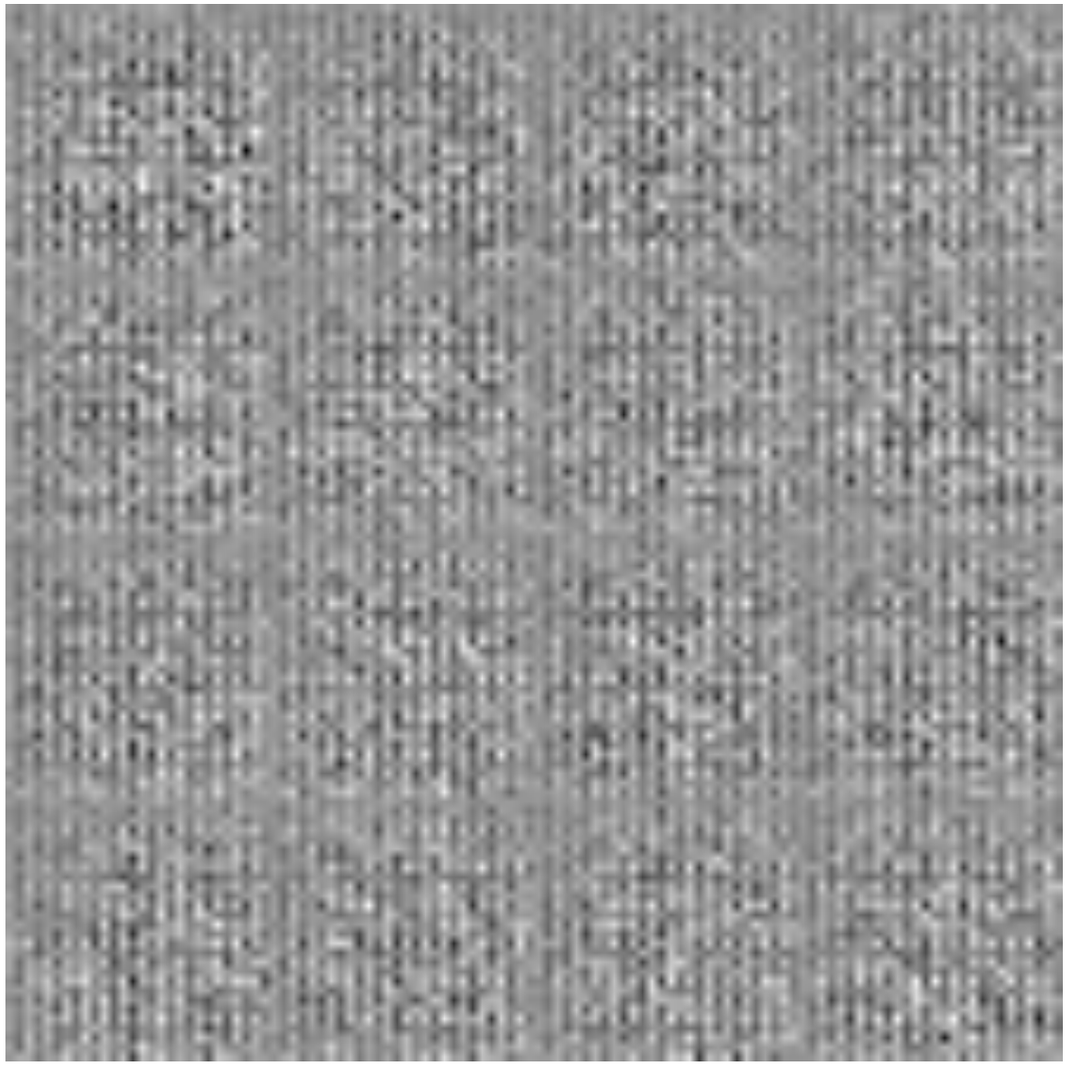
Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.



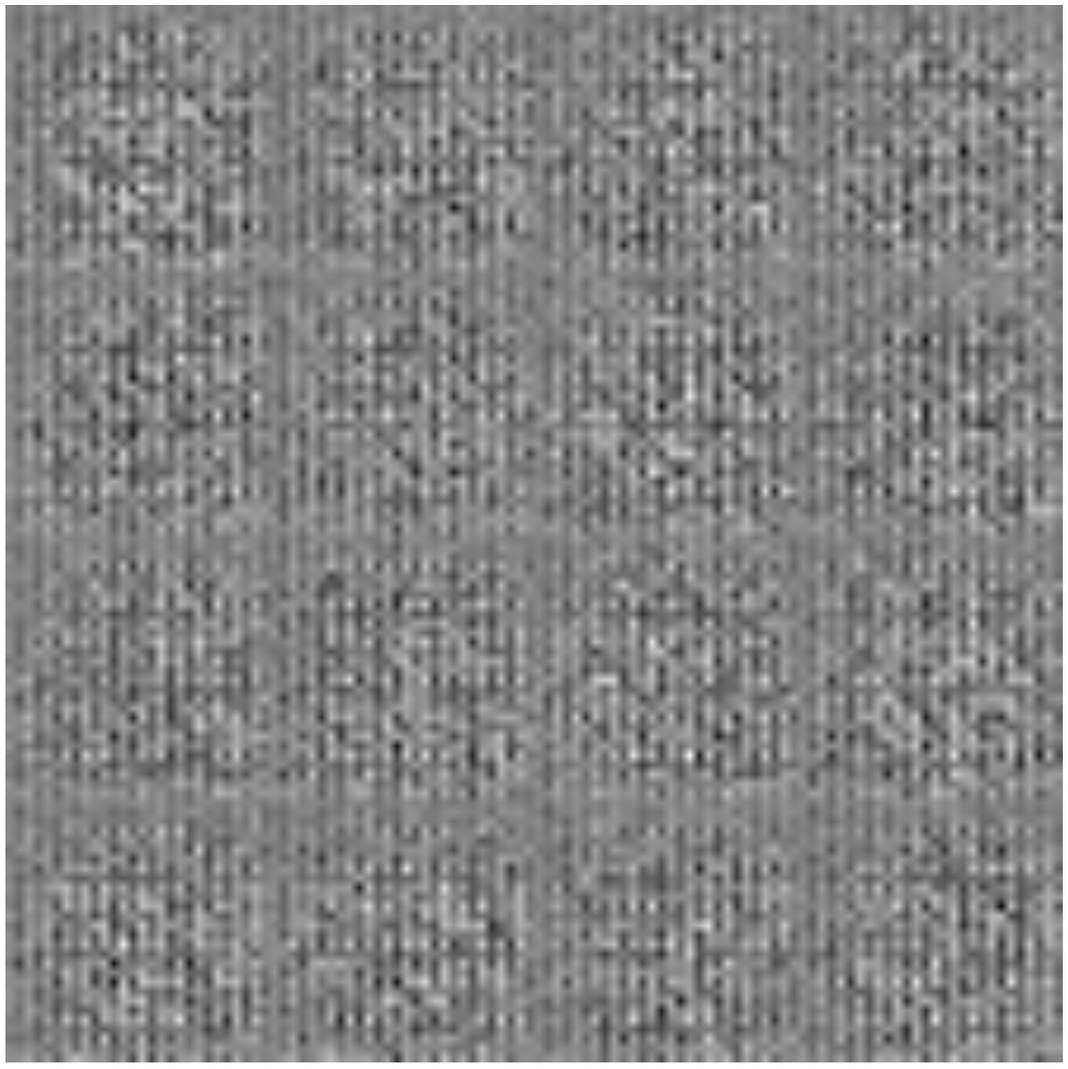
Every 10 iterations

Final generated digits



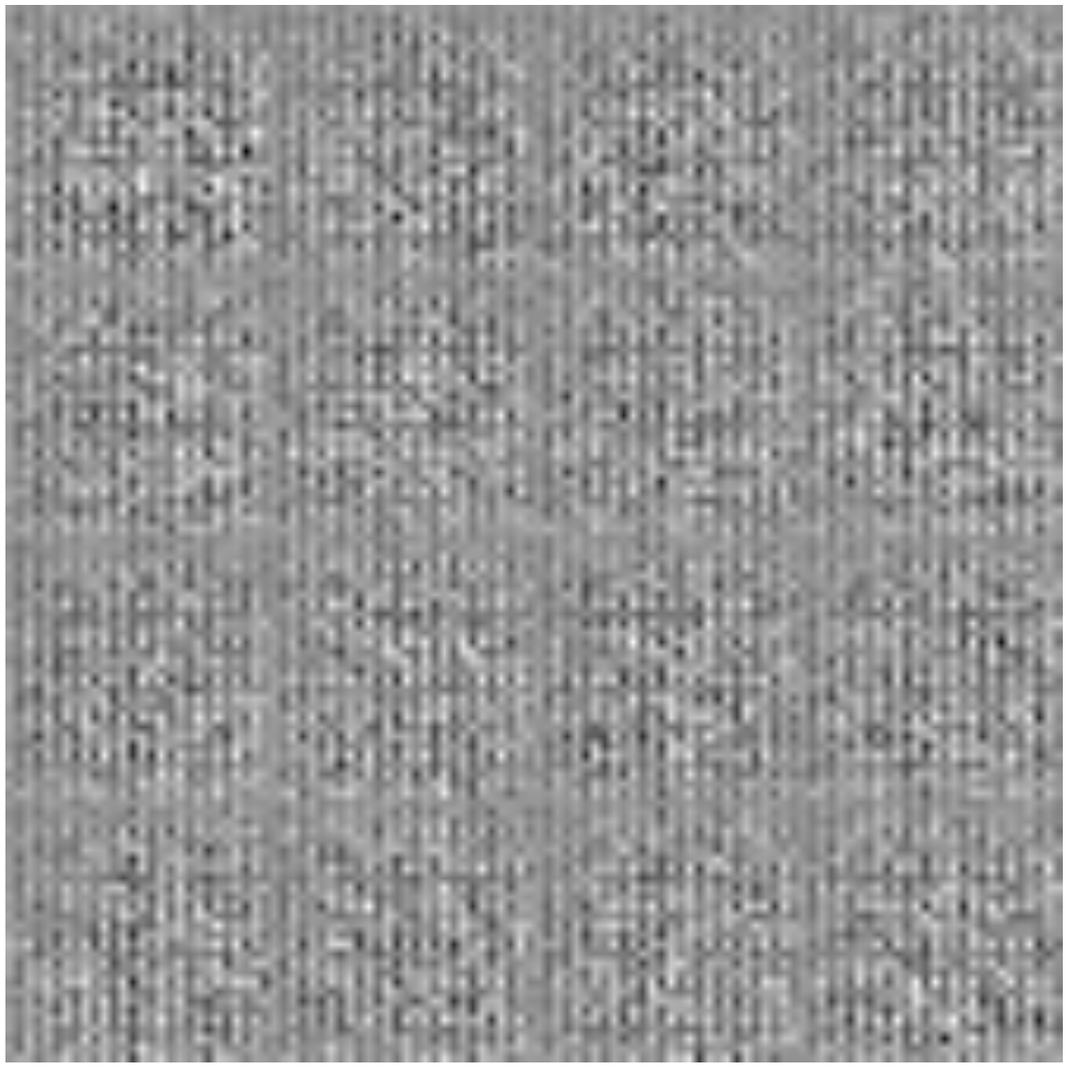
Every 10 iterations

Final generated digits



Every 10 iterations

Final generated digits

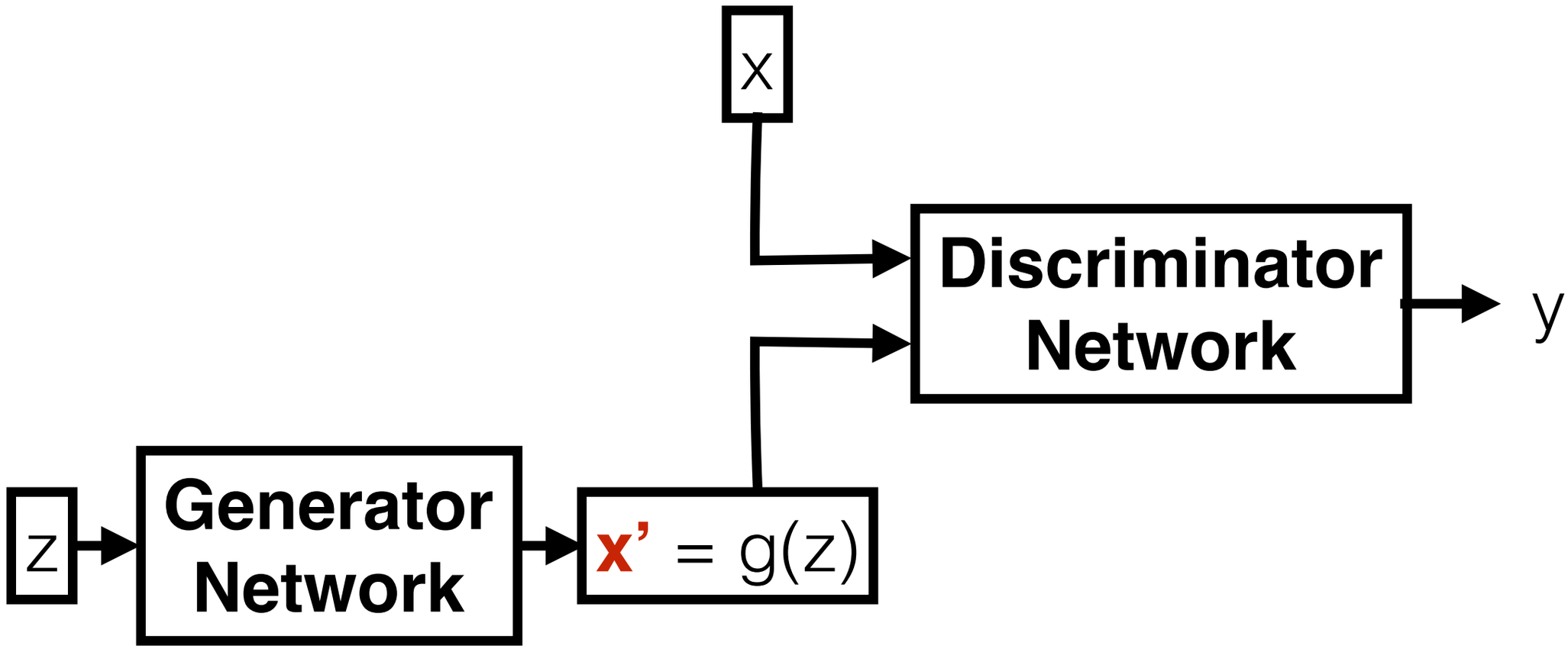


Every 10 iterations

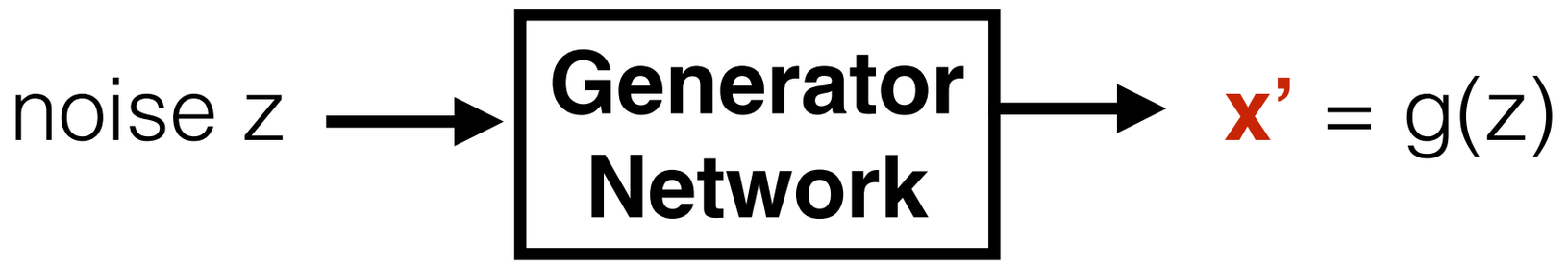
Final generated digits

BiGAN

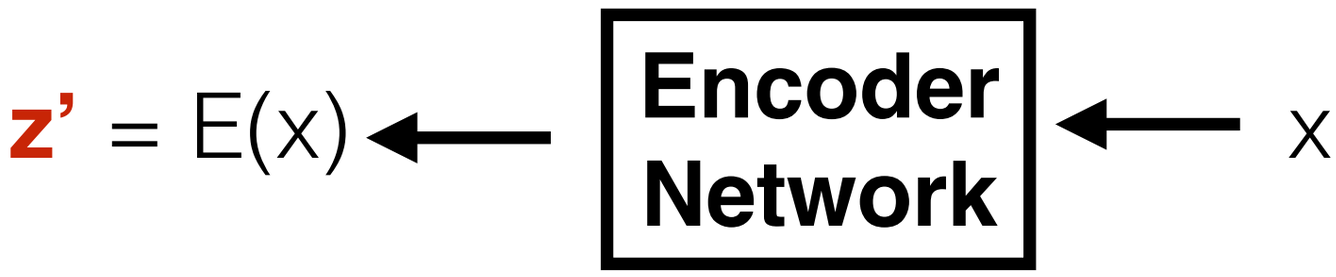
Bidirectional GAN

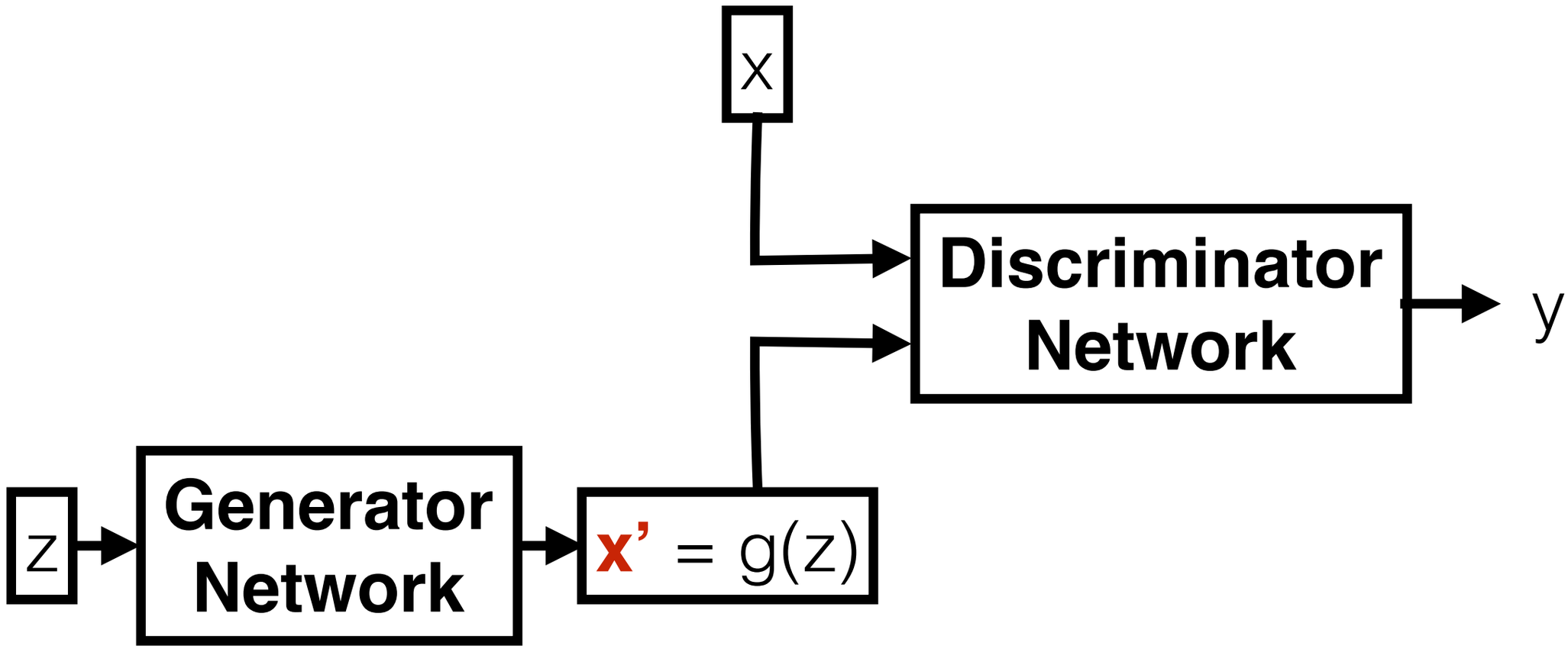

x
Discriminator Network
y
z
Generator Network
x’ = g(z)

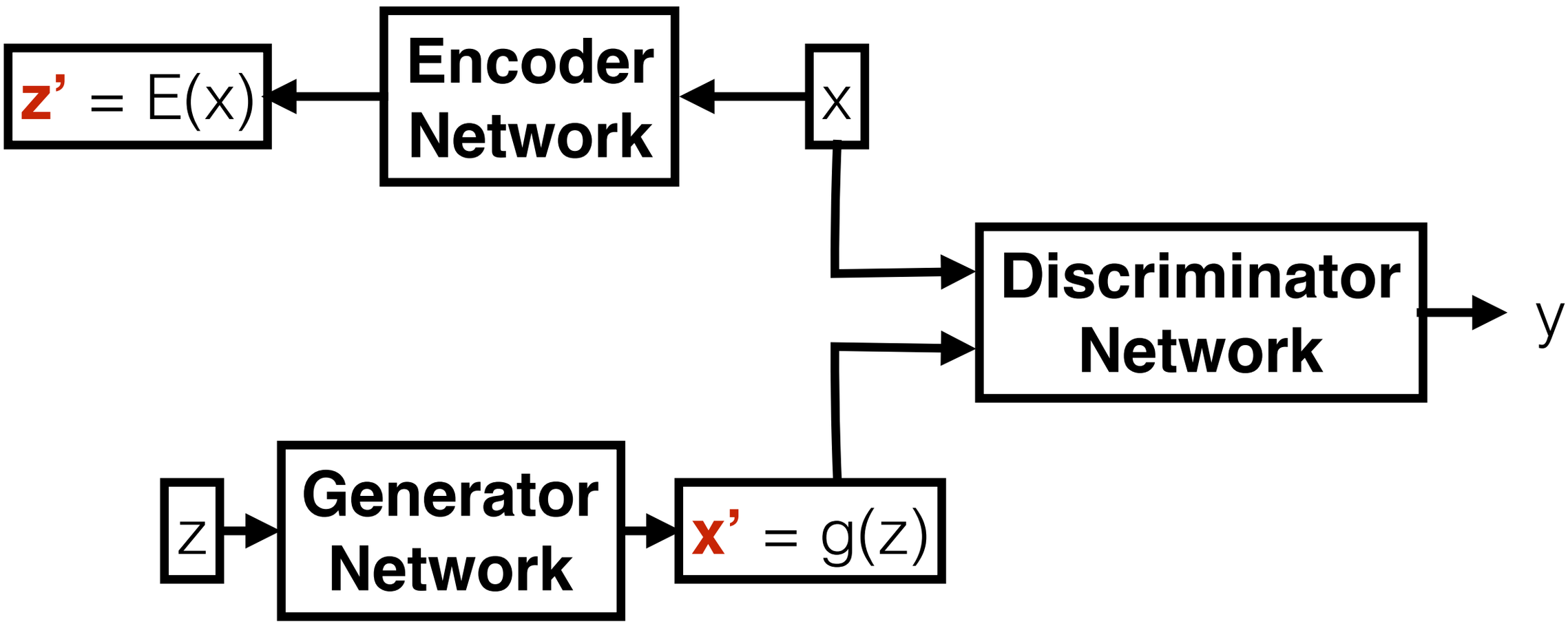
Give z, generate x’

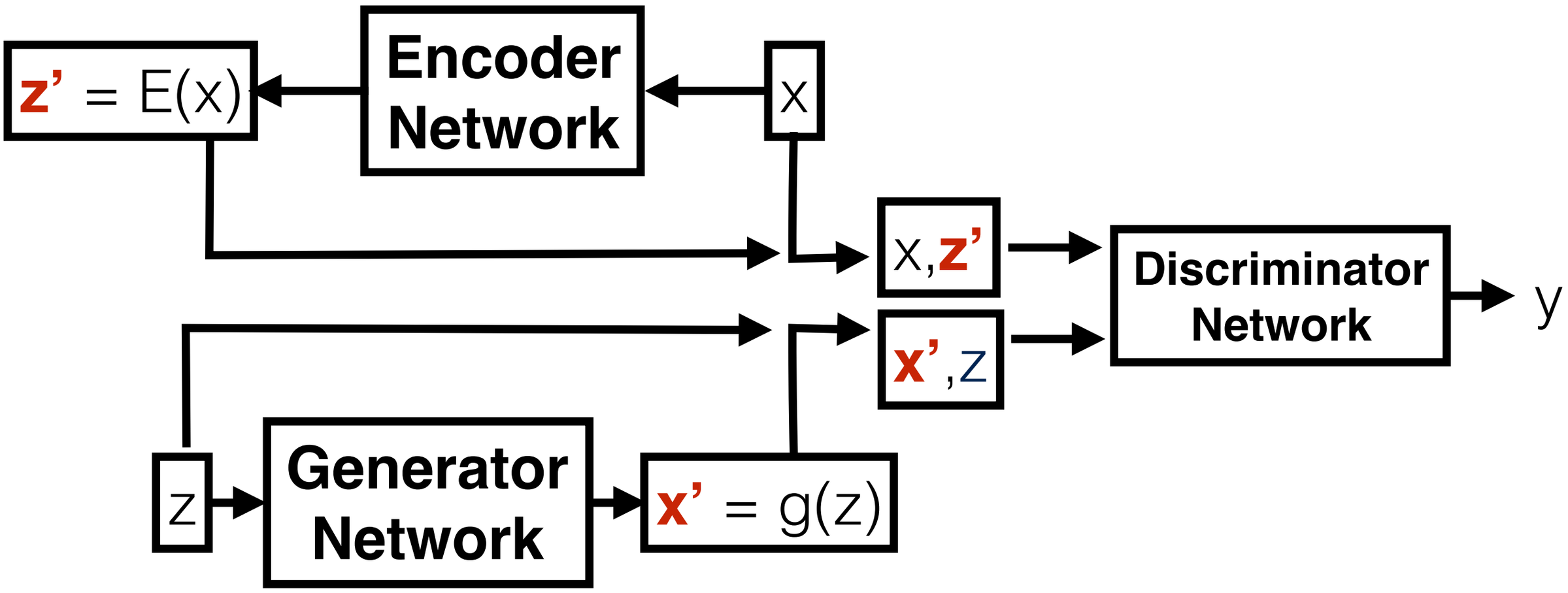


how about give x, generate z’




x
Discriminator Network
y
z
Generator Network
x’ = g(z)


z' = E(x)
Encoder Network
x
Discriminator Network
y
z
Generator Network
x' = g(z)


z' = E(x)
Encoder Network
x
x,z'
Discriminator Network
y
x',z
z
Generator Network
x' = g(z)

Cost function

$$C(w,\theta,\lambda) = -\left(\frac{1}{N^+}\sum_i \log\{D_w(x_i, E_\lambda(x_i))\} + \frac{1}{N^-}\sum_j \log\{1 - D_w(g_\theta(z_j), z_j\}\right)$$

$$\max_{\theta,\lambda} \min_{w} C(w,\theta,\lambda)$$

Adjust generator and encoder such that all fake data looks like real ones

$$g_\theta(z_j) = x' \to x \qquad\qquad E_\lambda(x_i) = z' \to z$$

By this procedure, $g(E_\lambda(x_i)) \to x_i$

$$E_\lambda \approx g_\theta^{-1}$$

x

g(E(x))

x

g(E(x))

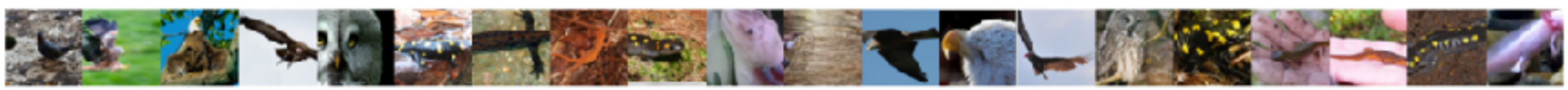
x

g(E(x))

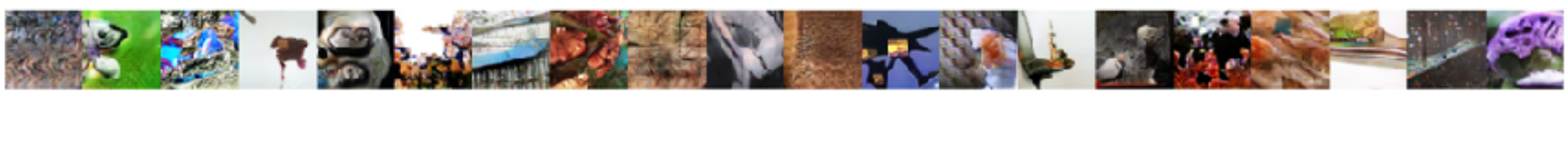
x

g(E(x))

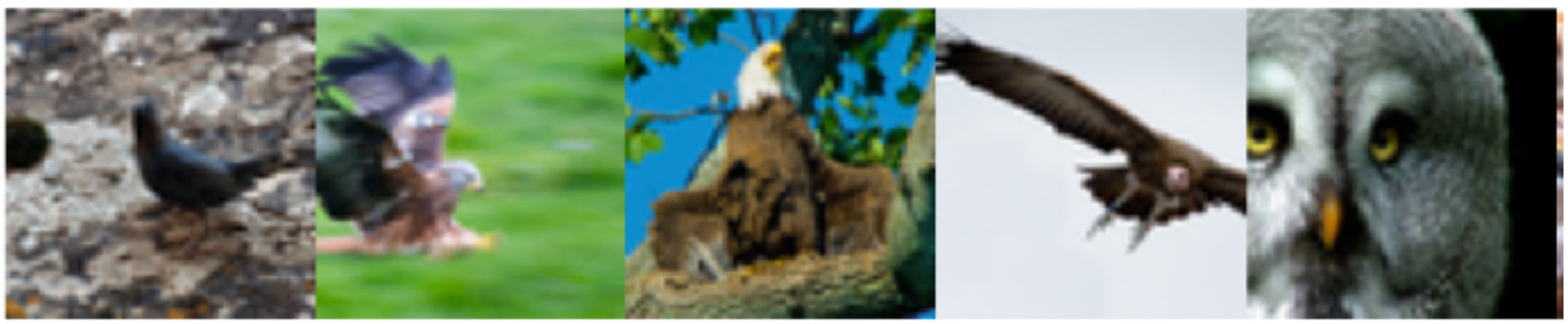
x



g(E(x))



x



g(E(x))

Assignment #3

we will announce this weekend