Generative Adversarial Network

Rookie Training Camp

Advisor: 李家岩 博士

Presenter: 歐子毓

Date: 2020.07.15

Agenda

01 Basic idea of GAN

02 GAN as structured learning

03 Can generator learn by itself

04 Can discriminator generate

05 Conditional GAN

POLab

Agenda

Conquer some issues using GANS

06

07

Implementation with PyTorch

Basic idea of GAN

Generation

We will control what to generate latter. → Conditional Generation

Image Generation

$$\begin{bmatrix} 0.3 \\ -0.1 \\ \vdots \\ -0.7 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.1 \\ \vdots \\ 0.7 \end{bmatrix} \begin{bmatrix} -0.3 \\ 0.1 \\ \vdots \\ 0.9 \end{bmatrix}$$

In a specific range

→ NN Generator →

Sentence Generation

$$\begin{bmatrix} 0.3 \\ -0.1 \\ \vdots \\ -0.7 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.1 \\ \vdots \\ 0.2 \end{bmatrix} \begin{bmatrix} -0.3 \\ 0.1 \\ \vdots \\ 0.5 \end{bmatrix}$$

→ NN Generator →

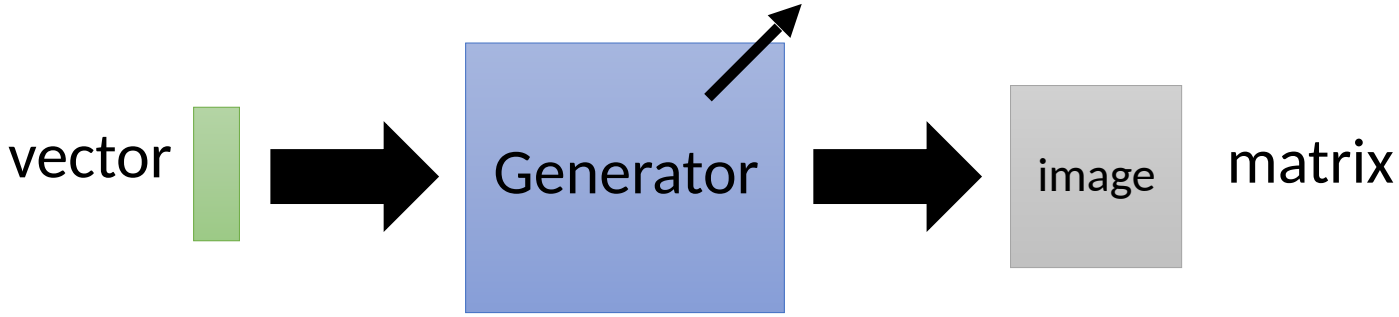
How are you?

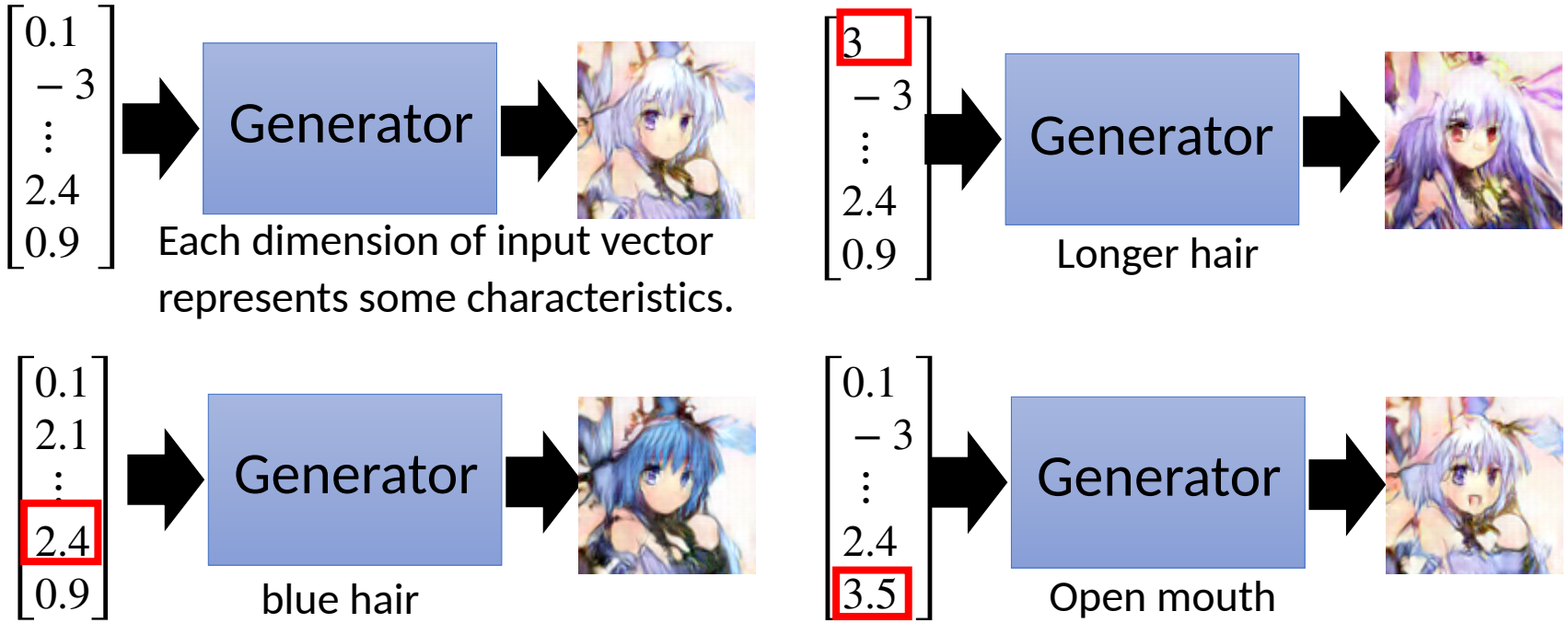
Good morning.

Good afternoon.

Basic Idea of GAN

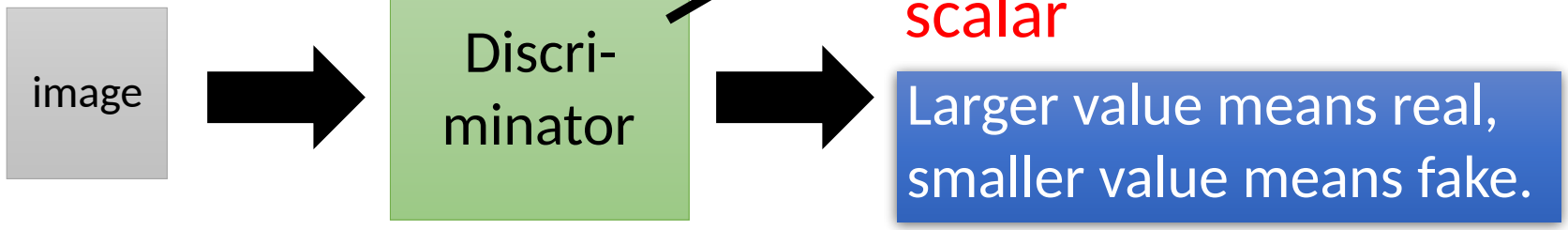
It is a neural network (NN), or a function.





Basic Idea of GAN

It is a neural network (NN), or a function.



Basic Idea of GAN

Generator

Brown

veins

Butterflies are not brown

Butterflies do not have veins

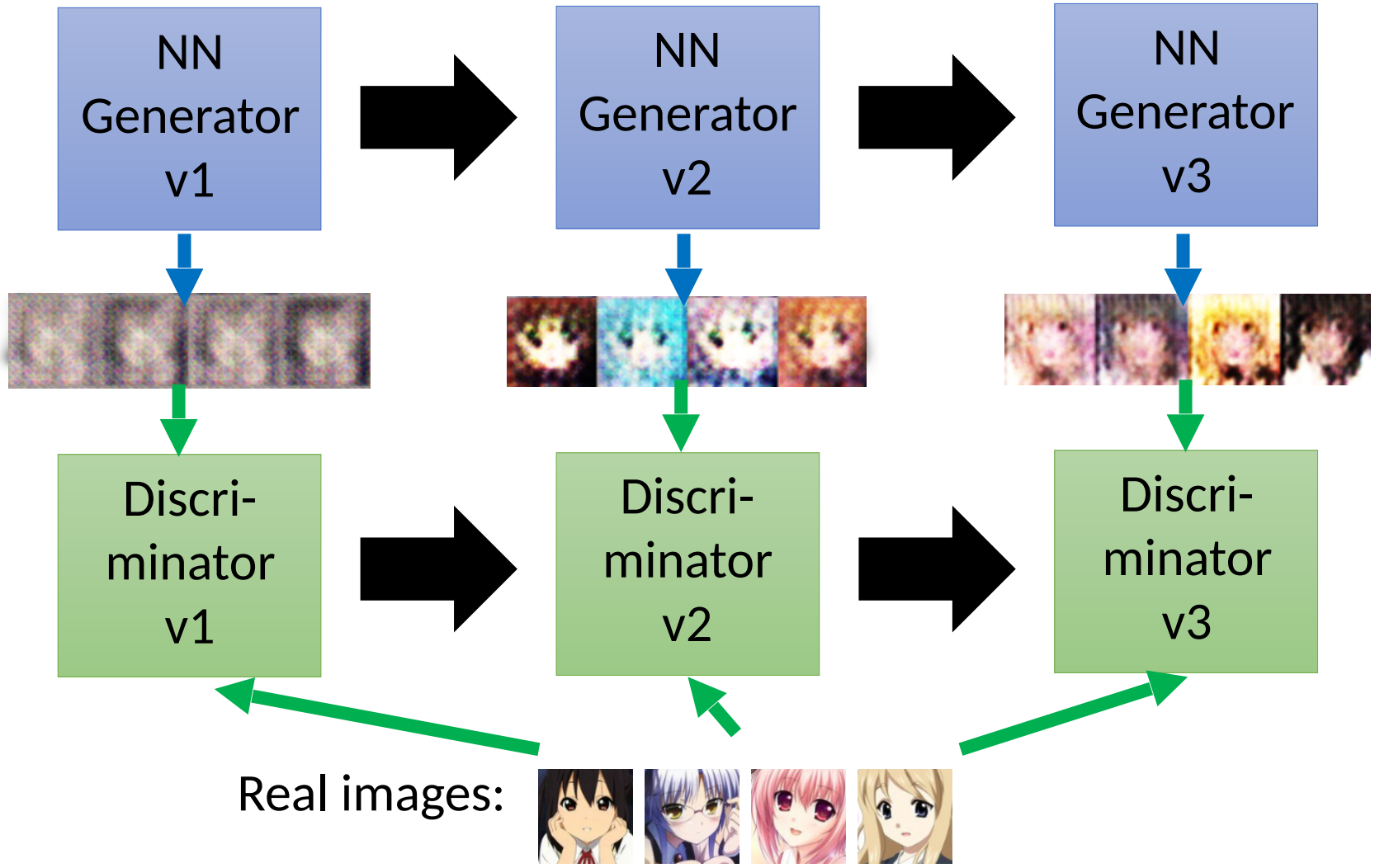
……

Discriminator

Basic Idea of GAN

This is where the term "***adversarial***" comes from.

You can explain the process in different ways…….

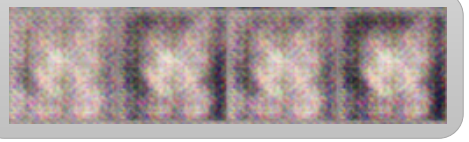


Basic Idea of GAN

Generator (student)

Discriminator (teacher)

Generator v1



No eyes

Discriminator v1

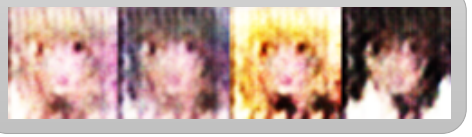
Generator v2

No mouth

Discriminator v2

為什麼不自己學？

Generator v3

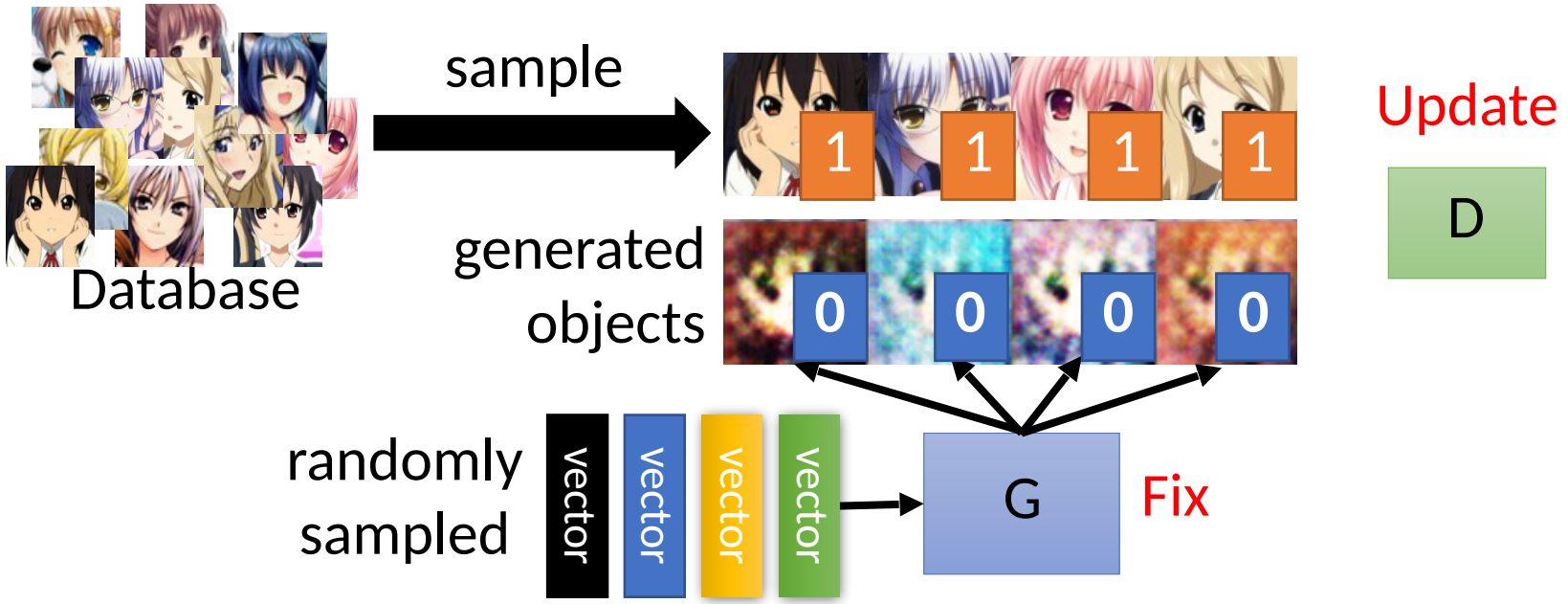


為什麼不自己做？

Algorithm

- Initialize generator and discriminator
- In each training iteration:

Step 1: Fix generator G, and update discriminator D



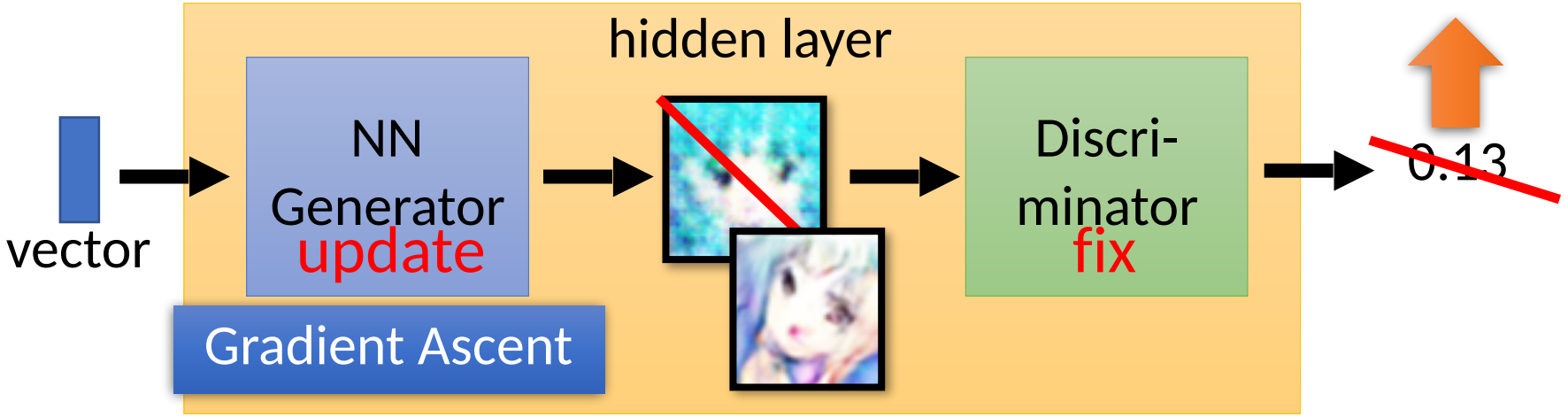
Discriminator learns to assign high scores to real objects and low scores to generated objects.

Algorithm

- Initialize generator and discriminator
- In each training iteration:

Step 2: Fix discriminator D, and update generator G

Generator learns to "fool" the discriminator



Algorithm

Initialize θ_d for D and θ_g for G

- In each training iteration:

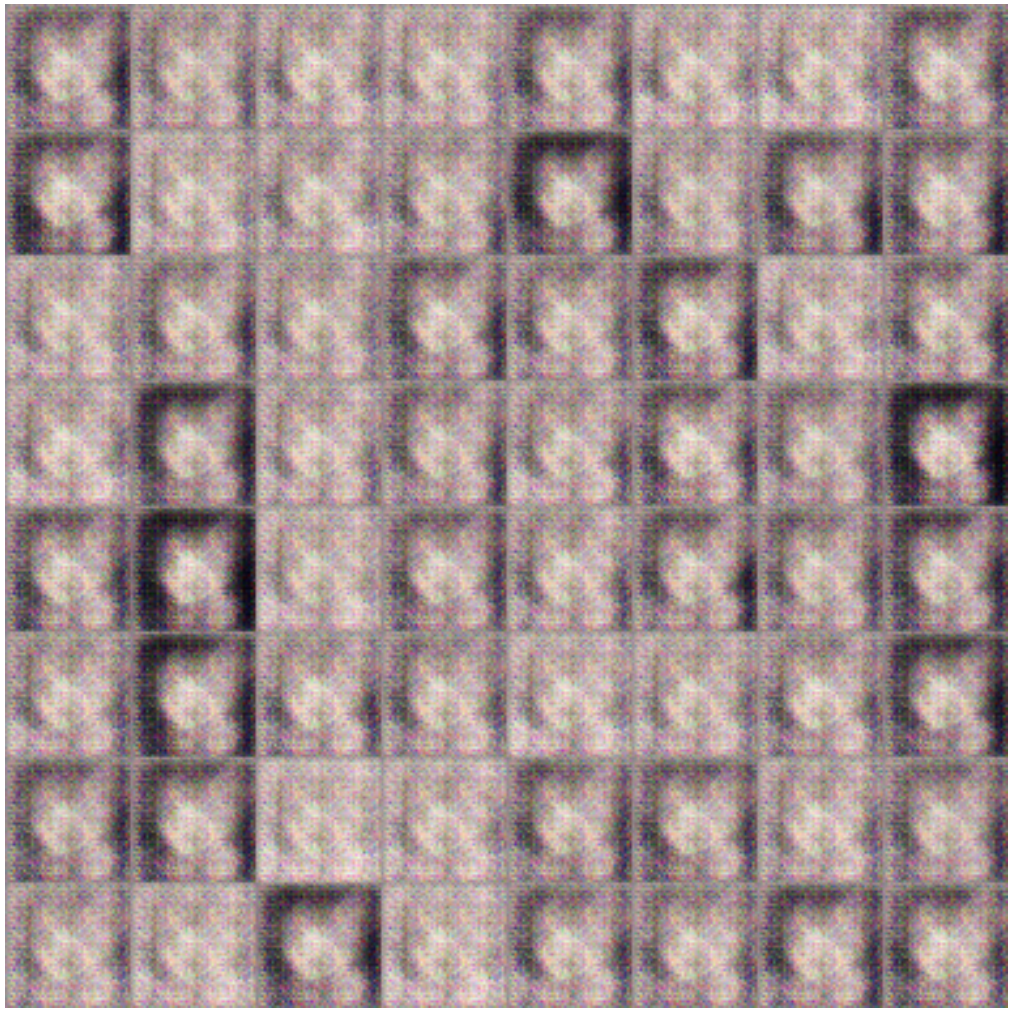
Learning D

- Sample m examples $\{x^1, x^2, \ldots, x^m\}$ from database
- Sample m noise samples $\{z^1, z^2, \ldots, z^m\}$ from a distribution
- Obtaining generated data $\{\tilde{x}^1, \tilde{x}^2, \ldots, \tilde{x}^m\}, \tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_d to maximize
  - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} log\left(1 - D(\tilde{x}^i)\right)$
  - $\theta_d \leftarrow \theta_d + \eta \nabla \tilde{V}(\theta_d)$

Learning G

- Sample m noise samples $\{z^1, z^2, \ldots, z^m\}$ from a distribution
- Update generator parameters θ_g to maximize
  - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} log\left(D\left(G(z^i)\right)\right)$
  - $\theta_g \leftarrow \theta_g - \eta \nabla \tilde{V}(\theta_g)$

Anime Face Generation



100 updates

Source of training data: https://zhuanlan.zhihu.com/p/24767059

Anime Face Generation

1000 updates

Anime Face Generation

5000 updates

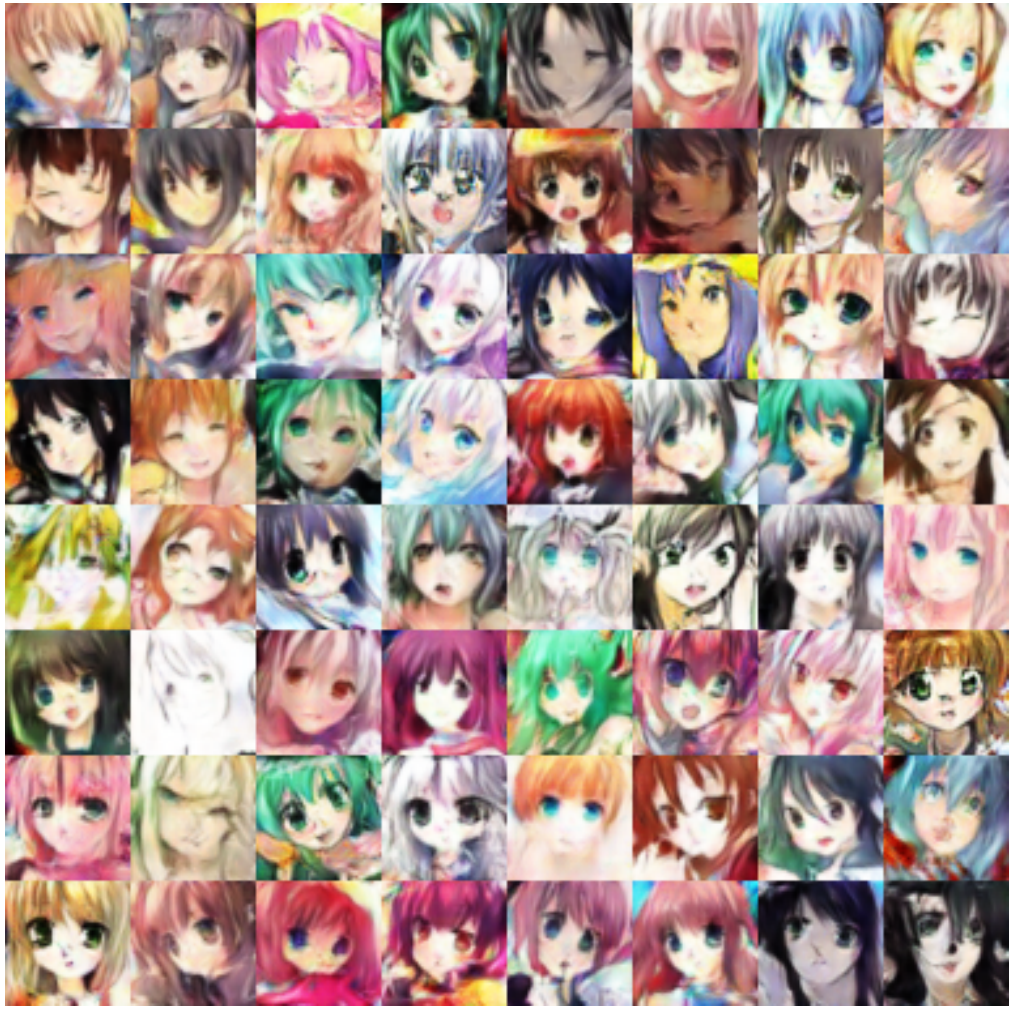
Anime Face Generation

10,000 updates

Anime Face Generation

20,000 updates

Anime Face Generation



50,000 updates

GAN as structured learning

Structured Learning

Machine learning is to find a function f

$$f : X \rightarrow Y$$

Regression: output a scalar

Classification: output a "class" (one-hot vector)

Class 1 Class 2 Class 3

Structured Learning/Prediction: output a sequence, a matrix, a graph, a tree ……

Output is composed of components with dependency

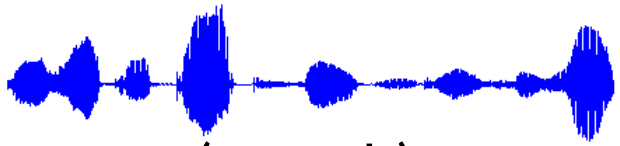
Output Sequence $f : X \rightarrow Y$

Machine Translation

X : “機器學習及其深層與結構化”
(sentence of language 1)

Y : “Machine learning and having it deep and structured”
(sentence of language 2)

Speech Recognition

X : 
(speech)

Y : 感謝大家來上課”
(transcription)

Chat-bot

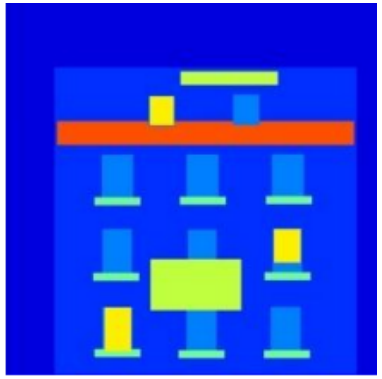
X : “How are you?”
(what a user says)

Y : “I’m fine.”
(response of machine)

Output Matrix

$$f : X \rightarrow Y$$

Image to Image

X : 

Y : 

Colorization:

Ref: https://arxiv.org/pdf/1611.07004v1.pdf

Text to Image

X : "this white and yellow flower have thin white petals and a round yellow stamen"

Y : 

ref: https://arxiv.org/pdf/1605.05396.pdf

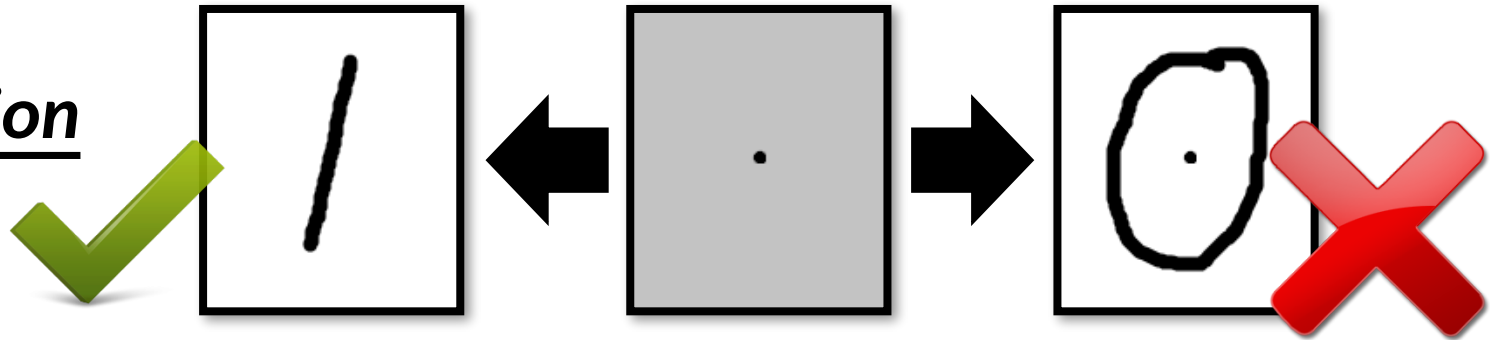
Why Structured Learning Challenging?

- **One-shot/Zero-shot Learning**:
  - In classification, each class has some examples.
  - In structured learning,
    - If you consider each possible output as a "class" ……
    - Since the output space is huge, most "classes" do not have any training data.
    - Machine has to create new stuff during testing.
    - Need more intelligence

Why Structured Learning Challenging?

- Machine has to learn to do ***planning***
  - Machine generates objects component-by-component, but it should have a big picture in its mind.
  - Because the output components have dependency, they should be considered globally.

Image Generation



Structured Learning Approach

Generator

Learn to generate the object at the component level

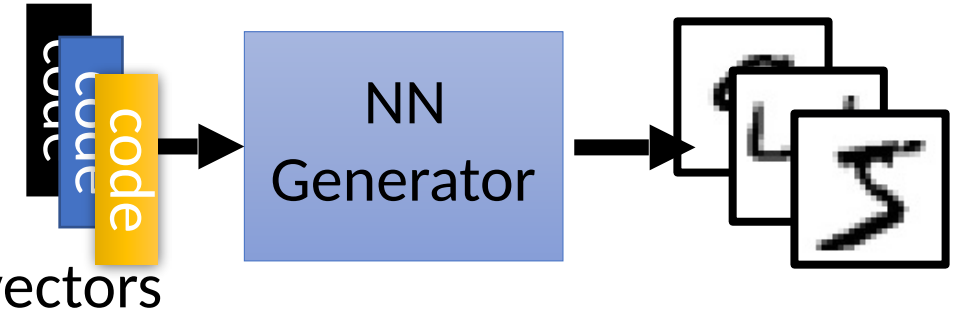
Discriminator

Evaluating the whole object, and find the best one

POLab

Can generator learn by itself?

Generator

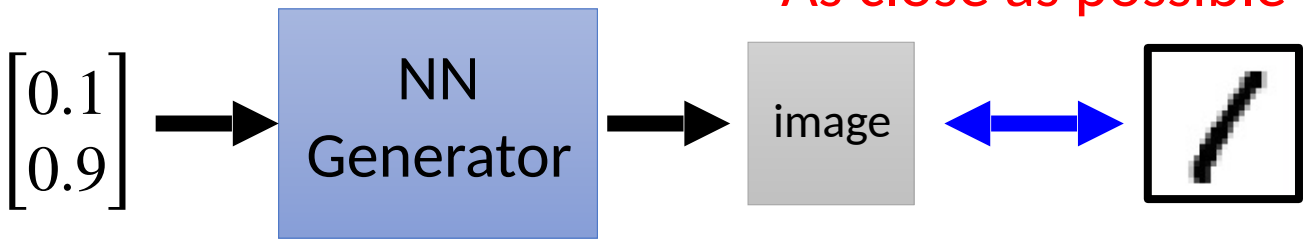


code: $\begin{bmatrix} 0.1 \\ -0.5 \end{bmatrix}$ $\begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}$ $\begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$ $\begin{bmatrix} 0.3 \\ 0.2 \end{bmatrix}$

(where does they come from?)

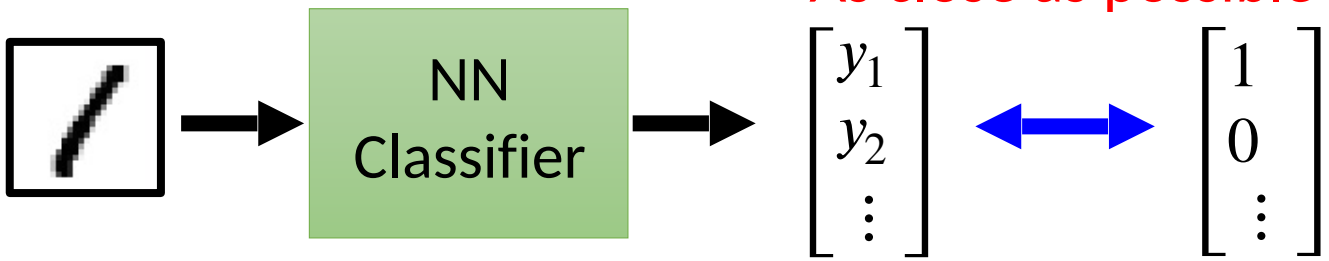
Image:

As close as possible

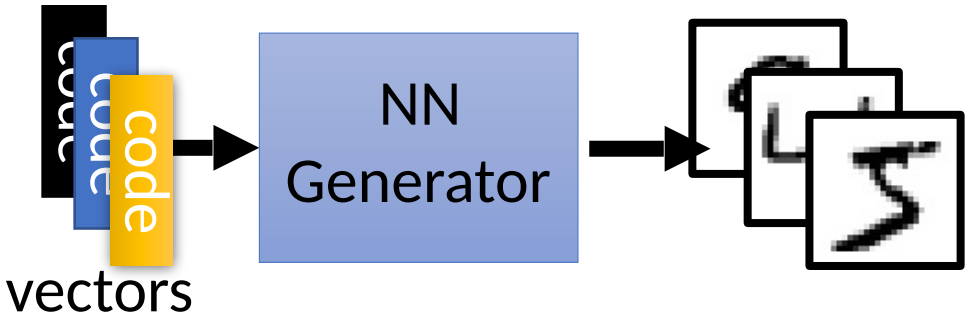


As close as possible

c.f.



Generator

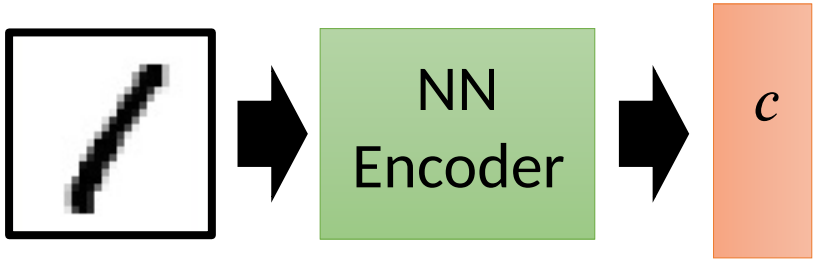


code:
(where does they come from?)

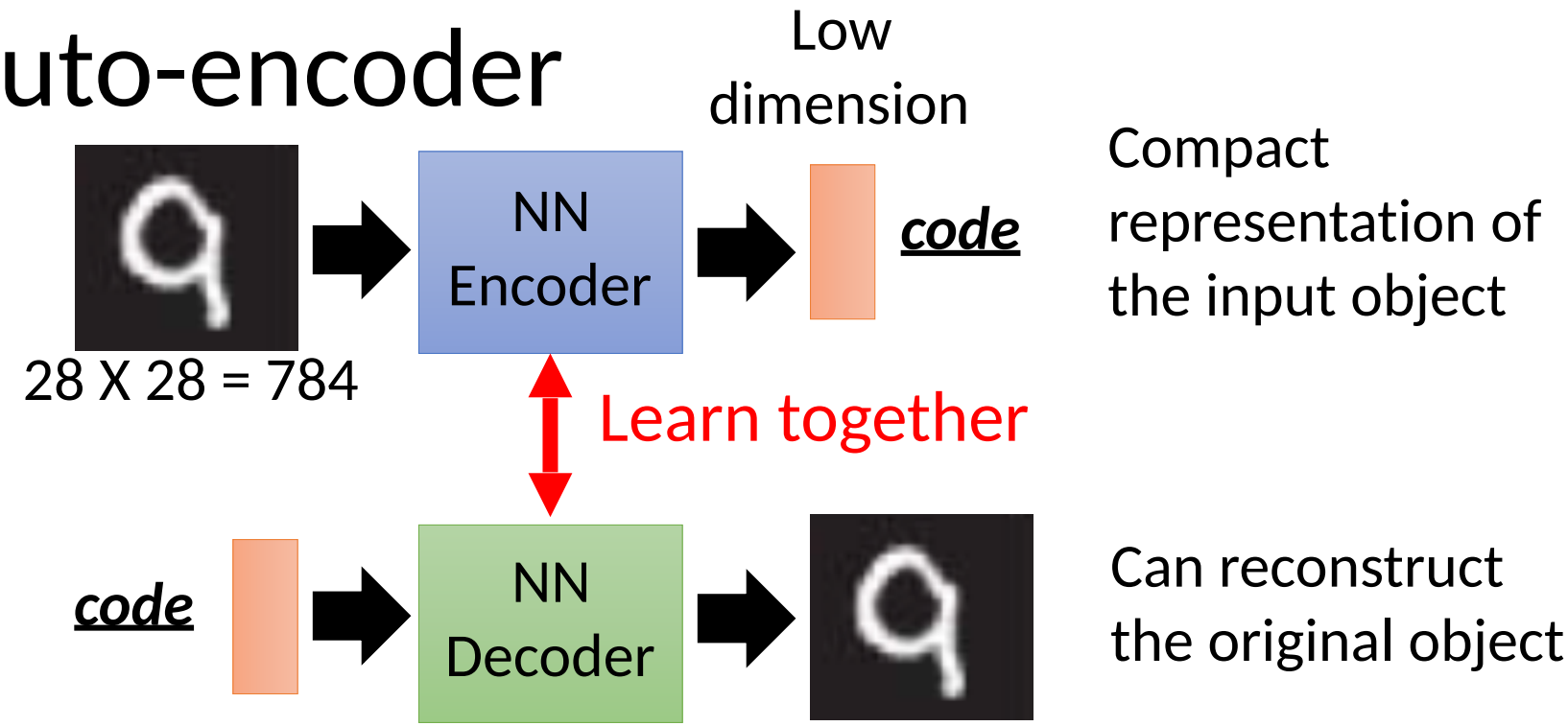
$$\begin{bmatrix} 0.1 \\ -0.5 \end{bmatrix} \quad \begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix} \quad \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} \quad \begin{bmatrix} 0.3 \\ 0.2 \end{bmatrix}$$

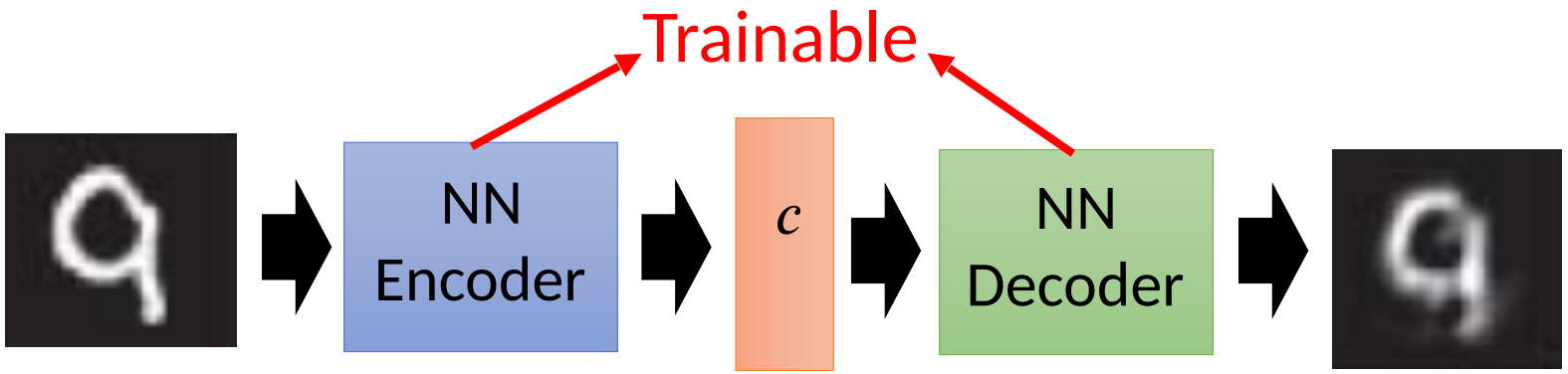
Image:

Encoder in auto-encoder provides the code ☺

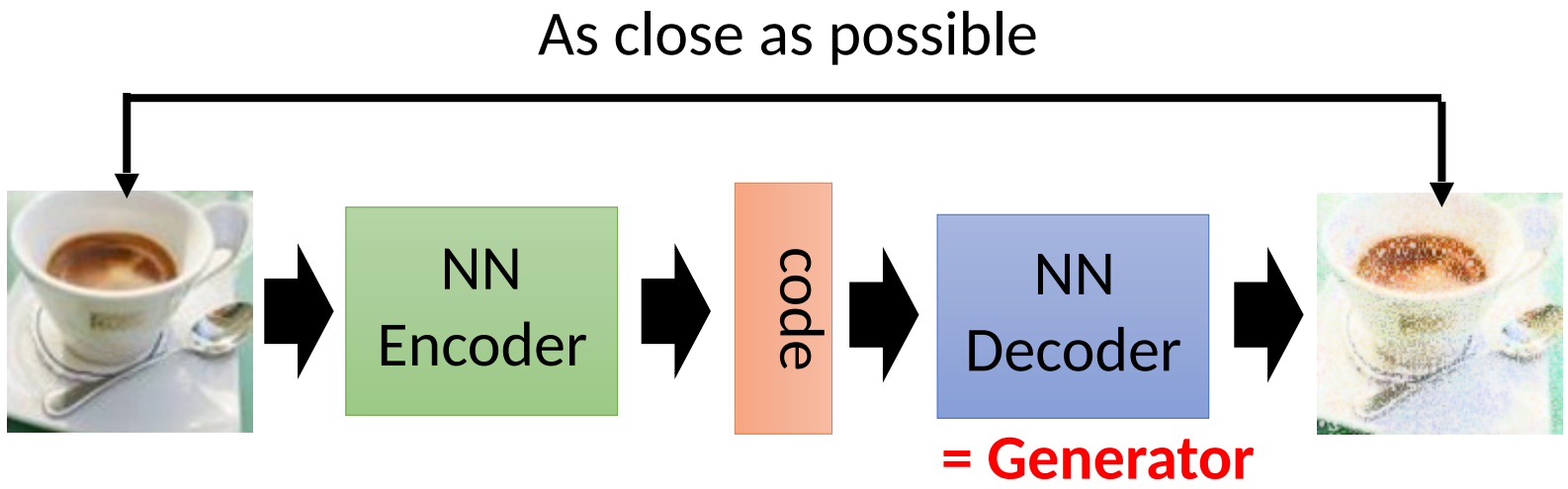


Auto-encoder

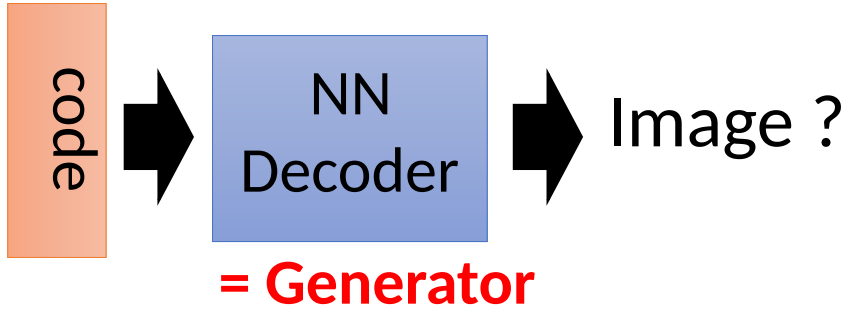




Auto-encoder

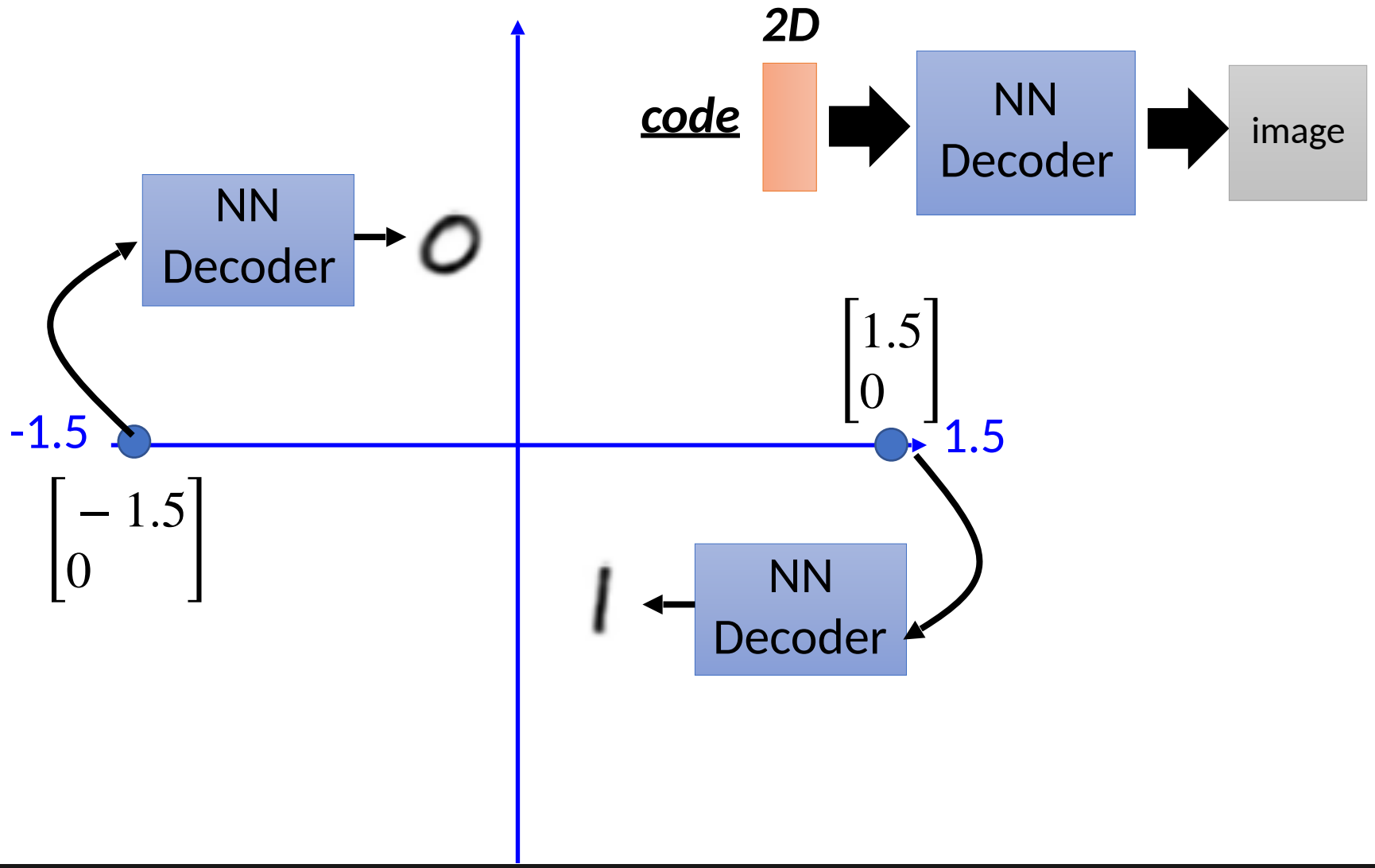


Randomly generate a vector as code



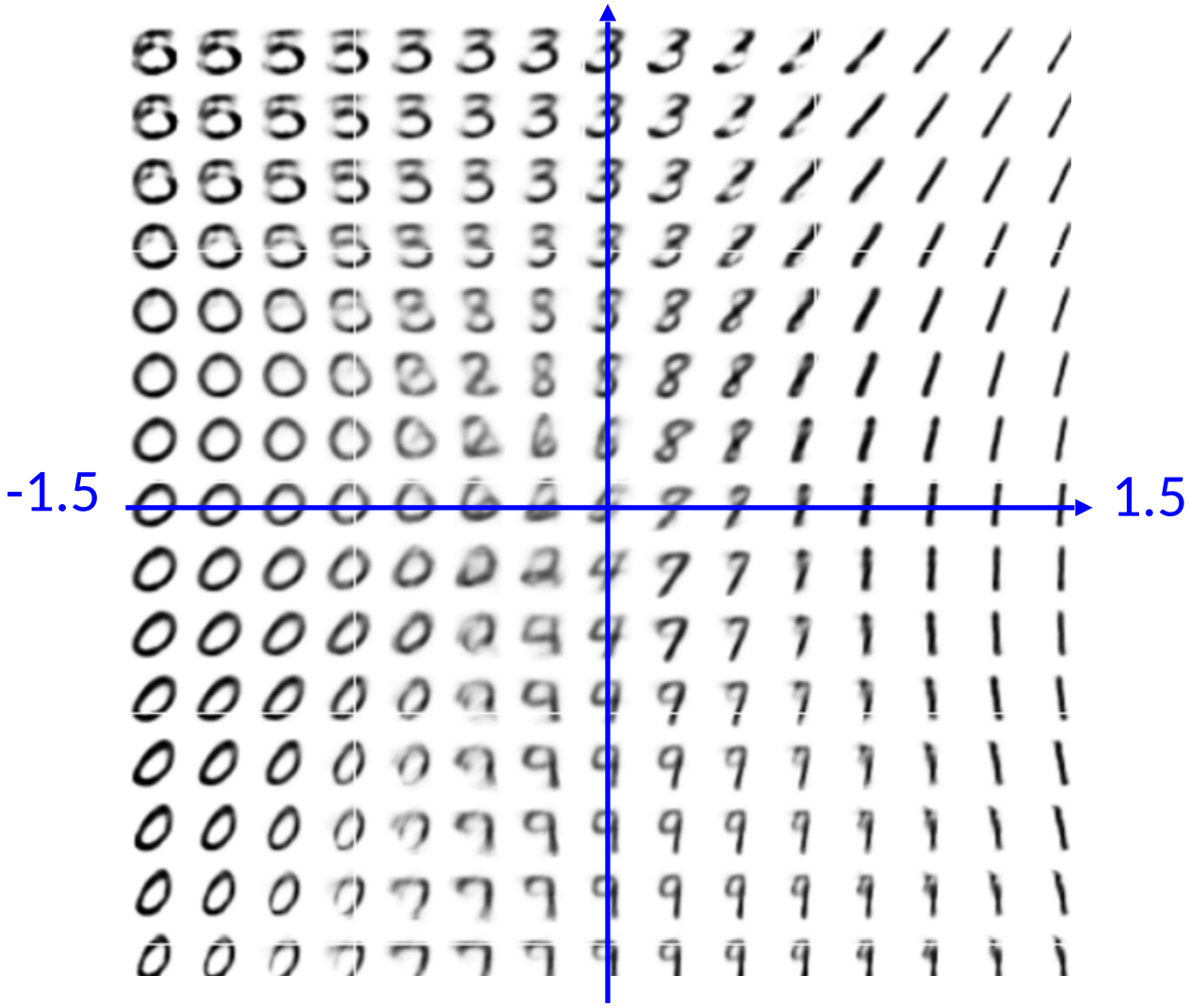
(real examples)

Auto-encoder

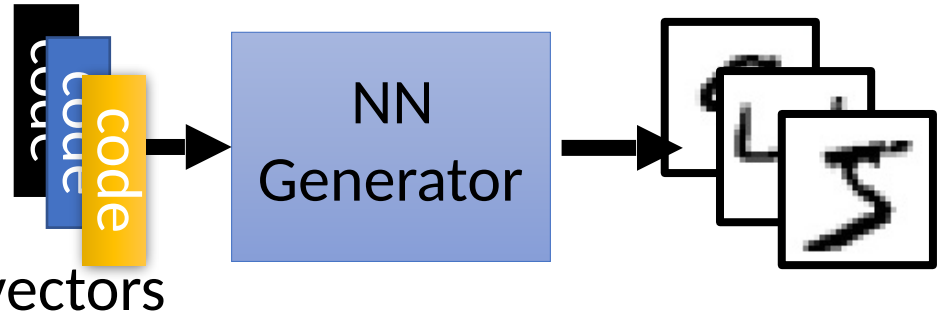


(real examples)

Auto-encoder



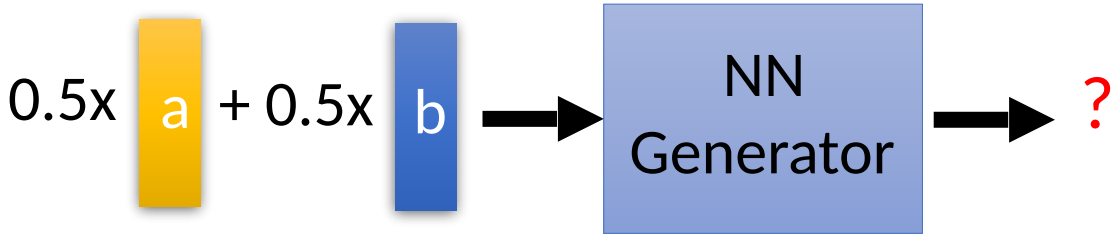
Auto-encoder



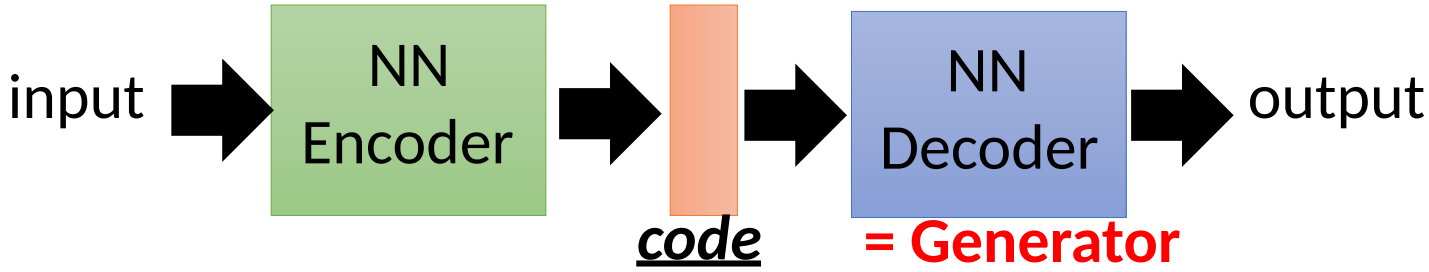
code: $\begin{bmatrix} 0.1 \\ -0.5 \end{bmatrix}$ $\begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}$ $\begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$ $\begin{bmatrix} 0.3 \\ 0.2 \end{bmatrix}$

(where does them come from?)

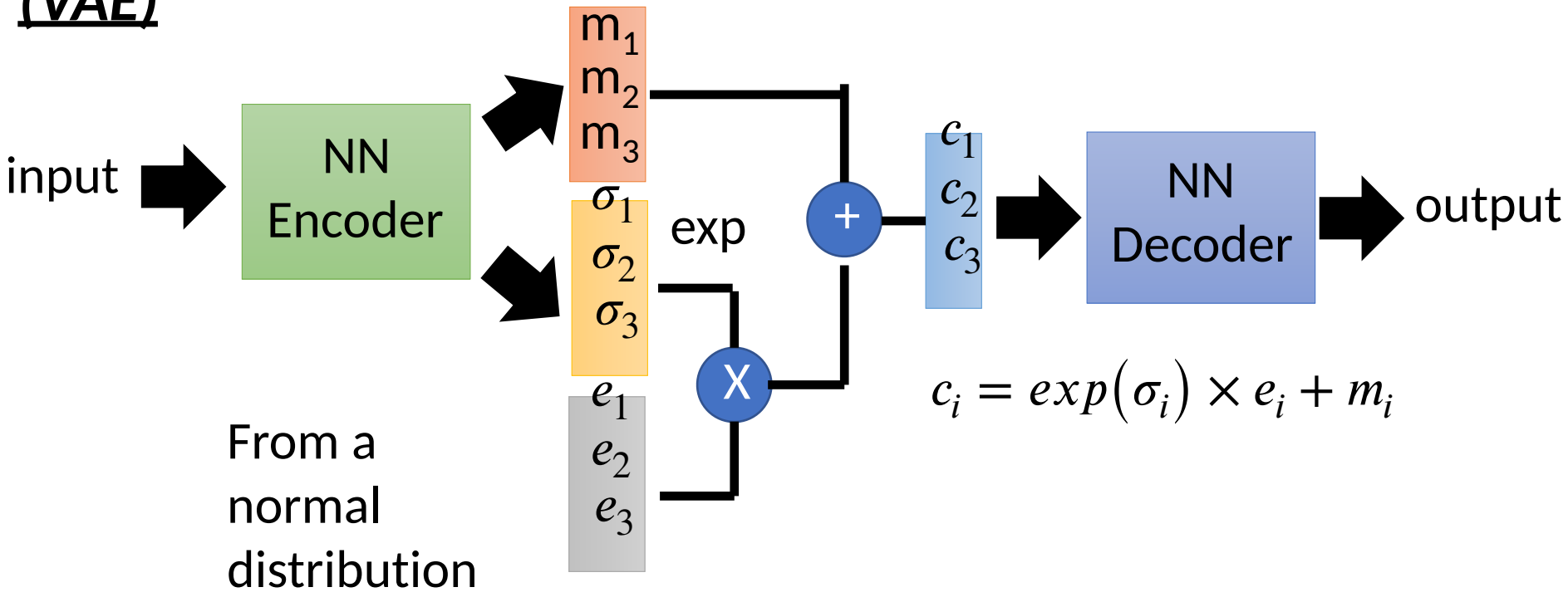
Image:



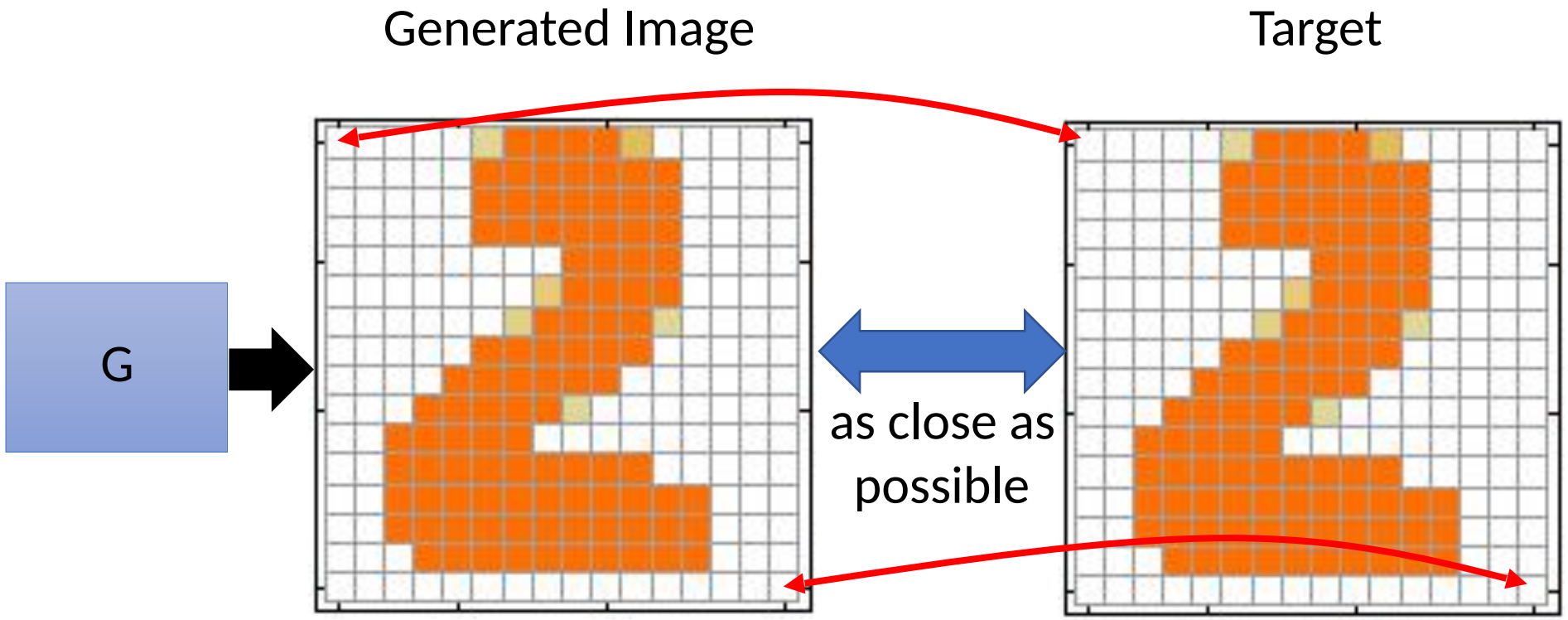
Auto-encoder



Variational Auto-encoder (VAE)



What do we miss?



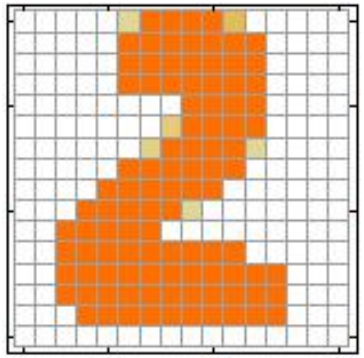
It will be fine if the generator can truly copy the target image.

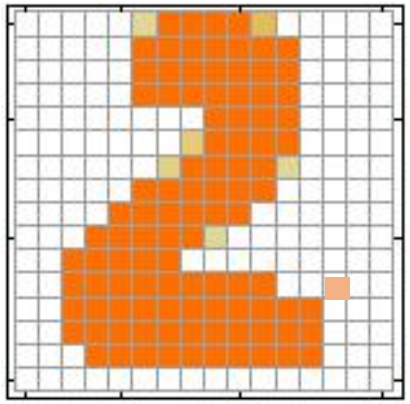
What if the generator makes some mistakes .....

Some mistakes are serious, while some are fine.

What do we miss?

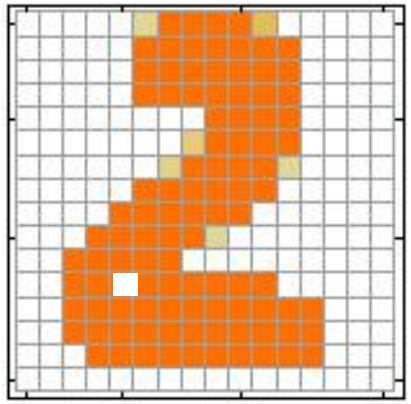
Target





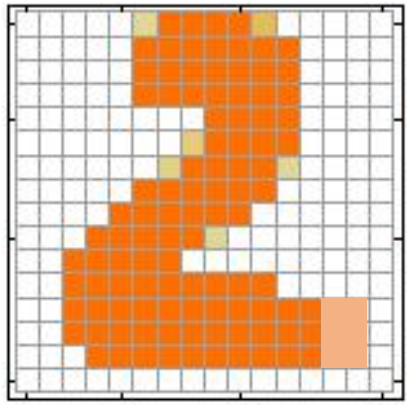
1 pixel error

我覺得不行



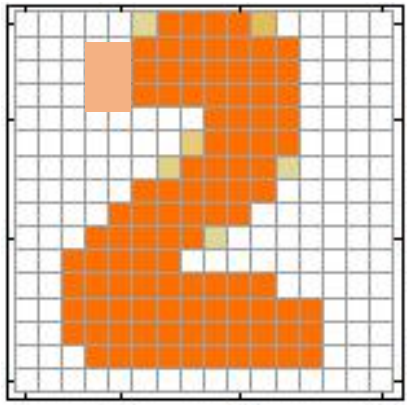
1 pixel error

我覺得不行



6 pixel errors

我覺得其實可以

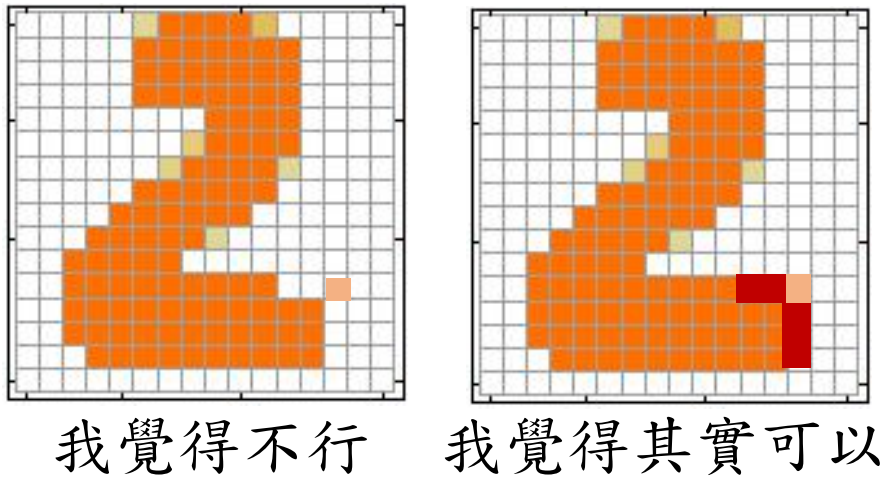


6 pixel errors

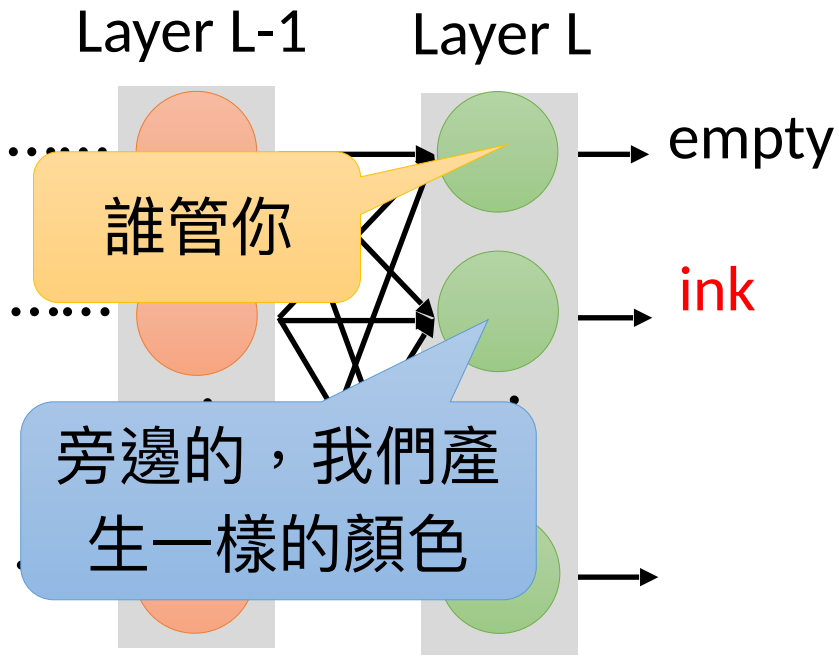
我覺得其實可以

What do we miss?

Each neural in output layer corresponds to a pixel.



我覺得不行　我覺得其實可以

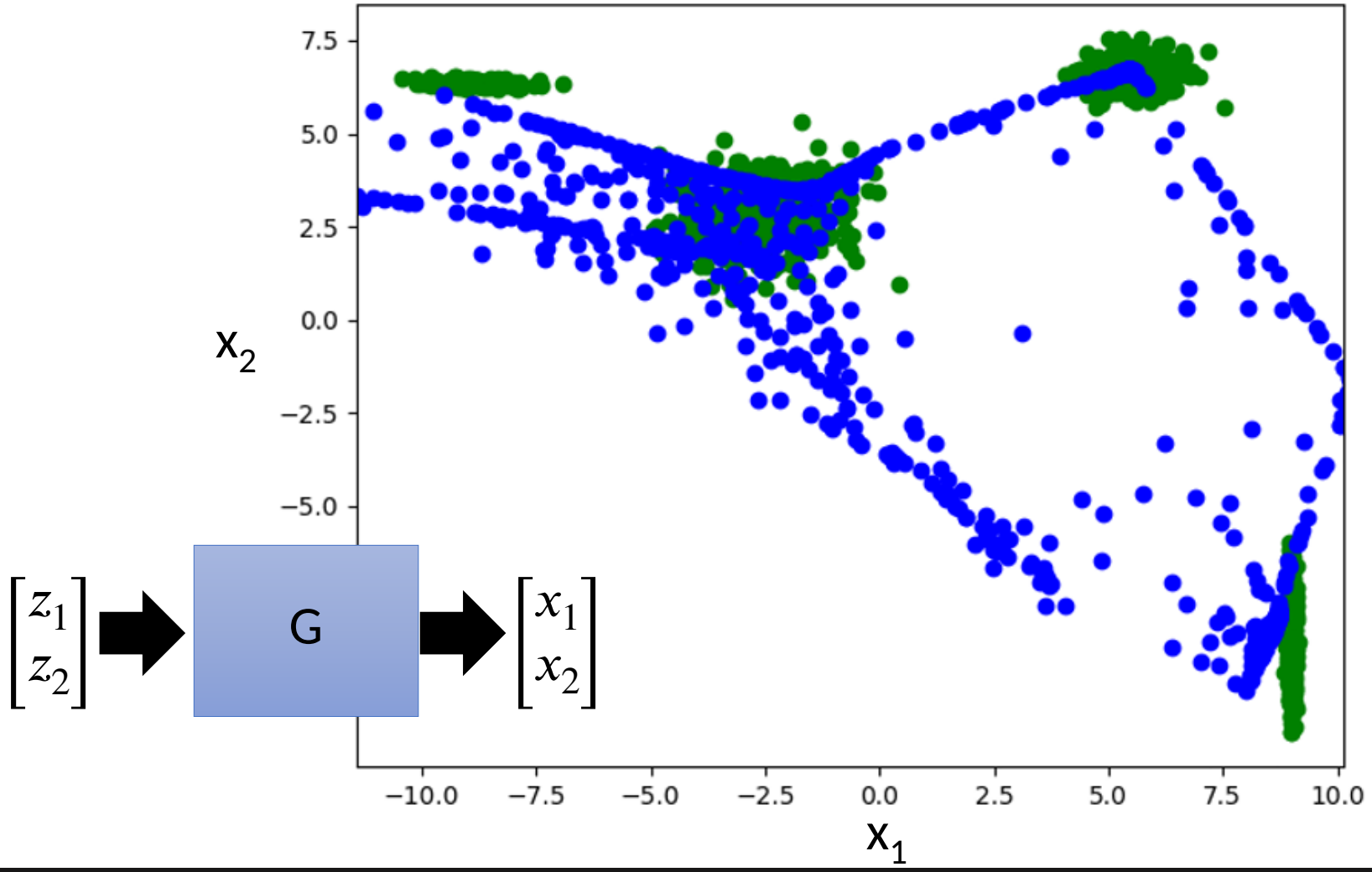


The relation between the components are critical.

Although highly correlated, they cannot influence each other.

Need deep structure to catch the relation between components.

(Variational) Auto-encoder



POLab

Can discriminator generate?

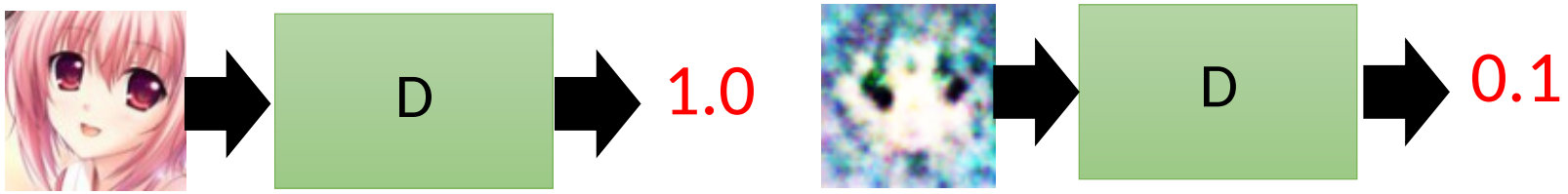
Discriminator

Evaluation function, Potential Function, Energy Function ...

- Discriminator is a function D (network, can deep)

$$\mathrm{D} : X \rightarrow \mathrm{R}$$

  - Input x: an object x (e.g. an image)
  - Output D(x): scalar which represents how "good" an object x is

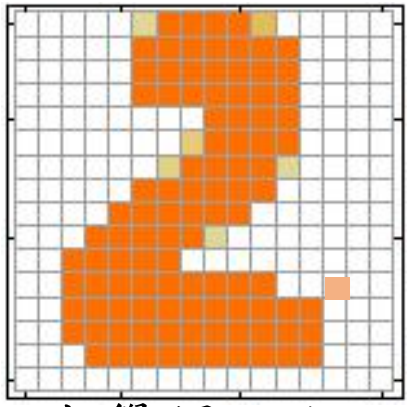


Can we use the discriminator to generate objects?

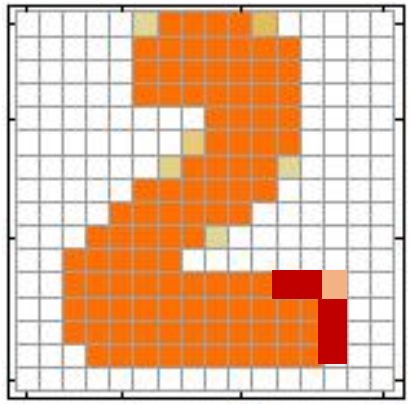
Yes.

Discriminator

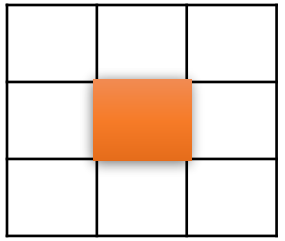
- It is easier to catch the relation between the components by top-down evaluation.



我覺得不行



我覺得其實 OK



This CNN filter is good enough.

Discriminator

- Suppose we already have a good discriminator D(x)

  …

Inference

- Generate object $\tilde{x}$ that

$$\tilde{x} = \arg\max_{x \in X} D(x)$$

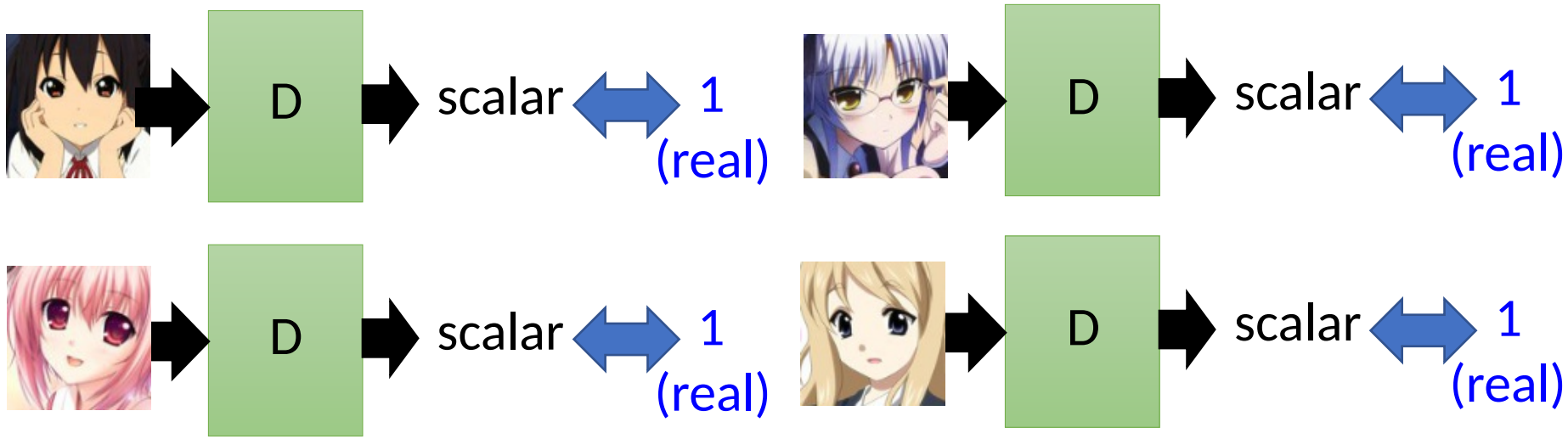
Enumerate all possible x !!!

It is feasible ???

How to learn the discriminator?

Discriminator - Training

- I have some real images

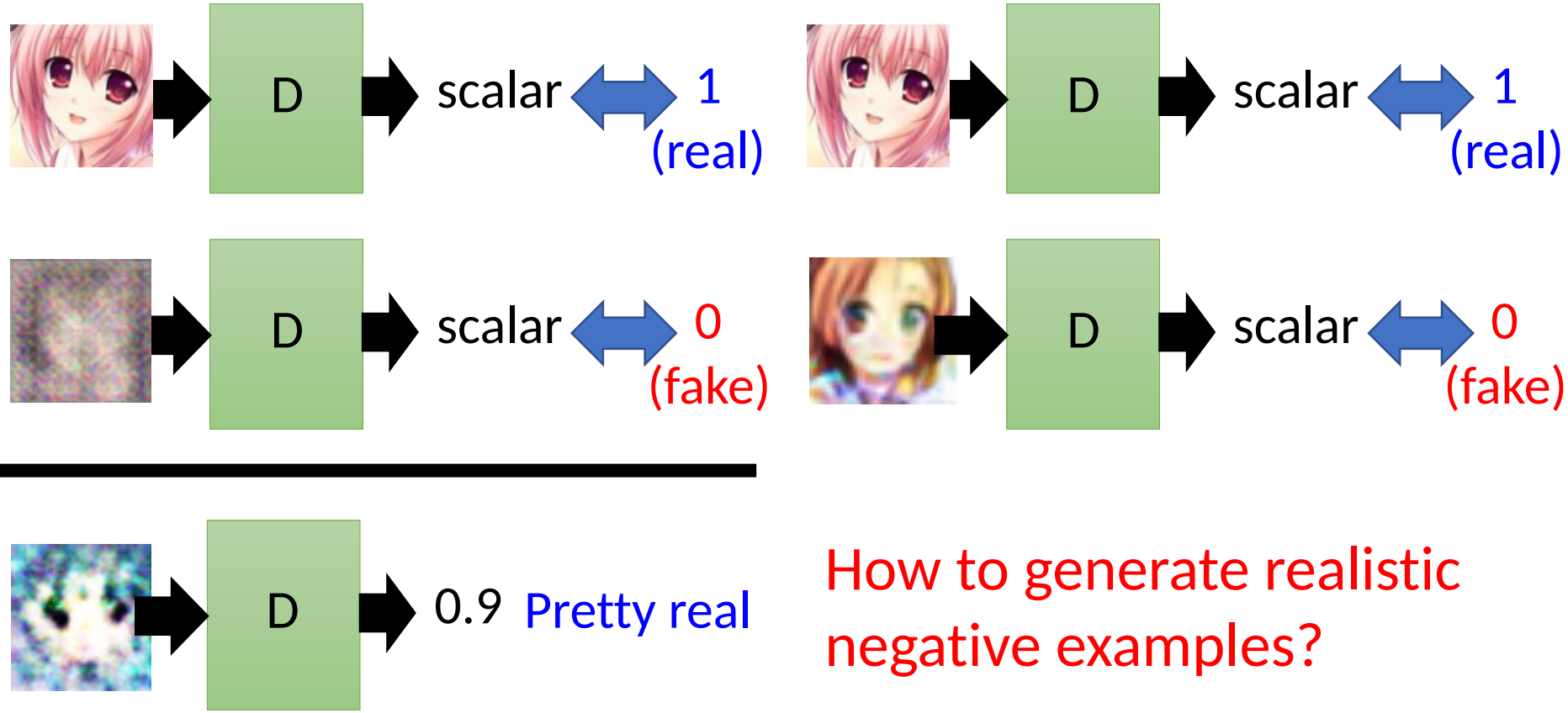


Discriminator only learns to output “1” (real).

Discriminator training needs some negative examples.

Discriminator - Training

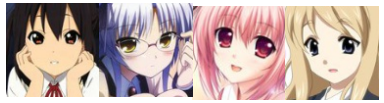
- Negative examples are critical.



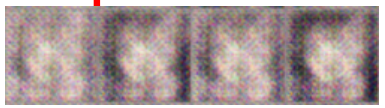
How to generate realistic negative examples?

Discriminator - Training

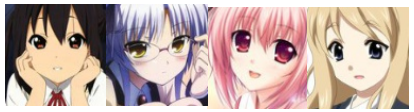
- **General Algorithm**



  - Given a set of positive examples, randomly generate a set of negative examples.



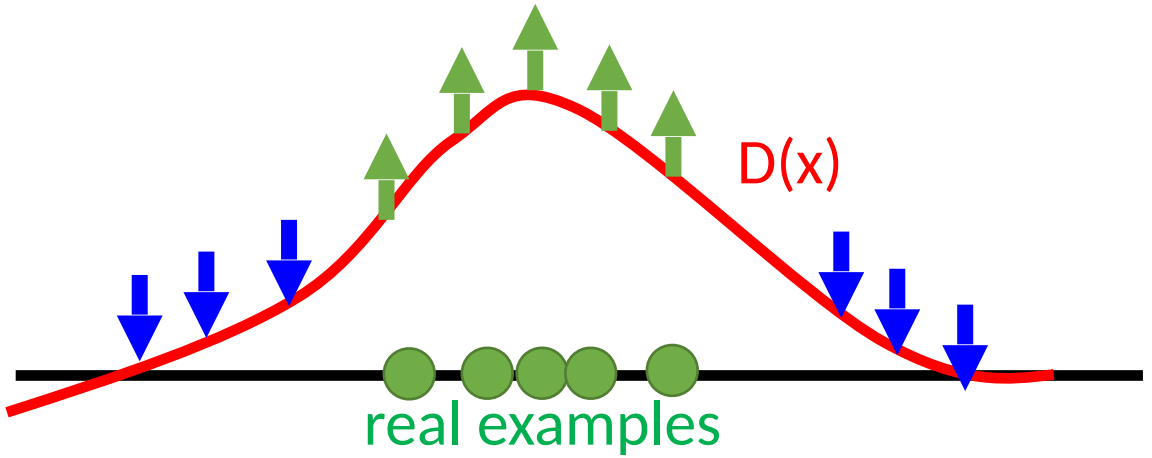
  - In each iteration
    - Learn a discriminator D that can discriminate positive and negative examples.



    - Generate negative examples by discriminator D

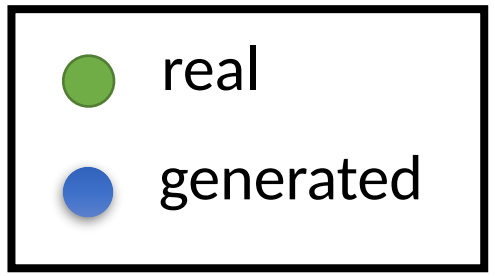
$$\tilde{x} = \arg\max_{x \in X} D(x)$$

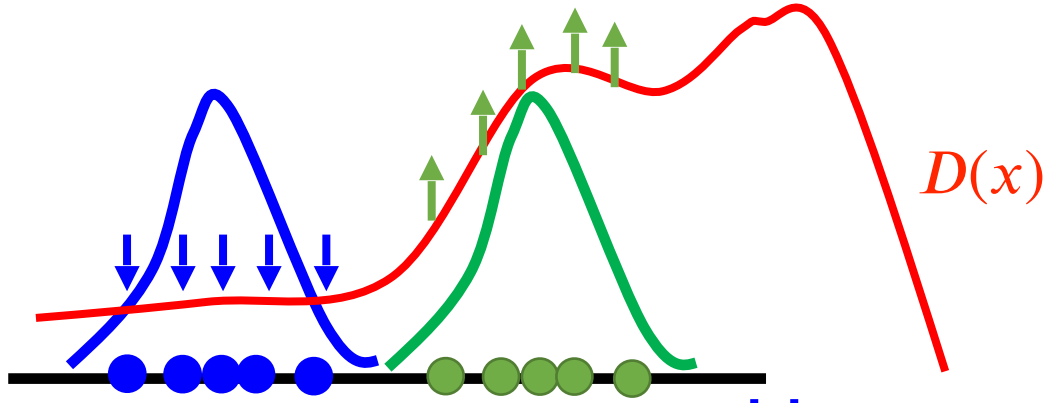
Discriminator - Training

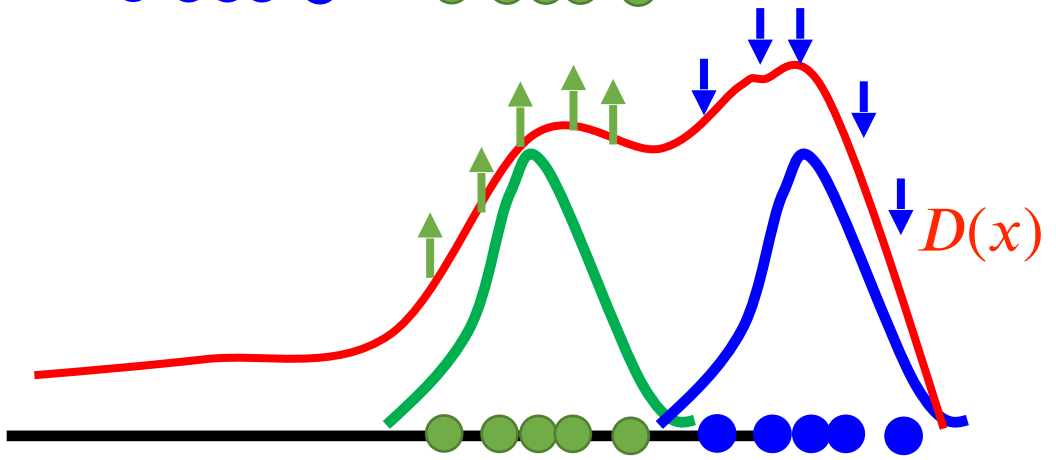


In practice, you cannot decrease all the x other than real examples.

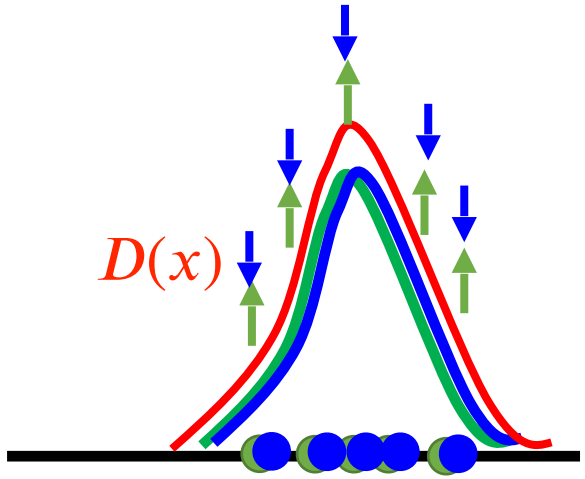
Discriminator - Training

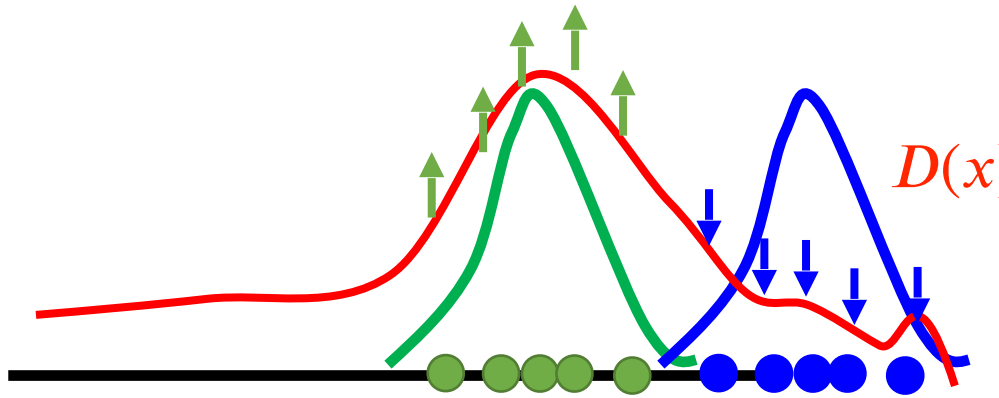






In the end ……





Generator v.s. Discriminator

- ***Generator***

- Pros:
  - Easy to generate even with deep model
- Cons:
  - Imitate the appearance
  - Hard to learn the correlation between components

- ***Discriminator***

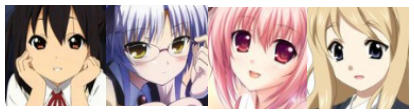
- Pros:
  - Considering the big picture
- Cons:
  - Generation is not always feasible
    - Especially when your model is deep
  - How to do negative sampling?

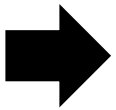
Generator + Discriminator

- **General Algorithm**

  - Given a set of positive examples, randomly generate a set of negative examples.

  - In each iteration
    - Learn a discriminator D that can discriminate positive and negative examples.


v.s.

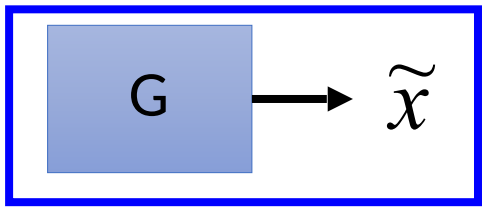

D

    - Generate negative examples by discriminator D

G $\rightarrow \tilde{x}$ = $\tilde{x} = \arg\max_{x \in X} D(x)$

Benefit of GAN

- From Discriminator's point of view
  - Using generator to generate negative samples



$$=\quad \tilde{x} = \arg\max_{x \in X} D(x)$$

efficient

- From Generator's point of view
  - Still generate the object component-by-component
  - But it is learned from the discriminator with global view.

Conditional Gan

Text-to-Image

a dog is running 

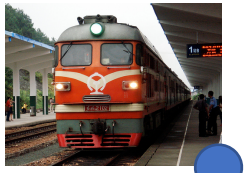
a bird is flying 

- **Traditional supervised approach**

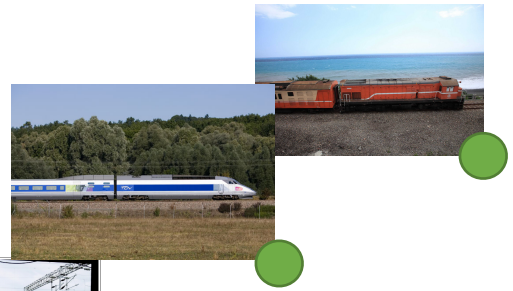
c^1: a dog is running → NN → Image ↔ 

as close as possible

Text: “train”



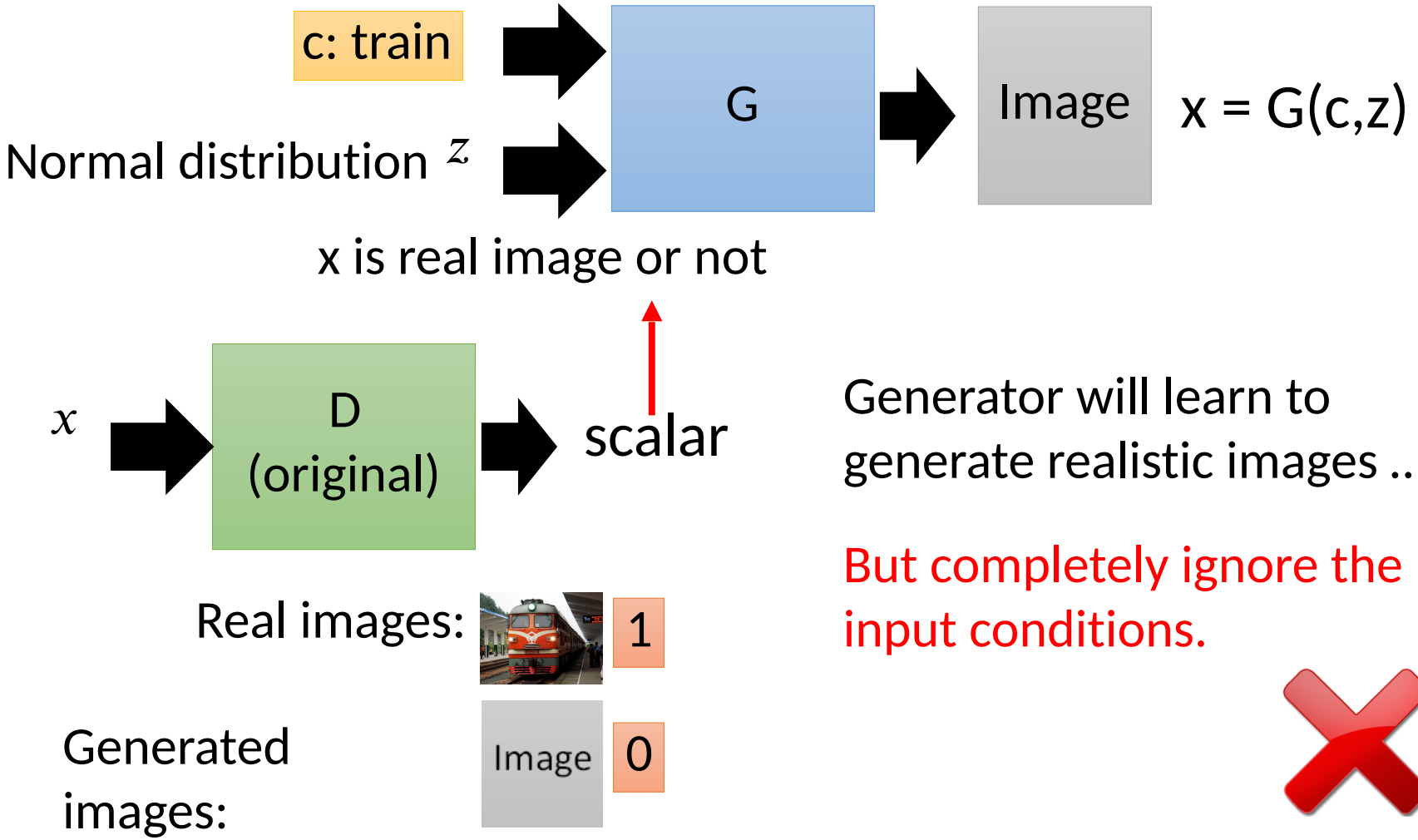
Target of NN output



A blurry image!

[Scott Reed, et al, ICML, 2016]

Conditional GAN

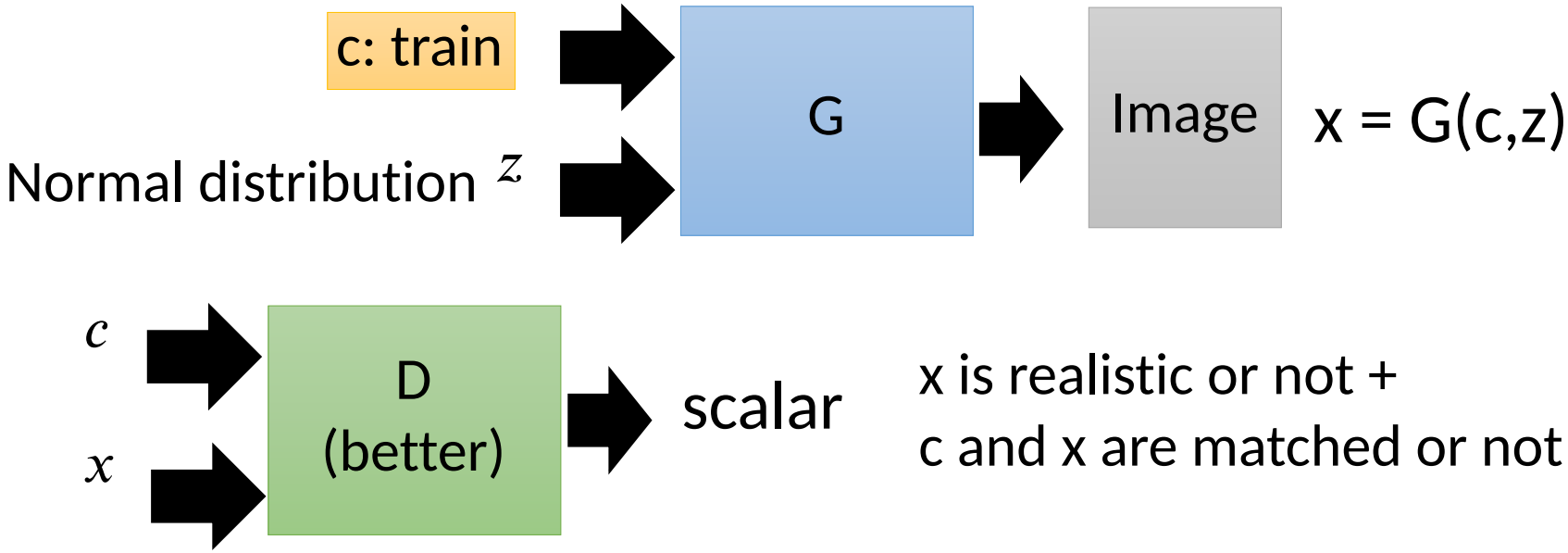


Generator will learn to generate realistic images ....

But completely ignore the input conditions.

[Scott Reed, et al, ICML, 2016]

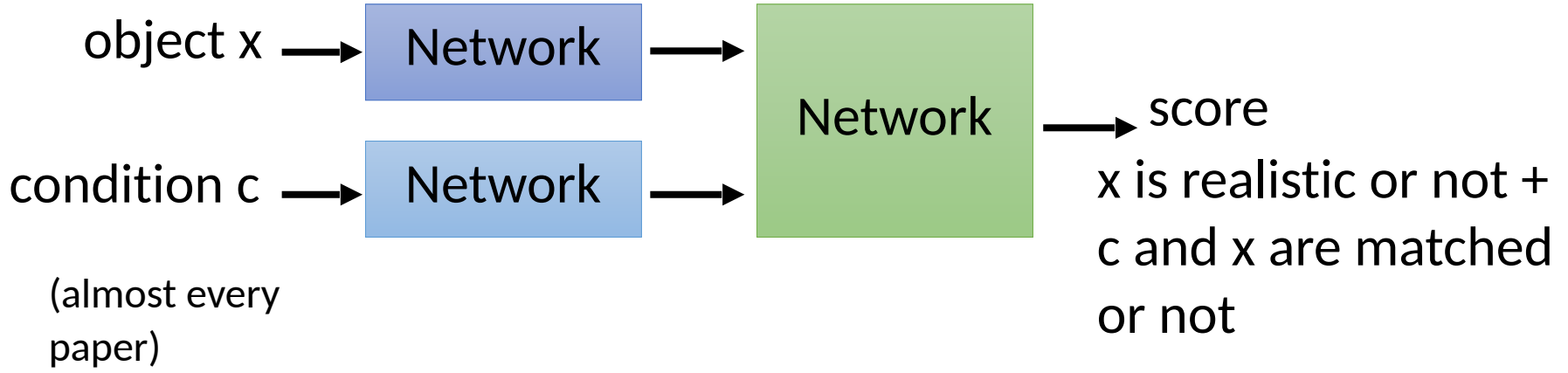
Conditional GAN



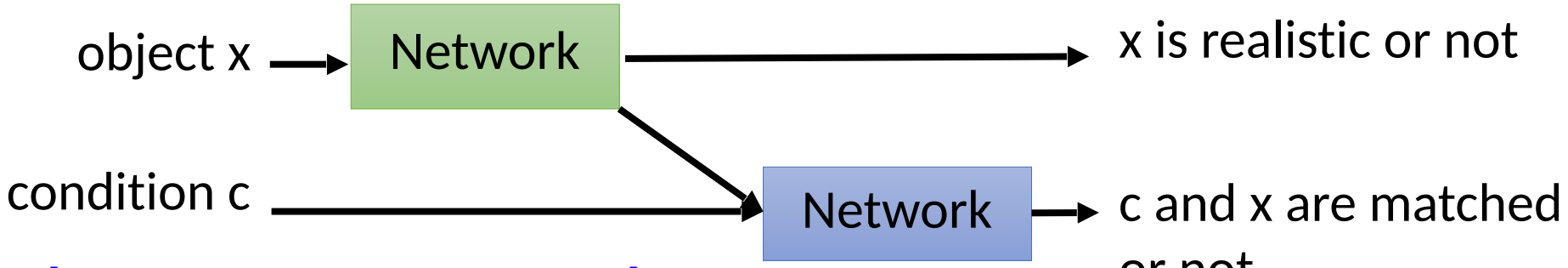
- In each training iteration:
  - Sample m positive examples $\{(c^1, x^1), (c^2, x^2), \dots, (c^m, x^m)\}$ from database
  - Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
  - Obtaining generated data $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(c^i, z^i)$
  - Sample m objects $\{\hat{x}^1, \hat{x}^2, \dots, \hat{x}^m\}$ from database
  - Update discriminator parameters θ_d to maximize
    - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} logD(c^i, x^i)$

      $+\frac{1}{m}\sum_{i=1}^{m} log\left(1 - D(c^i, \tilde{x}^i)\right) + \frac{1}{m}\sum_{i=1}^{m} log\left(1 - D(c^i, \hat{x}^i)\right)$
    - $\theta_d \leftarrow \theta_d + \eta\nabla\tilde{V}(\theta_d)$
  - Sample m noise samples$\{z^1, z^2, \dots, z^m\}$ from a distribution
  - Sample m conditions $\{c^1, c^2, \dots, c^m\}$ from a database
  - Update generator parameters θ_g to maximize
    - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} log\left(D\left(G(c^i, z^i)\right)\right)$, $\theta_g \leftarrow \theta_g - \eta\nabla\tilde{V}(\theta_g)$

Conditional GAN - Discriminator



(almost every paper)



[Augustus Odena et al., ICML, 2017]
[Takeru Miyato, et al., ICLR, 2018]
[Han Zhang, et al., arXiv, 2017]

Conditional GAN

paired data

blue eyes
red hair
short hair

Collecting anime faces and the description of its characteristics

red hair, green eyes



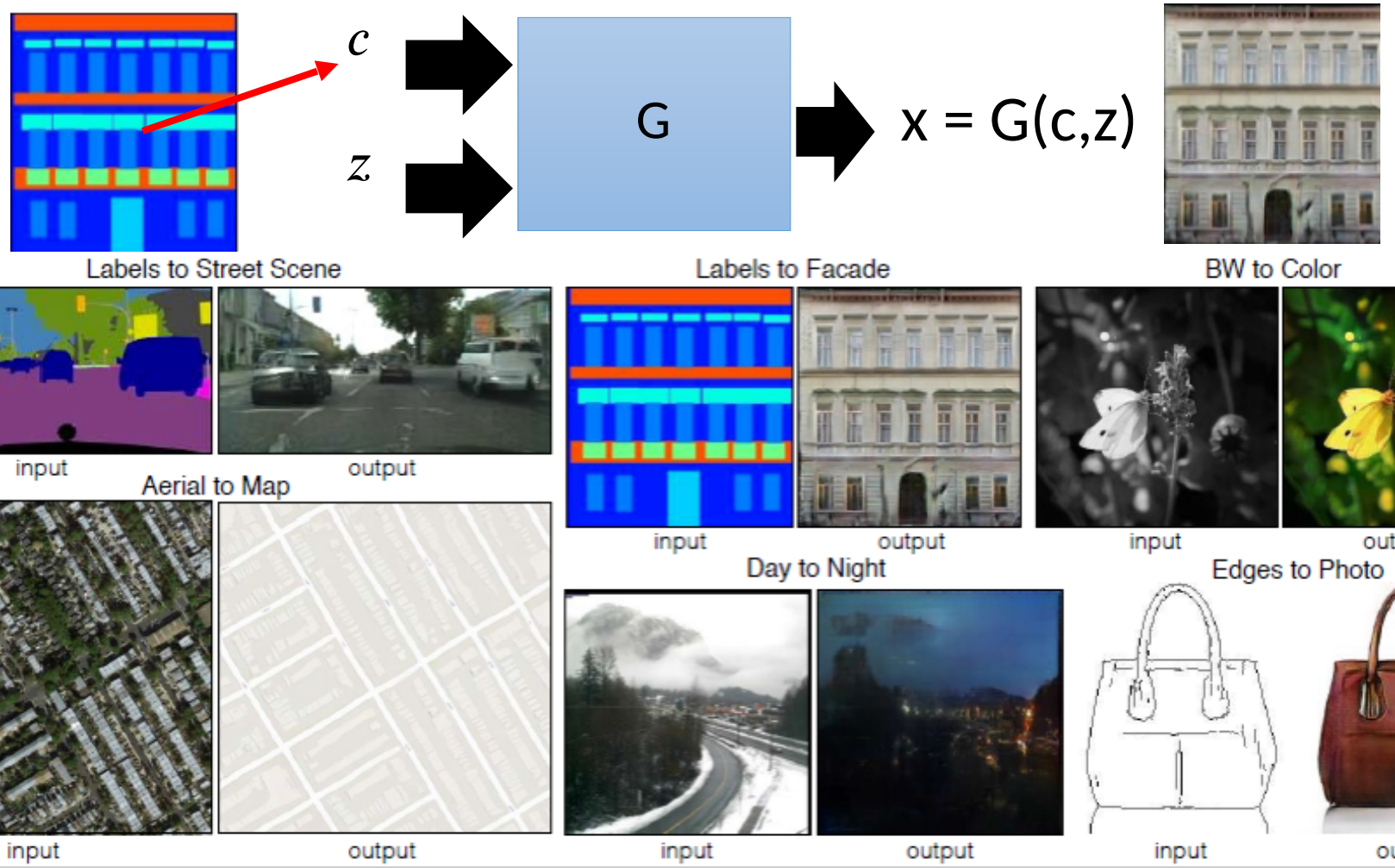


blue hair, red eyes





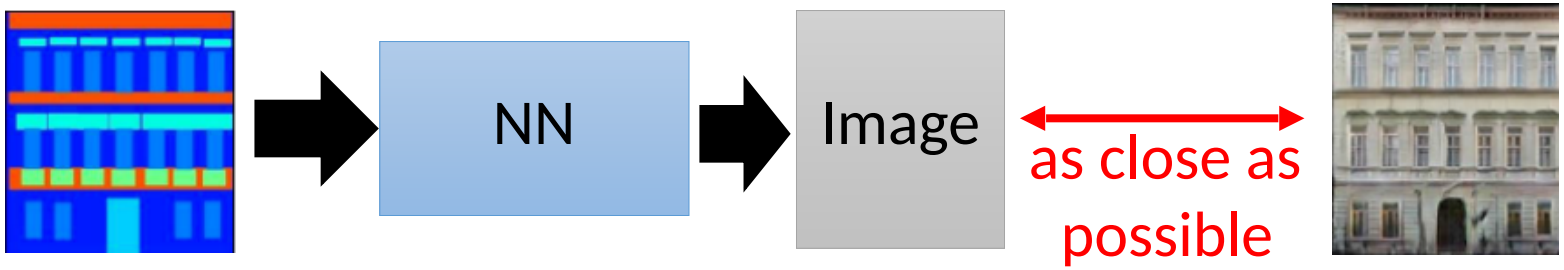
Image-to-image



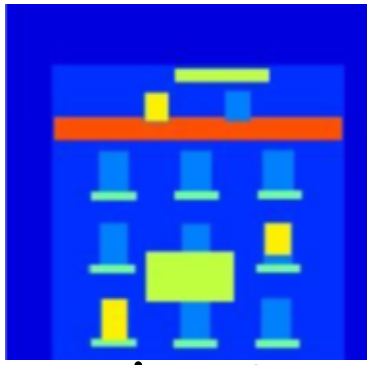
https://arxiv.org/pdf/1611.07004

Image-to-image

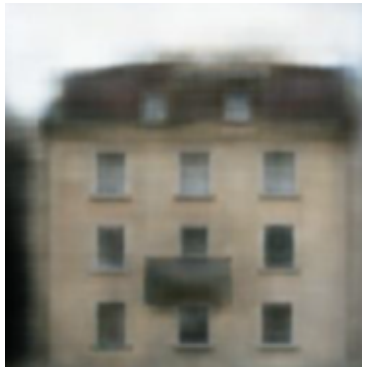
- Traditional supervised approach



Testing:



input

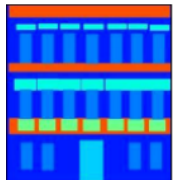


close

It is blurry because it is the average of several images.

Image-to-image

- Experimental results



z

G

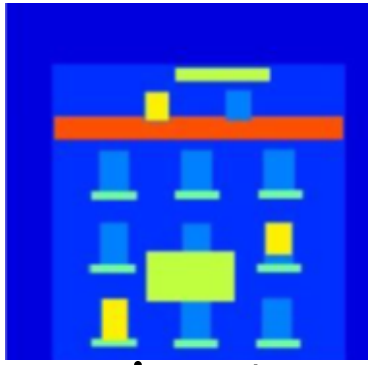
Image

Image

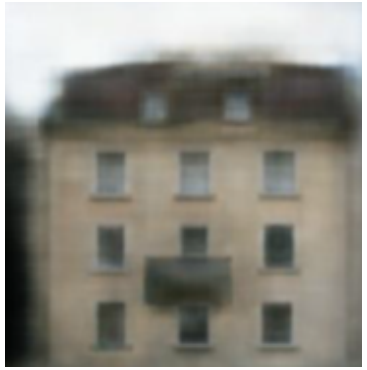
D

scalar

Testing:



input



close

GAN

GAN + close

POLab

Conquer some issues using GANs

- Missing value imputation and GANs
- Class imbalanced dataset and GANs

Missing data imputation

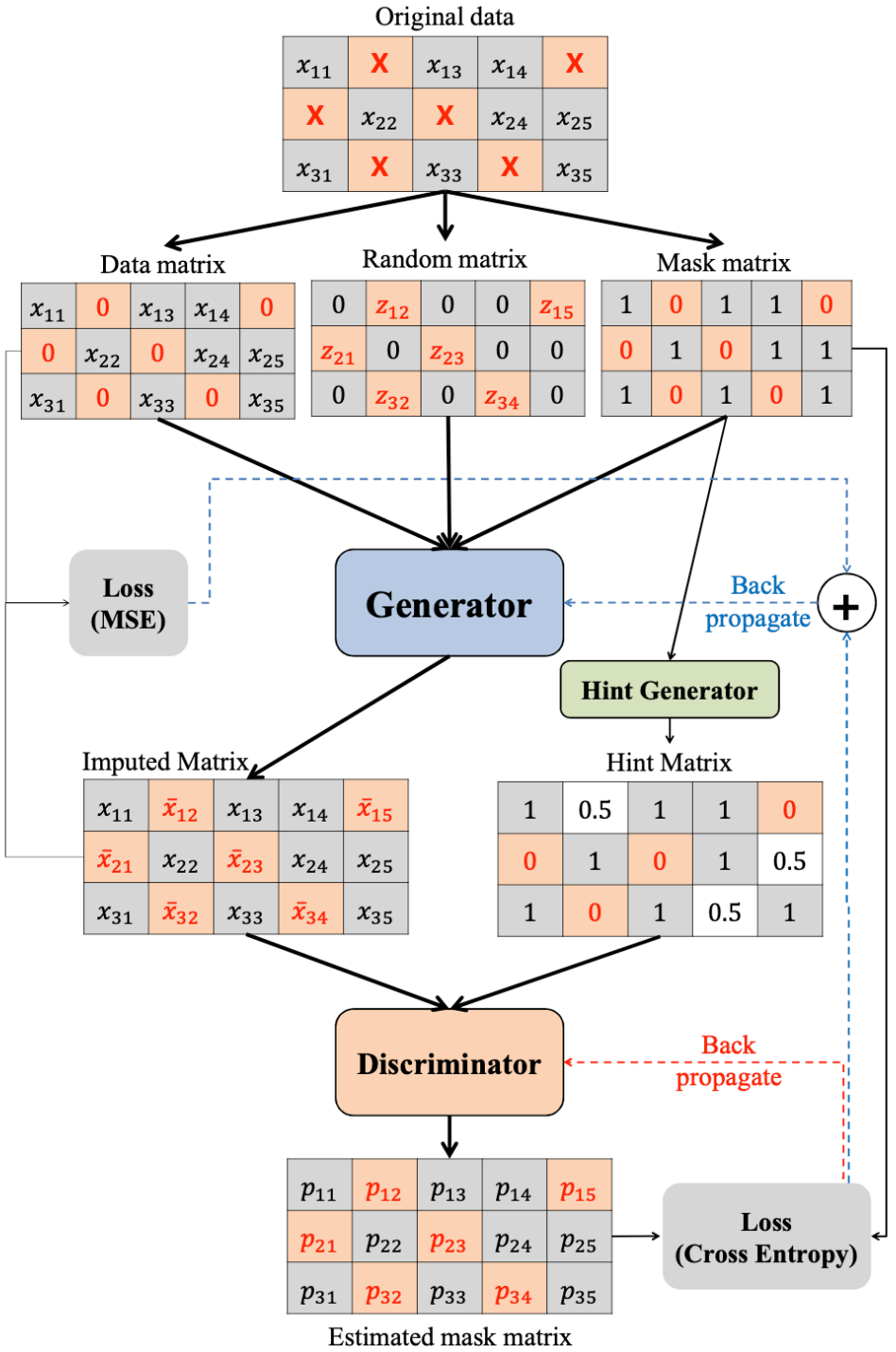
Common Ways

- Drop missing values
- Using mean/median values
- Using most frequent values
- Using k-nn
- Using multivariate imputation by chained equations (MICE)

Based on GAN

- Generative Adversarial Imputation Nets (GAIN)

The architecture of GAIN



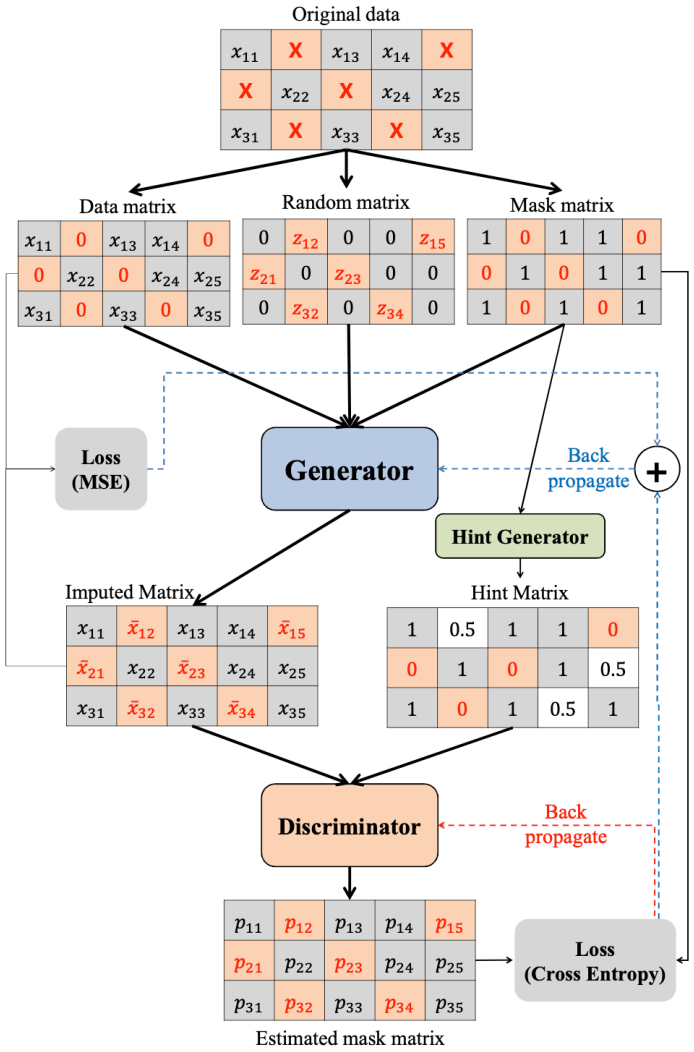
Given d-dimensional space $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_d$

X = (X1 , …, Xd) is a random variable taking values in *X*

M = (M1 , …, Md) ,taking values in {0,1}

$$\tilde{\mathbf{X}} = (\tilde{X}_1, \ldots, \tilde{X}_d) \in \tilde{\mathcal{X}}$$

$$\tilde{X}_i = \begin{cases} X_i, & \text{if } M_i = 1 \\ *, & \text{otherwise} \end{cases}$$

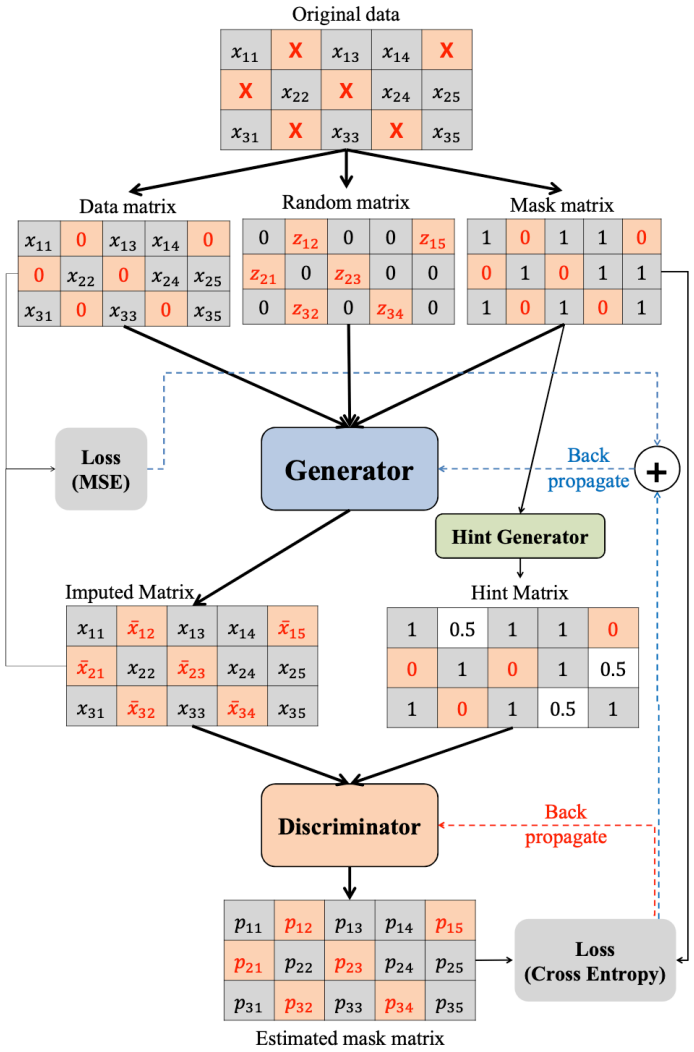


$$\bar{\mathbf{X}} = G(\tilde{\mathbf{X}}, \mathbf{M}, (\mathbf{1} - \mathbf{M}) \odot \mathbf{Z})$$

$$\hat{\mathbf{X}} = \mathbf{M} \odot \tilde{\mathbf{X}} + (\mathbf{1} - \mathbf{M}) \odot \bar{\mathbf{X}}$$

Why Hint Generator?

```
H_mb_temp = binary_sampler(hint_rate, batch_size, dim)
H_mb = M_mb * H_mb_temp
```



Deal with class imbalanced dataset

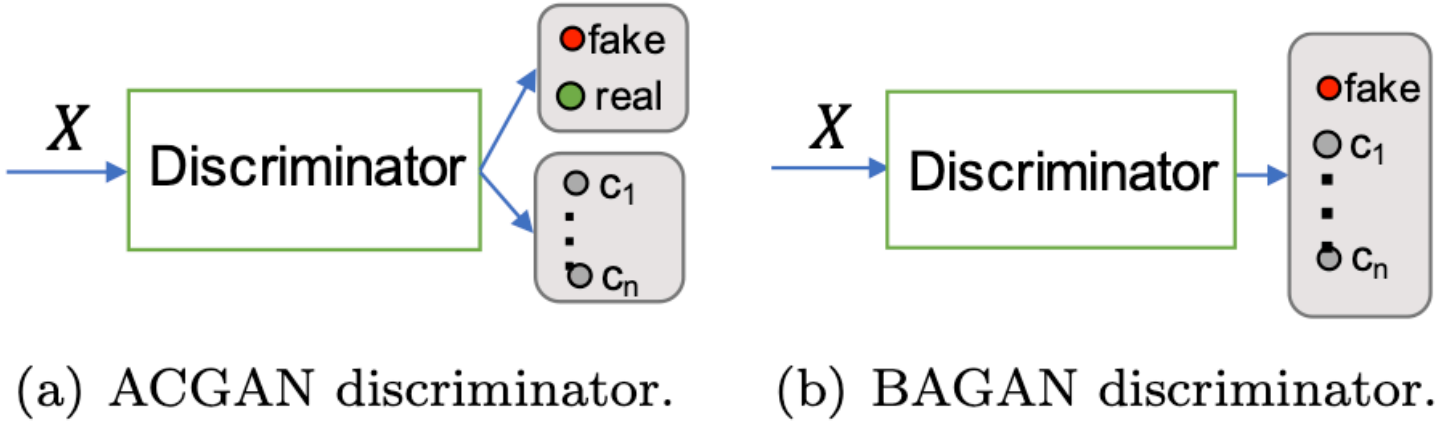
Common ways

- Over-sampling/Under-sampling
- Synthetic Minority Oversampling Technique (SMOTE)

Based on GAN

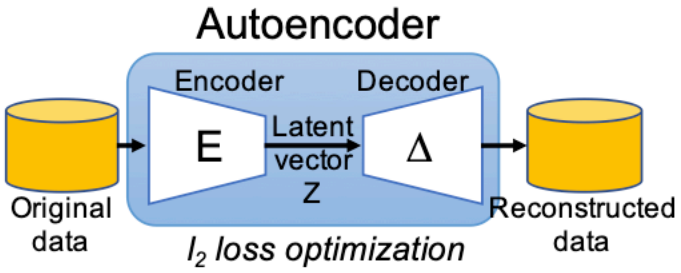
- Balancing GAN (BAGAN)

Discuss difference between two discriminators

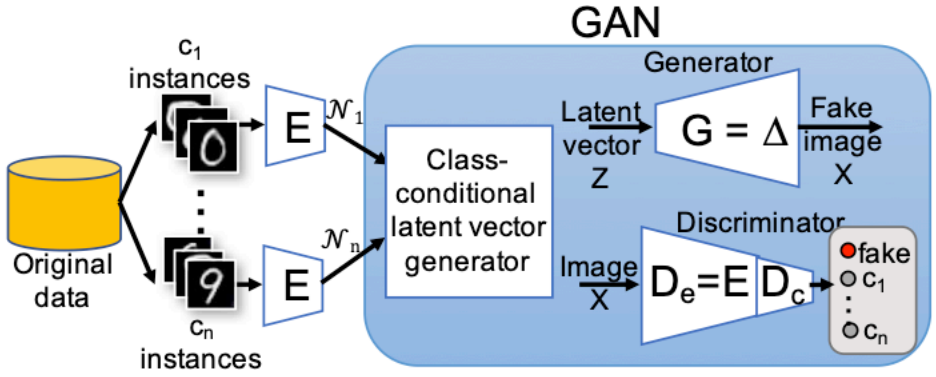


(a) ACGAN discriminator. (b) BAGAN discriminator.

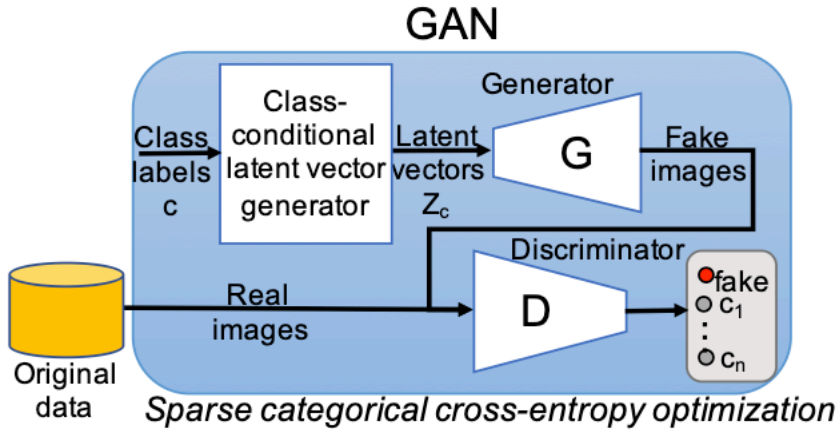
BAGAN Architecture



(a) Autoencoder training.



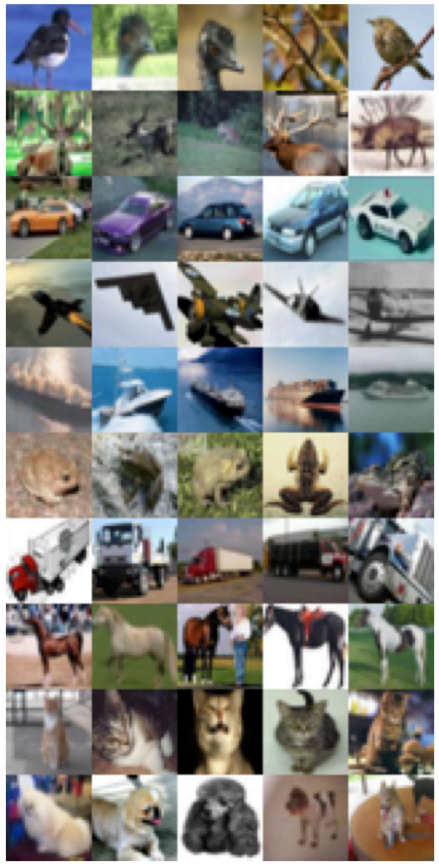
(b) GAN initialization.



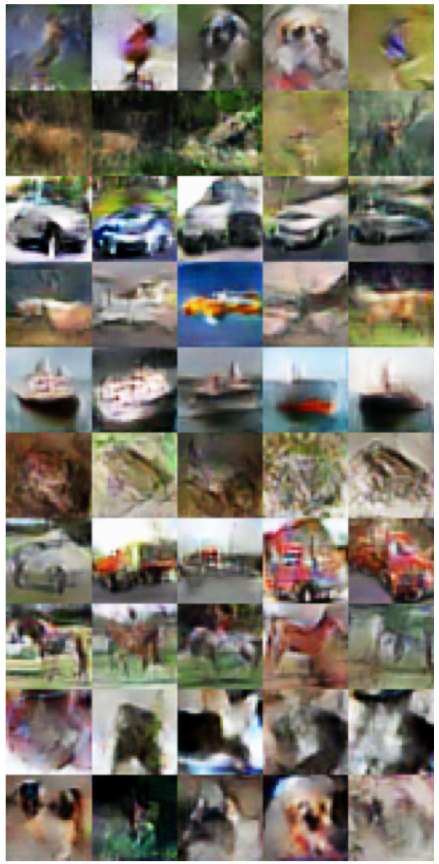
(c) GAN training.

POLab

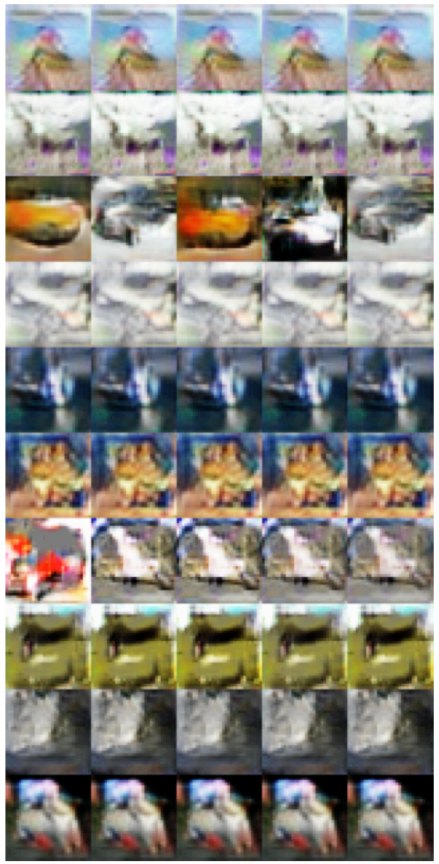
Results



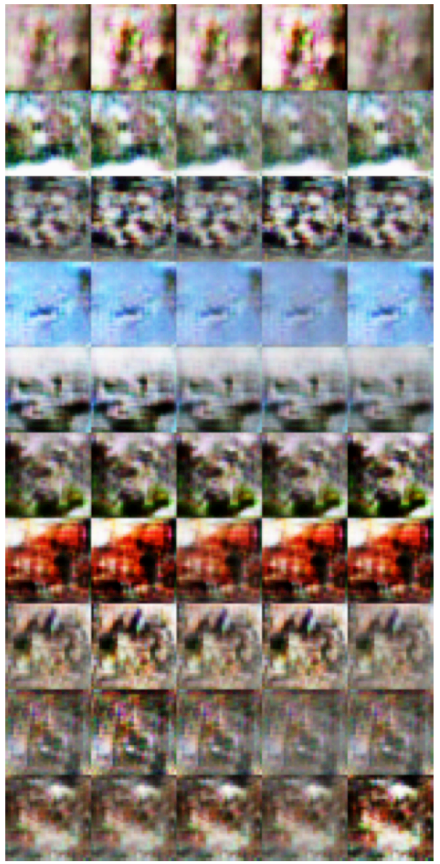
(a) Real image samples



(b) *BAGAN*



(c) *ACGAN*



(d) *Simple GAN*

Implementation with PyTorch

https://github.com/TzuYuOu/GAN_with_PyTorch

POLab

Generator

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        def block(in_feat, out_feat, normalize=True):
            layers = [nn.Linear(in_feat, out_feat)]
            if normalize:
                layers.append(nn.BatchNorm1d(out_feat, 0.8))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *block(latent_dim, 128, normalize=False),
            *block(128, 256),
            *block(256, 512),
            *block(512, 1024),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        img = img.view(img.size(0), *img_shape)
        return img
```

POLab

Discriminator

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        self.model = nn.Sequential(
            nn.Linear(int(np.prod(img_shape)), 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, img):
        img_flat = img.view(img.size(0), -1)
        validity = self.model(img_flat)

        return validity
```

POLab

Training preparation

```
cuda = True if torch.cuda.is_available() else False
# Loss function
adversarial_loss = torch.nn.BCELoss()

# Initialize generator and discriminator
generator = Generator()
discriminator = Discriminator()

if cuda:
    generator.cuda()
    discriminator.cuda()
    adversarial_loss.cuda()

# Configure data loader
os.makedirs("../../data/mnist", exist_ok=True)
dataloader = torch.utils.data.DataLoader(
    datasets.MNIST(
        "../../data/mnist",
        train=True,
        download=True,
        transform=transforms.Compose(
            [transforms.Resize(img_size), transforms.ToTensor(), transforms.Normalize([0.5], [0.5])]
        ),
    ),
    batch_size=batch_size,
    shuffle=True,
)

# Optimizers
optimizer_G = torch.optim.Adam(generator.parameters(), lr=lr, betas=(b1,b2))
optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=lr, betas=(b1, b2))

Tensor = torch.cuda.FloatTensor if cuda else torch.FloatTensor
```

POLab

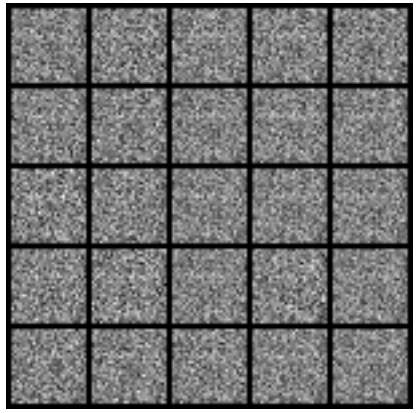
Training

```
for epoch in range(n_epochs):
    for i, (imgs, _) in enumerate(dataloader):

        # Adversarial ground truths
        valid = Variable(Tensor(imgs.size(0), 1).fill_(1.0), requires_grad=False)
        fake = Variable(Tensor(imgs.size(0), 1).fill_(0.0), requires_grad=False)

        # Configure input
        real_imgs = Variable(imgs.type(Tensor))

        # -----------------
        #  Train Generator
        # -----------------

        optimizer_G.zero_grad()

        # Sample noise as generator input
        z = Variable(Tensor(np.random.normal(0, 1, (imgs.shape[0], latent_dim))))

        # Generate a batch of images
        gen_imgs = generator(z)

        # Loss measures generator's ability to fool the discriminator
        g_loss = adversarial_loss(discriminator(gen_imgs), valid)

        g_loss.backward()
        optimizer_G.step()
```

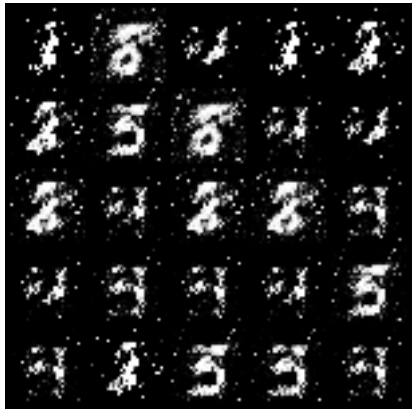
POLab

```
        # ---------------------
        #  Train Discriminator
        # ---------------------

        optimizer_D.zero_grad()

        # Measure discriminator's ability to classify real from generated samples
        real_loss = adversarial_loss(discriminator(real_imgs), valid)
        fake_loss = adversarial_loss(discriminator(gen_imgs.detach()), fake)
        d_loss = (real_loss + fake_loss) / 2

        d_loss.backward()
        optimizer_D.step()

        print(
            "[Epoch %d/%d] [Batch %d/%d] [D loss: %f] [G loss: %f]"
            % (epoch, n_epochs, i, len(dataloader), d_loss.item(), g_loss.item())
        )

        batches_done = epoch * len(dataloader) + i
        if batches_done % sample_interval == 0:
            save_image(gen_imgs.data[:25], "images/%d.png" % batches_done, nrow=5, normalize=True)
```

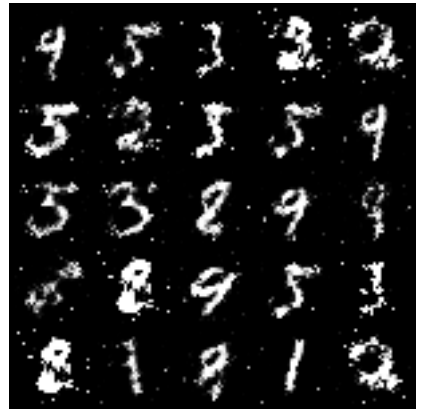
Results



epoch1



epoch20



epoch50

Results

epoch100

epoch150

epoch200

Reference

- Machine Learning and having it deep and structured (2018,Spring) http://speech.ee.ntu.edu.tw/~tlkagk/courses_MLDS18.html
- Missing Data Imputation using Generative Adversarial Nets https://arxiv.org/abs/1806.02920
- Data Argumentation with Balancing GAN https://arxiv.org/abs/1803.09655
- PyTorch-GAN https://github.com/eriklindernoren/PyTorch-GAN
- ACGAN https://arxiv.org/abs/1610.09585