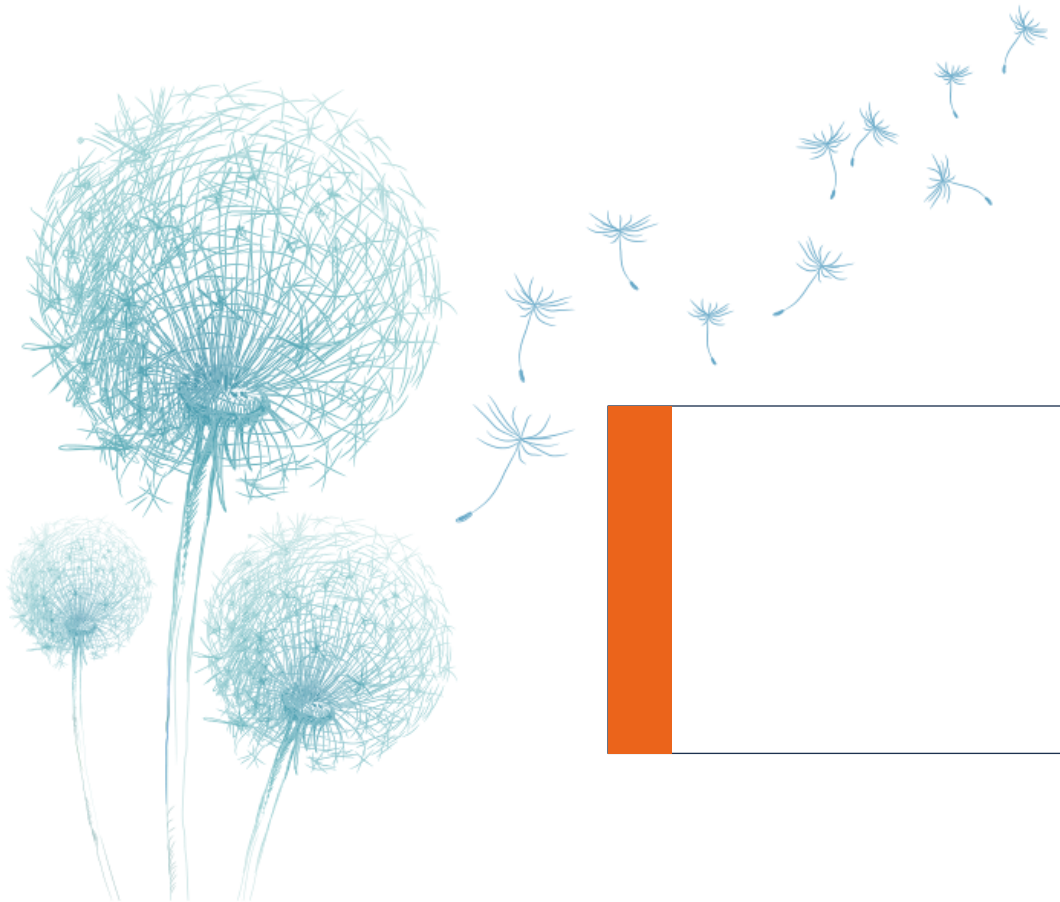
Neural networks and deep learning



GAN

Kun Suo
Computer Science, Kennesaw State University
https://kevinsuo.github.io/

Generative Adversarial Network (GAN)

- https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-unsupervised-learning

What are some recent and potentially upcoming breakthroughs in unsupervised learning?

Yann LeCun, Director of AI Research at Facebook and Professor at NYU

Written Jul 29 · Upvoted by Joaquin Quiñonero Candela, Director Applied Machine Learning at Facebook and Huang Xiao

Adversarial training is the coolest thing since sliced bread.

I've listed a bunch of relevant papers in a previous answer.

Expect more impressive results with this technique in the coming years.

What's missing at the moment is a good understanding of it so we can make it work reliably. It's very finicky. Sort of like ConvNet were in the 1990s, when I had the reputation of being the only person who could make them work (which wasn't true).

Yann LeCun's comment

What are some recent and potentially upcoming breakthroughs in deep learning?

Yann LeCun, Director of AI Research at Facebook and Professor at NYU

Written Jul 29 · Upvoted by Joaquin Quiñonero Candela, Director Applied Machine Learning at Facebook and Nikhil Garg, I lead a team of Quora engineers working on ML/NLP problems

.....

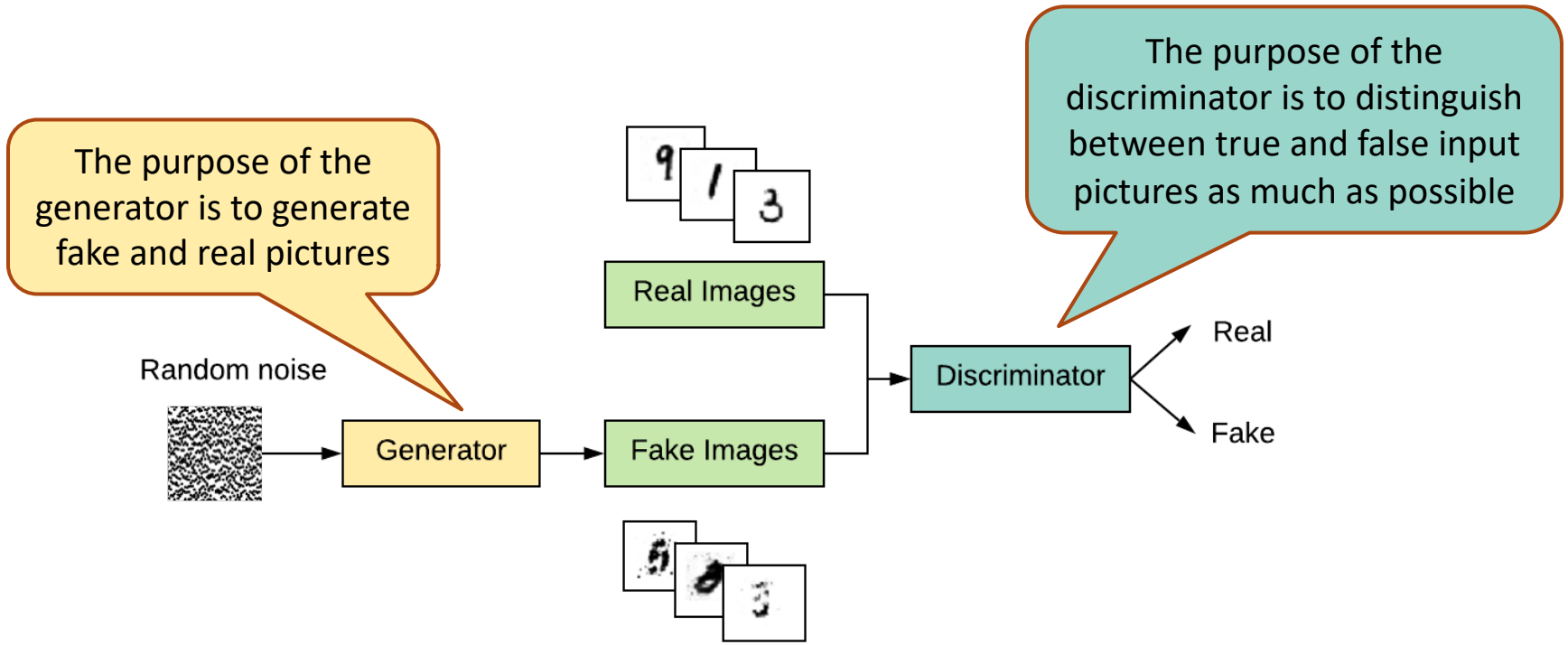
The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks). This is an idea that was originally proposed by Ian Goodfellow when he was a student with Yoshua Bengio at the University of Montreal (he since moved to Google Brain and recently to OpenAI).

This, and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion.

Generative Adversarial Network

- GAN was first proposed by GoodFellow in 2014, check the original paper: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

- GAN structure: a Generator + a Discriminator



All Kinds of GAN …

- https://github.com/hindupuravinash/the-gan-zoo

GAN

ACGAN

BGAN

CGAN

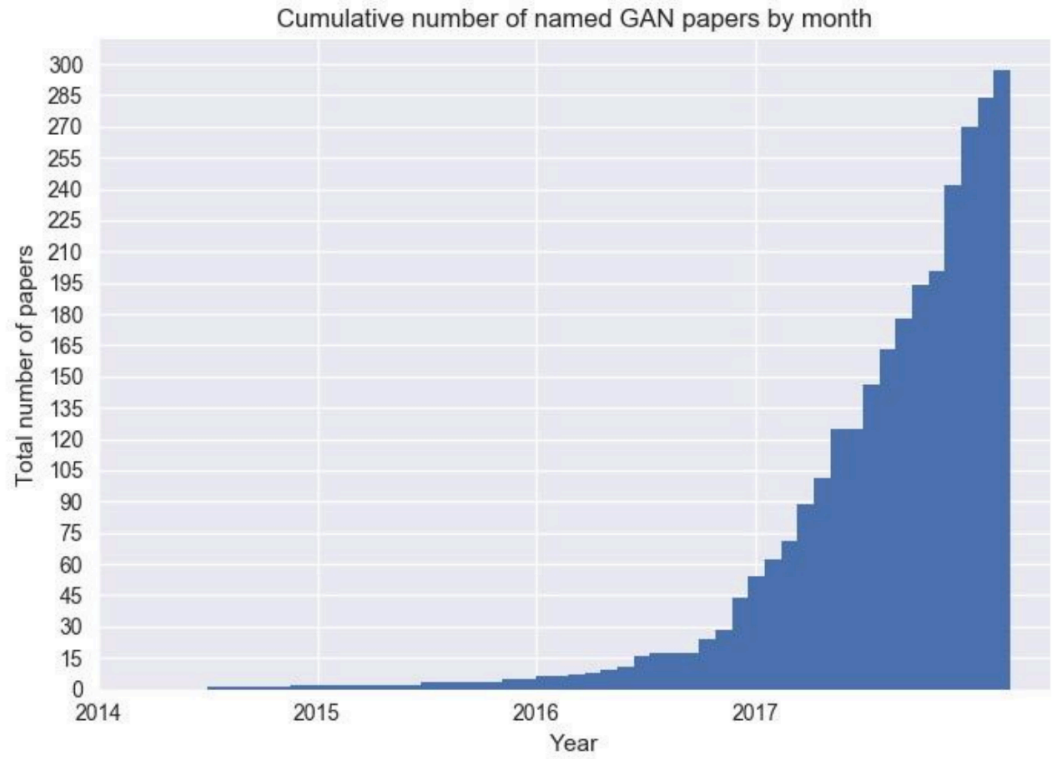
DCGAN

EBGAN

fGAN

GoGAN

⋮

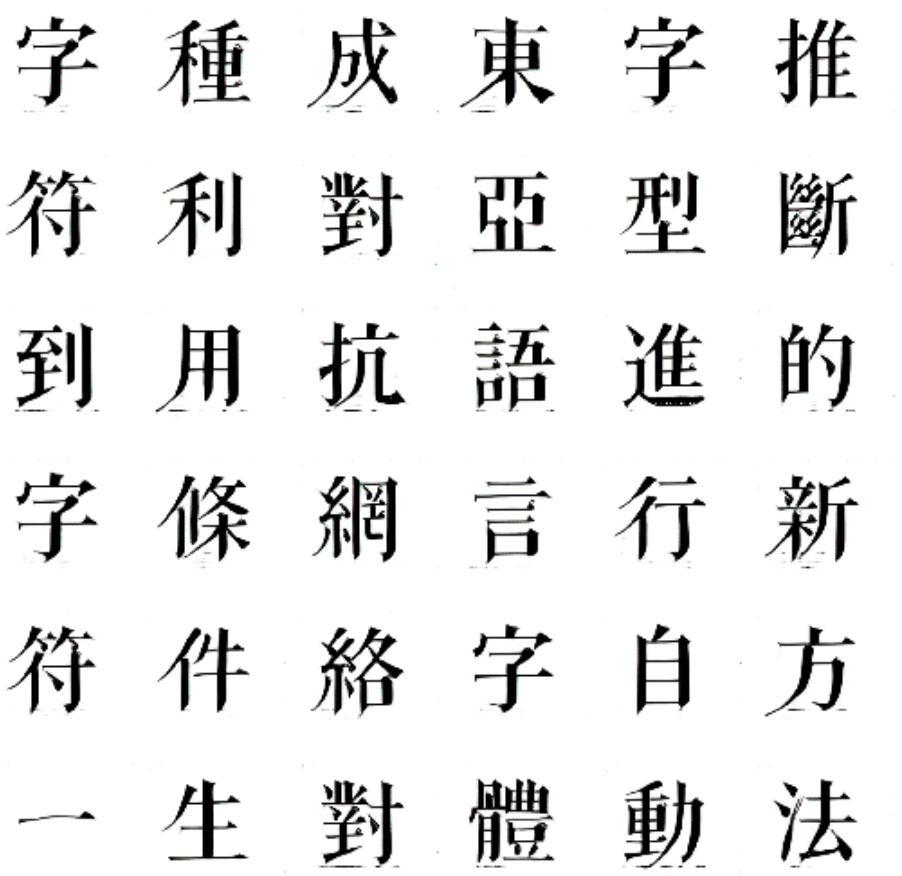


Mihaela Rosca, Balaji Lakshminarayanan, David Warde-Farley, Shakir Mohamed, "Variational Approaches for Auto-Encoding Generative Adversarial Networks", arXiv, 2017

[2]We use the Greek α prefix for α-GAN, as AEGAN and most other Latin prefixes seem to have been taken https://deephunt.in/the-gan-zoo-79597dc8c347.

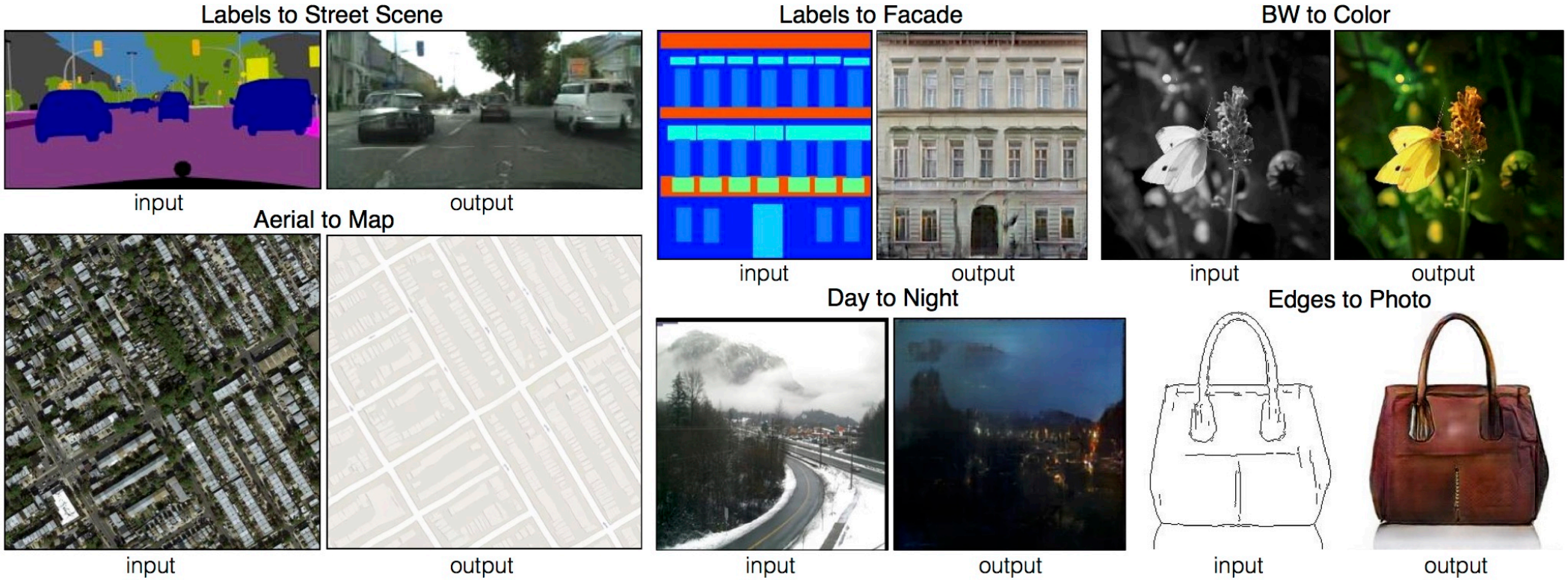
Example: zi2zi: Master Chinese Calligraphy with Conditional Adversarial Networks (Font learning)

- https://github.com/kaonashi-tyc/zi2zi



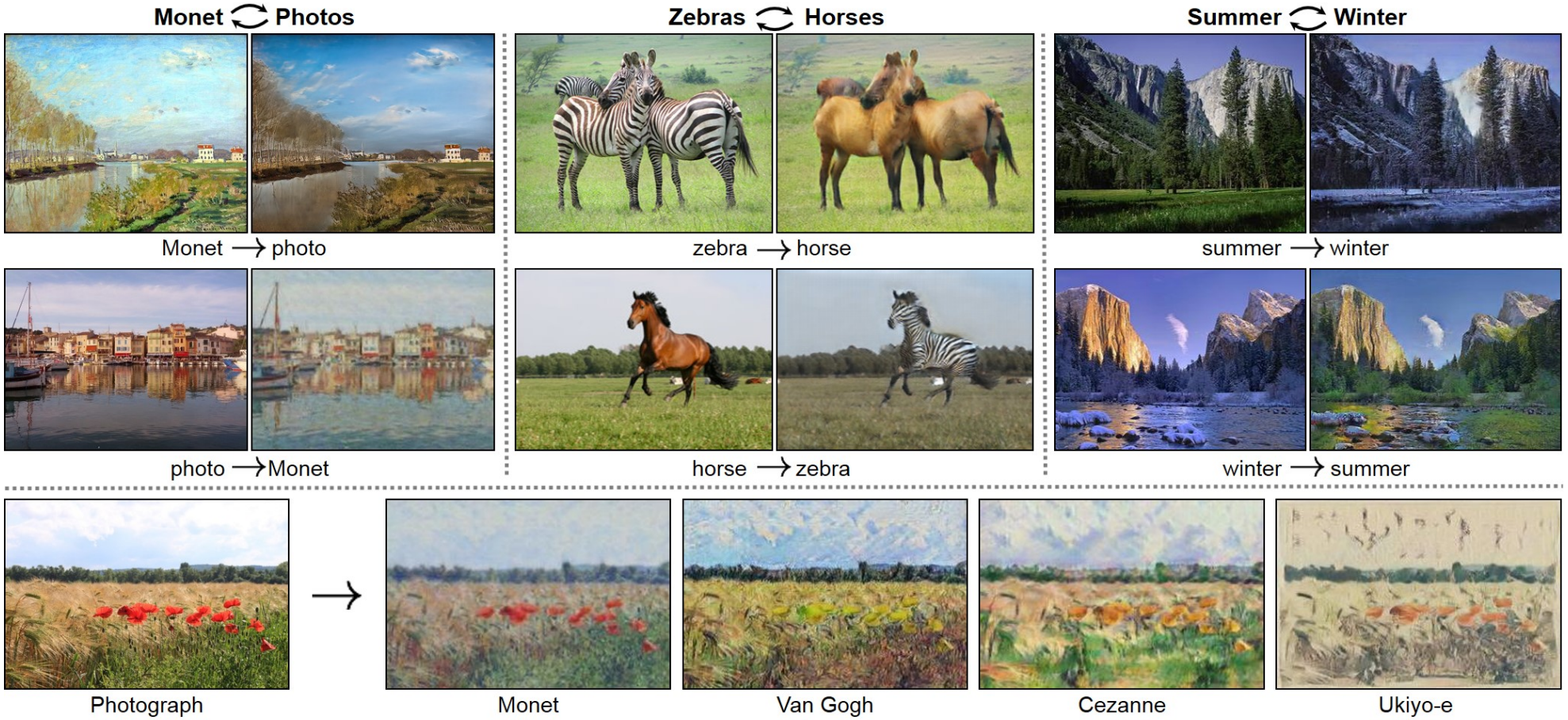
Example: pix2pixGAN, Learns a mapping from input images to output images

- https://github.com/affinelayer/pix2pix-tensorflow



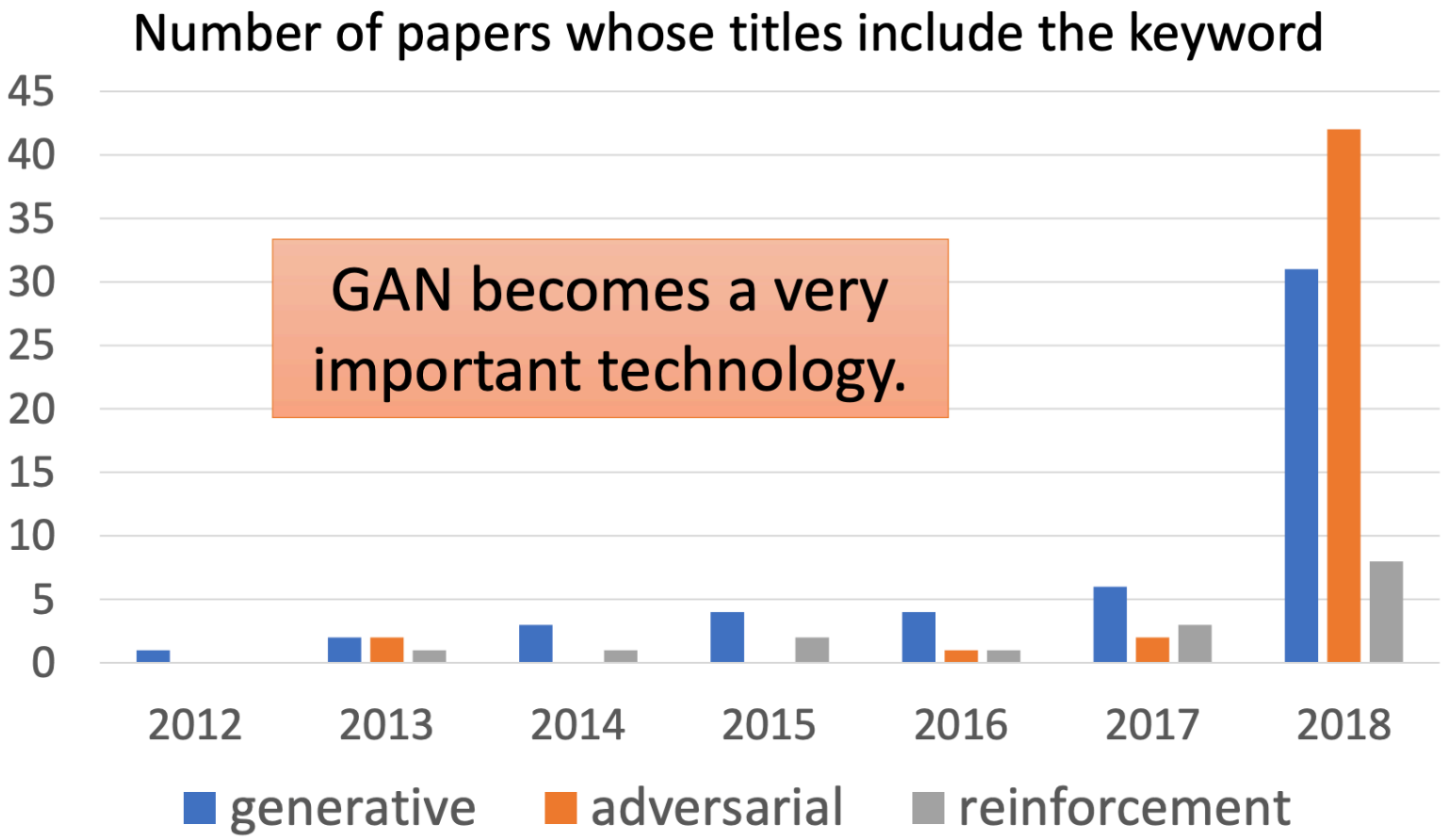
Example: Style transfer

- https://github.com/junyanz/CycleGAN



ICASSP

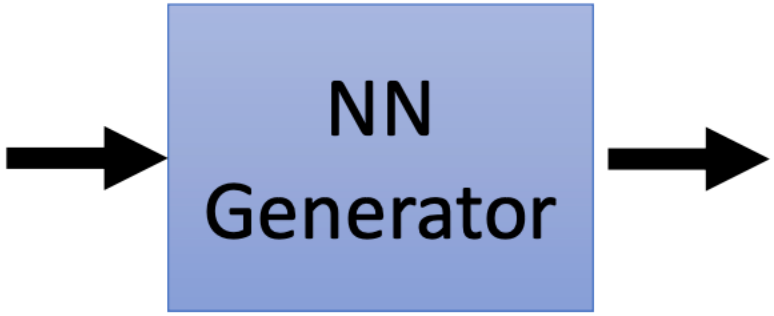
Keyword search on session index page, so session names are included.

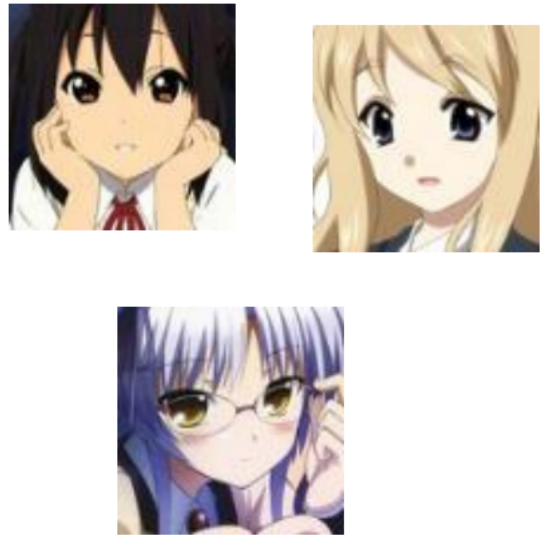


Generation

- Image Generation

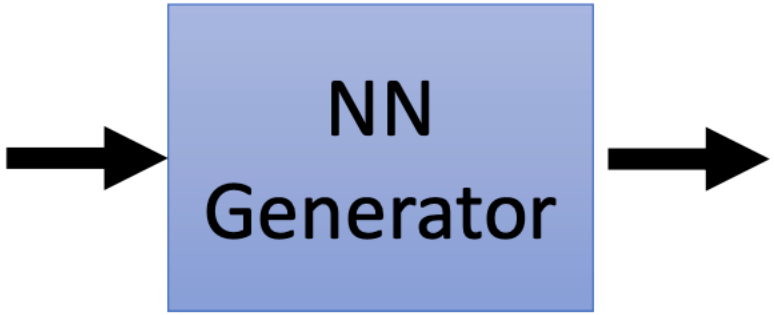
$$\begin{bmatrix} 0.3 \\ -0.1 \\ \vdots \\ -0.7 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.1 \\ \vdots \\ 0.7 \end{bmatrix} \begin{bmatrix} -0.3 \\ 0.1 \\ \vdots \\ 0.9 \end{bmatrix}$$





- Sentence Generation

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



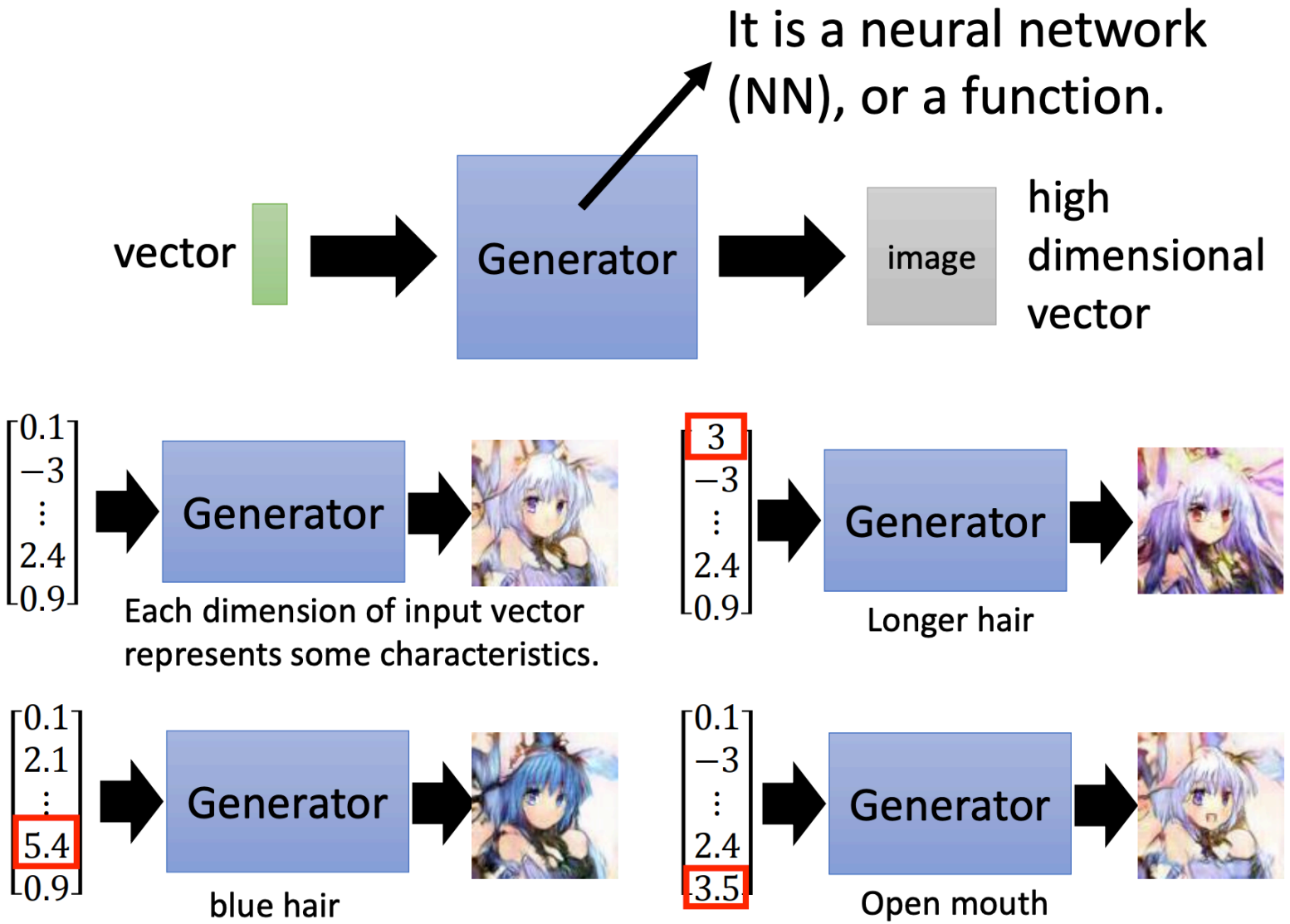
How are you?

Good morning.

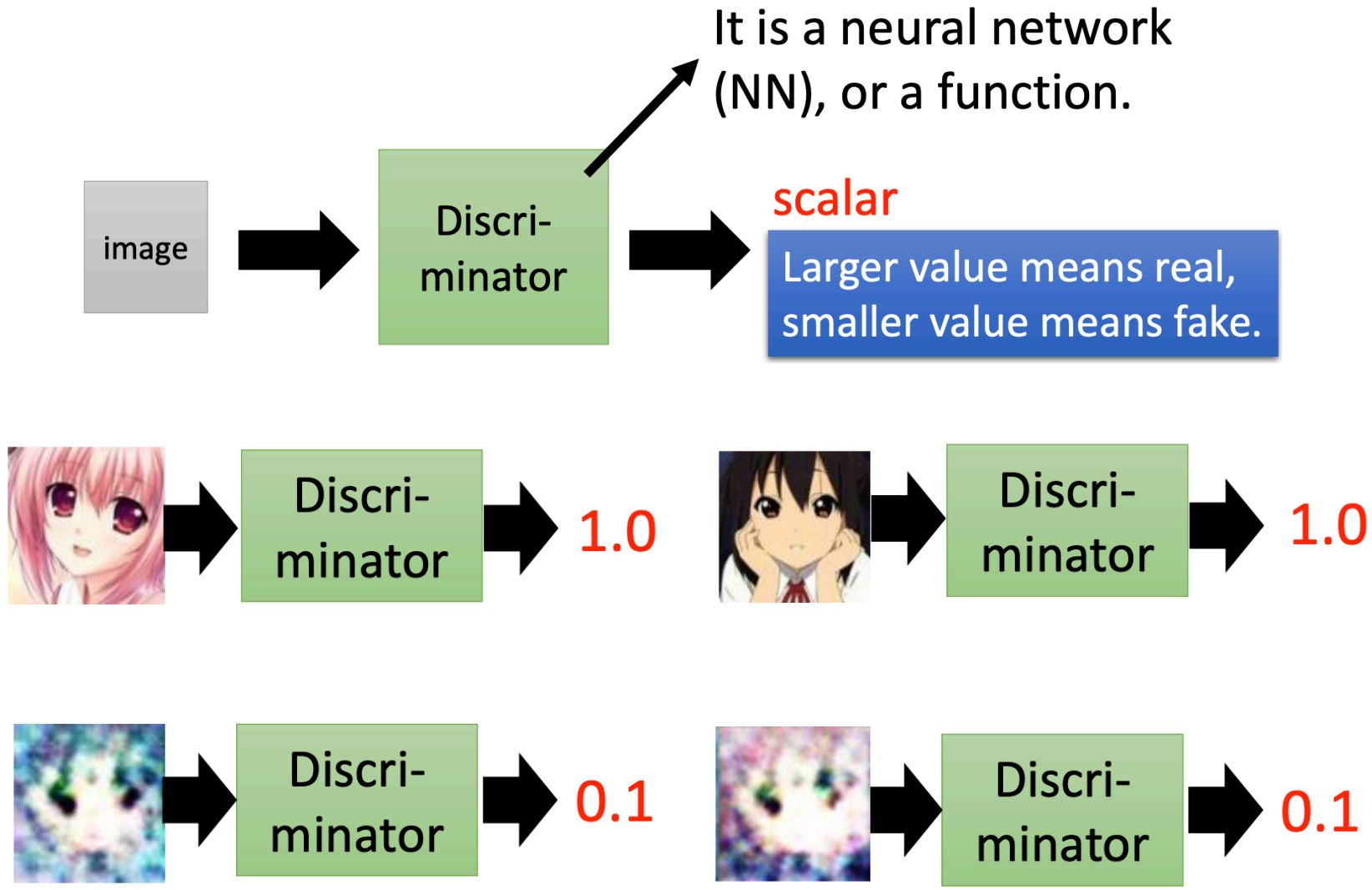
Good afternoon.

Basic Idea of GAN

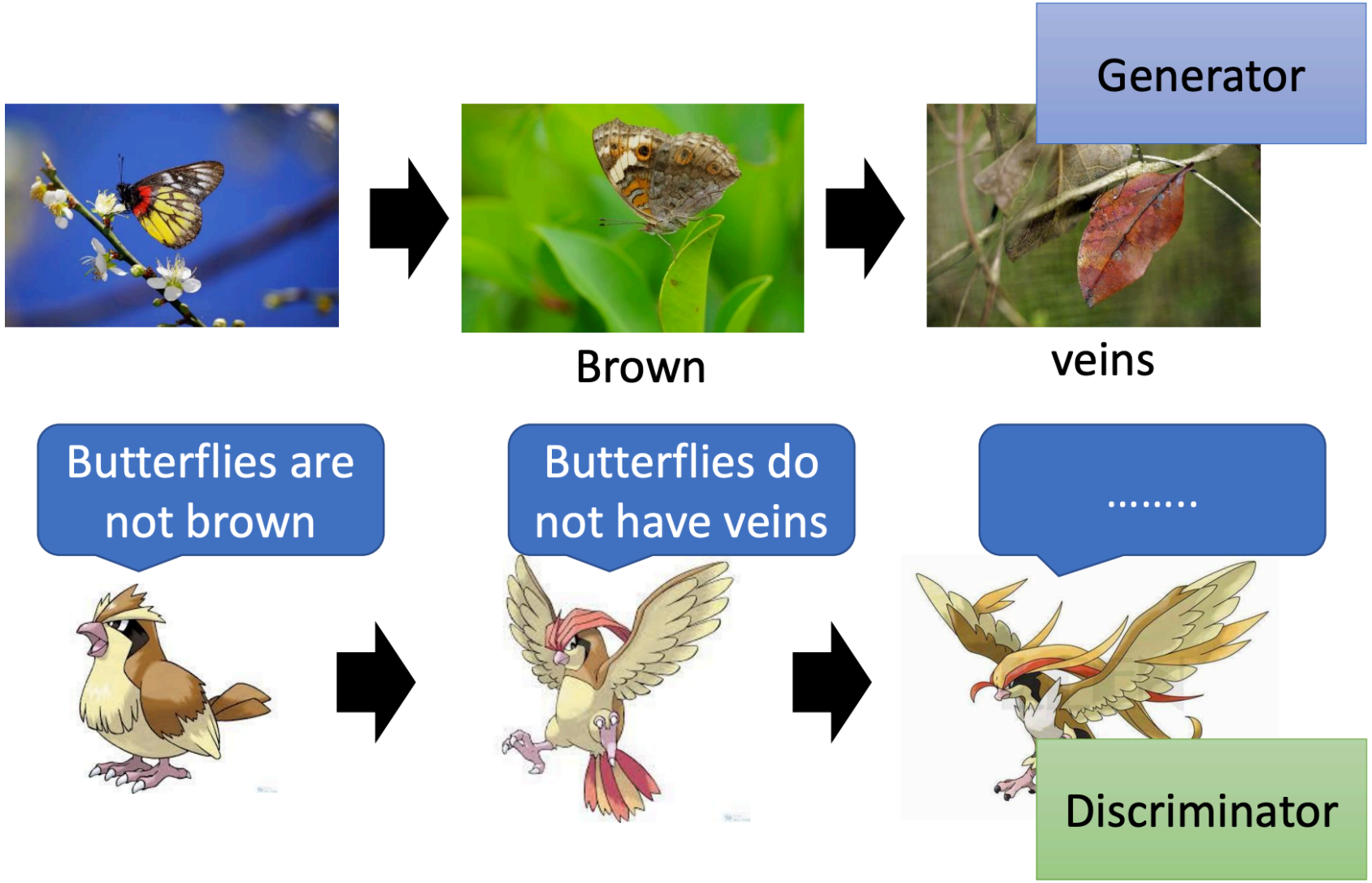
http://mattya.github.io/chainer-DCGAN/



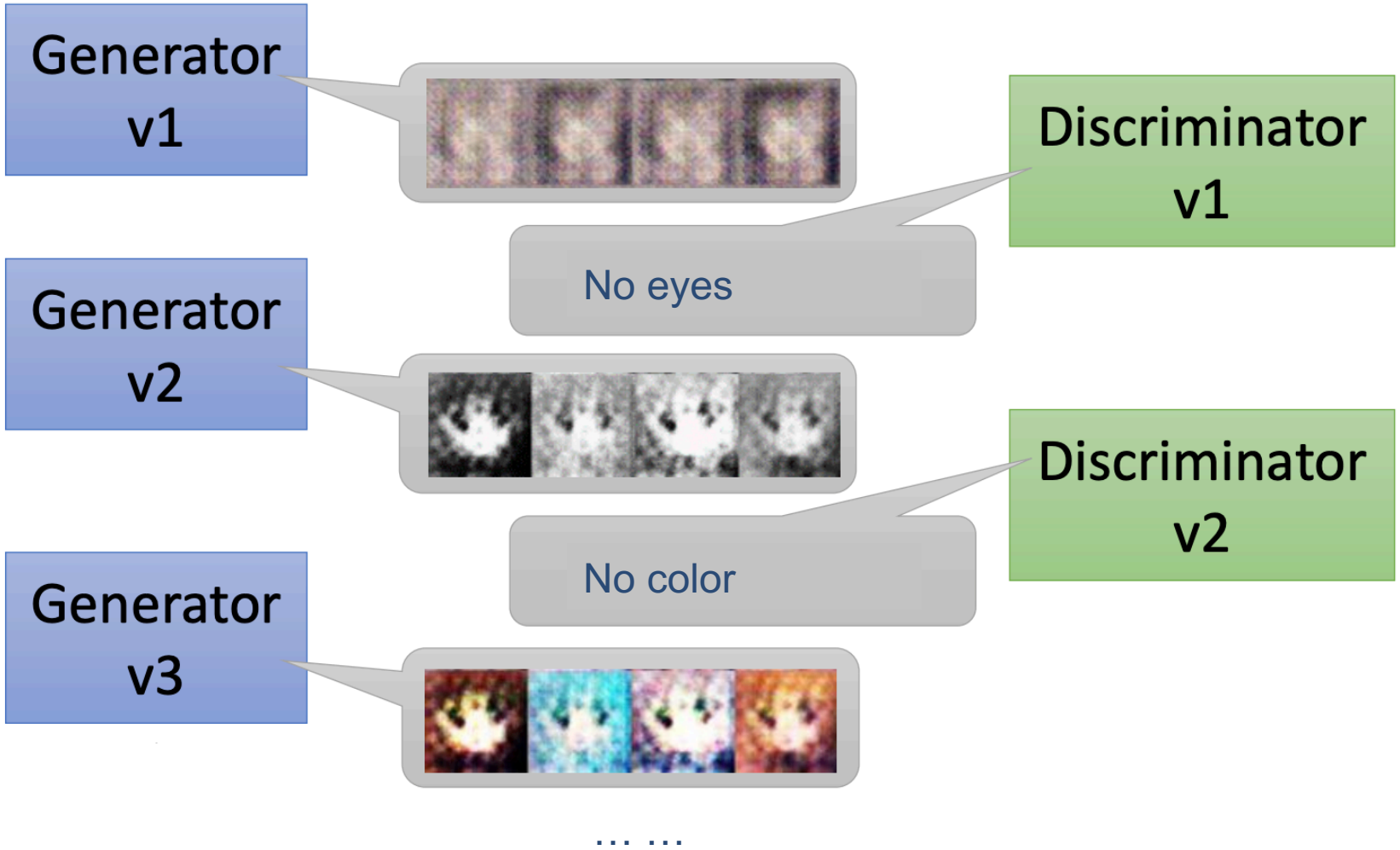
Basic Idea of GAN



Basic Idea of GAN

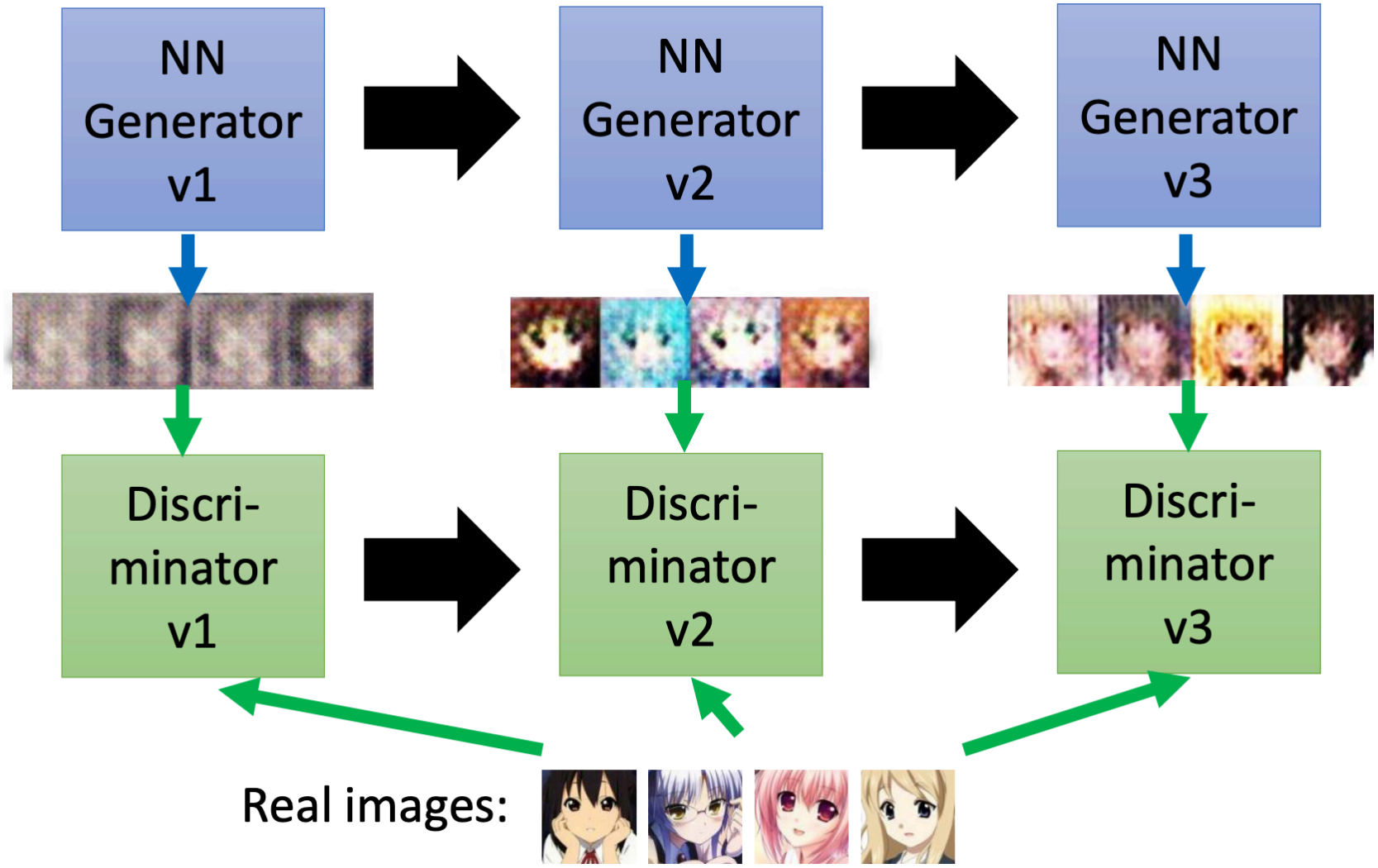


Basic Idea of GAN



Basic Idea of GAN

This is where the term "adversarial" comes from. You can explain the process in different ways…….

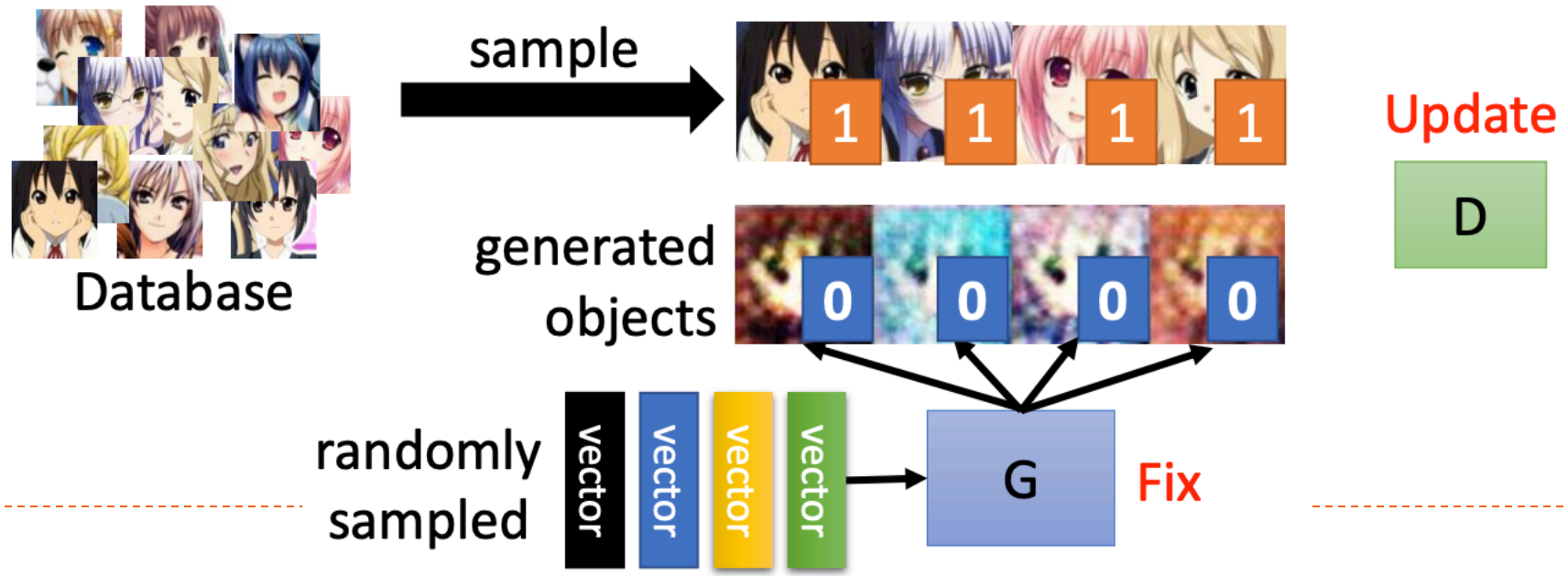


Algorithm of GAN

- 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.

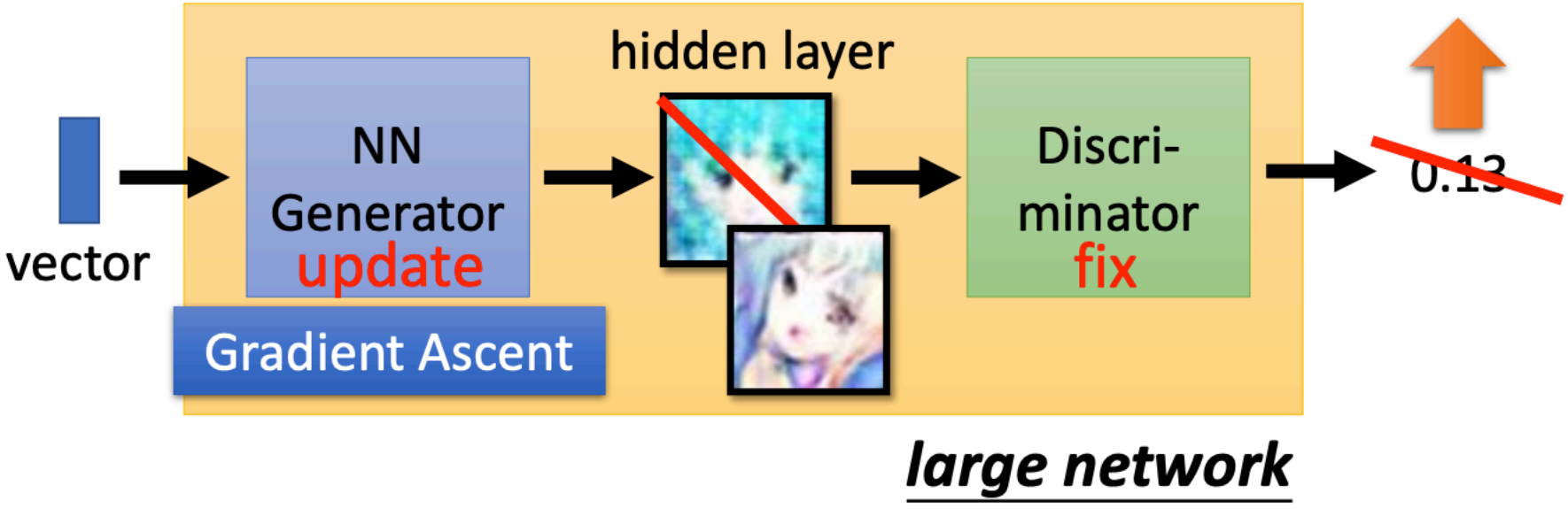


Algorithm of GAN

- Initialize generator and discriminator
- In each training iteration:

Step 2: Fix discriminator D, and update generator G

Generator learns to "fool" the discriminator



Algorithm of GAN

Initialize θ_d for D and θ_g for G

- In each training iteration:

Learning D

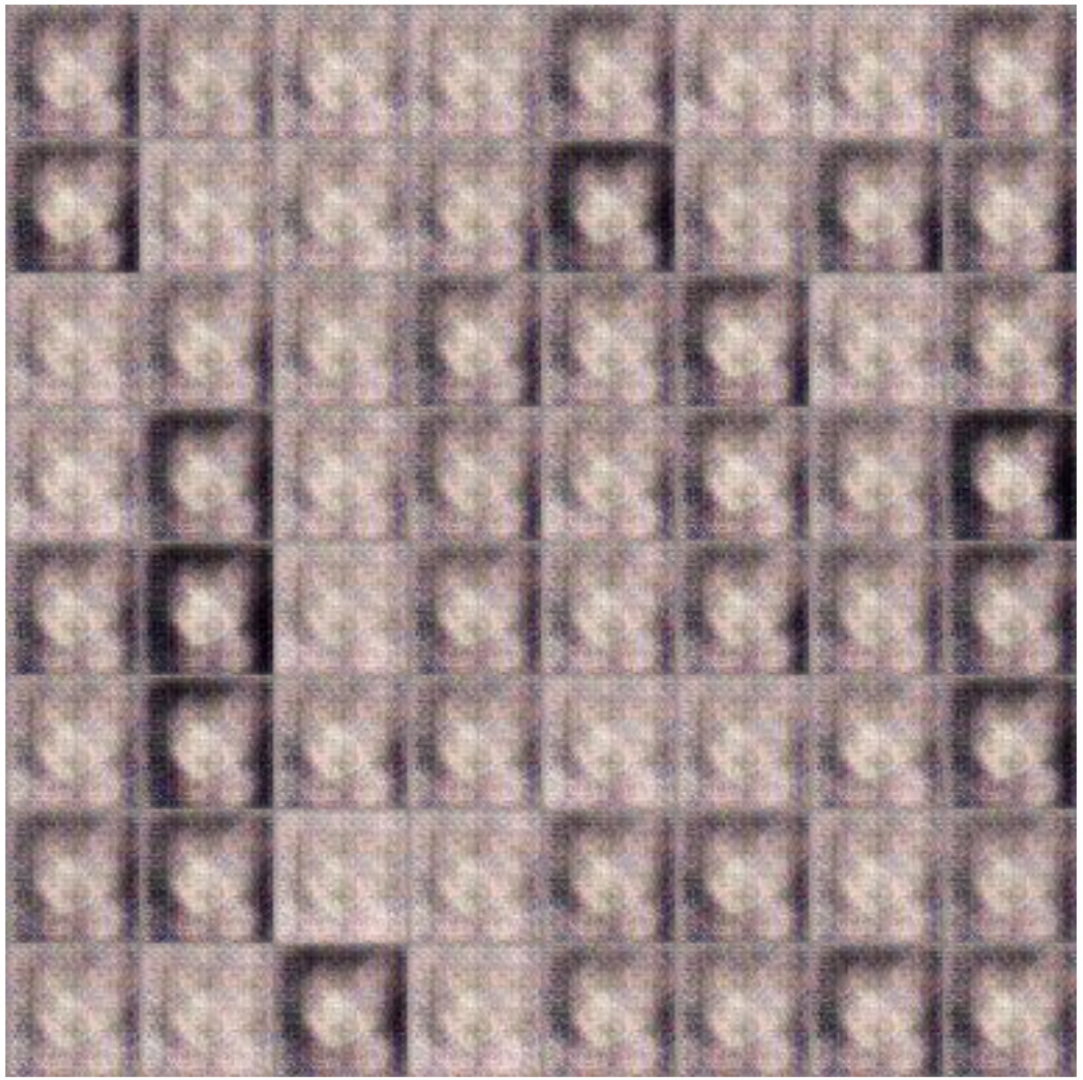
- Sample m examples $\{x^1, x^2, \dots, x^m\}$ from database
- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Obtaining generated data $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_d to maximize
  - $\tilde{V} =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

Learning G

- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ 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



Anime Face Generation

- 1000 updates

Anime Face Generation

- 2000 updates

Anime Face Generation

- 5000 updates

Anime Face Generation

- 10000 updates

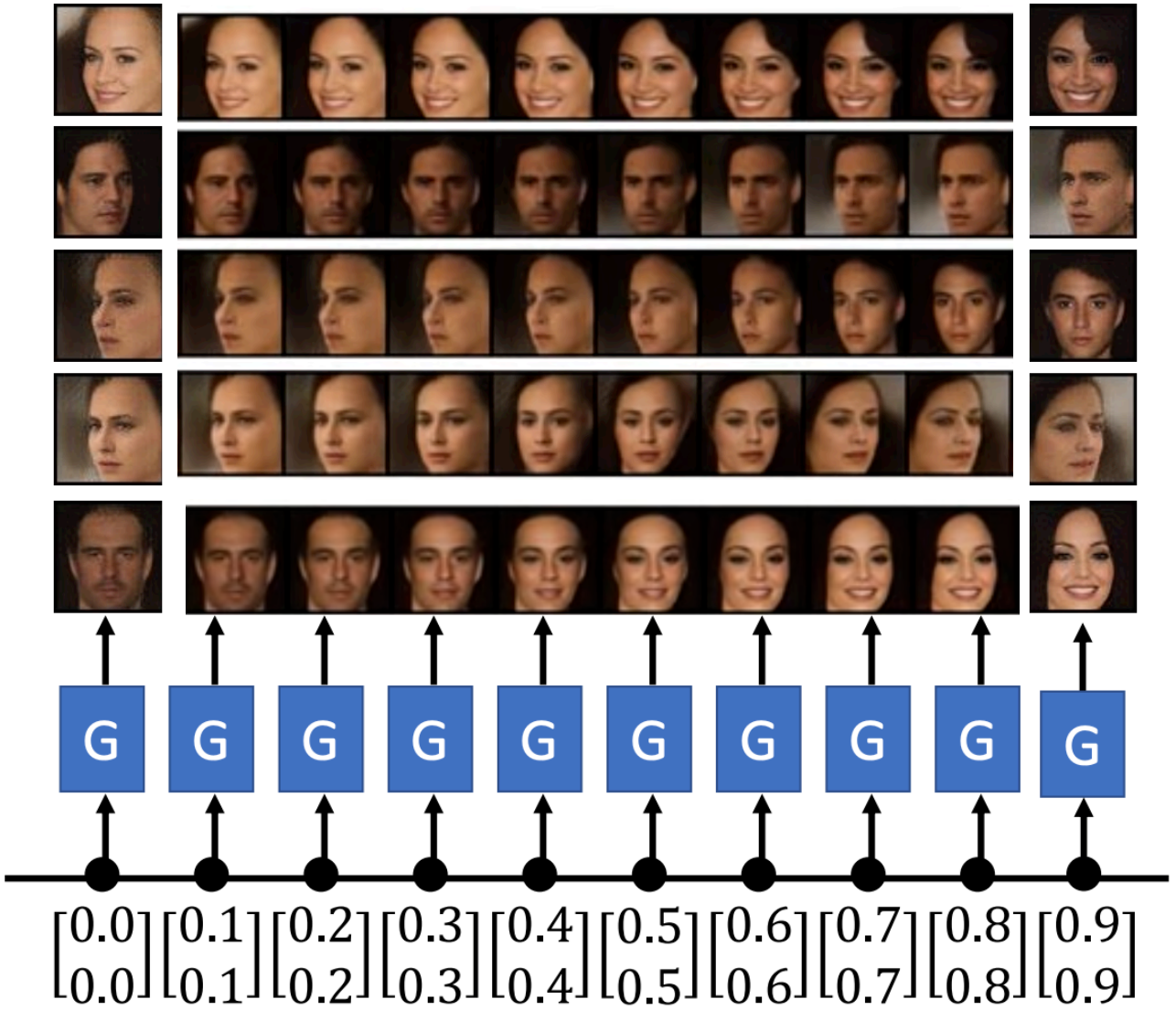
Anime Face Generation

- 20000 updates

Anime Face Generation

- 50000 updates

Example


G G G G G G G G G G
[0.0 0.0] [0.1 0.1] [0.2 0.2] [0.3 0.3] [0.4 0.4] [0.5 0.5] [0.6 0.6] [0.7 0.7] [0.8 0.8] [0.9 0.9]

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)

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

Output Sequence

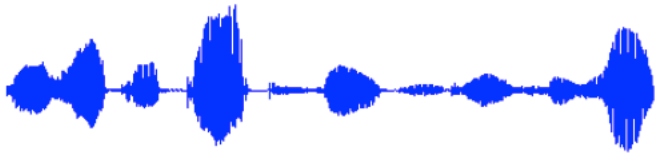
$$f: X \rightarrow Y$$

- 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: Hello world
(transcription)

- Chat-bot

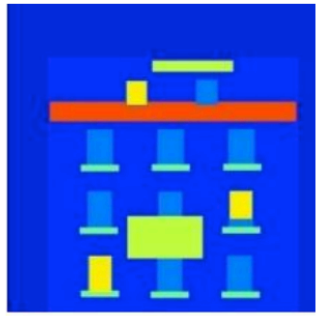
X: "How are you?"
(what a user says)

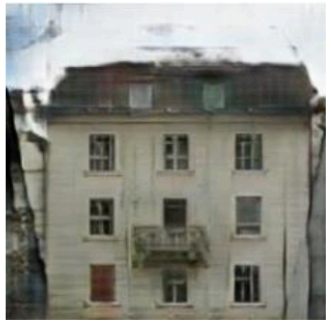
Y: "I'm fine."
(response of machine)

Output Matrix

$$f : X \rightarrow Y$$

- Image to Image

X : 

Y : 

Colorization:

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

- Text to Image

Text to Image

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

Y : 

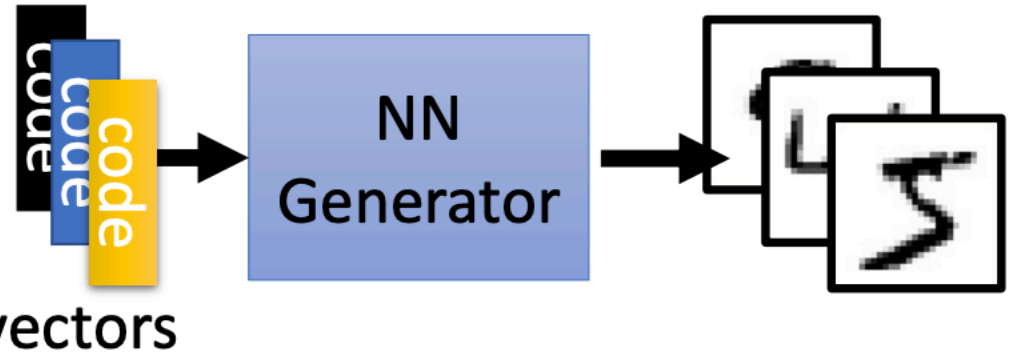
https://arxiv.org/pdf/1605.05396.pdf

Structured Learning Approach

Generator:
Learn to generate the object at the component level

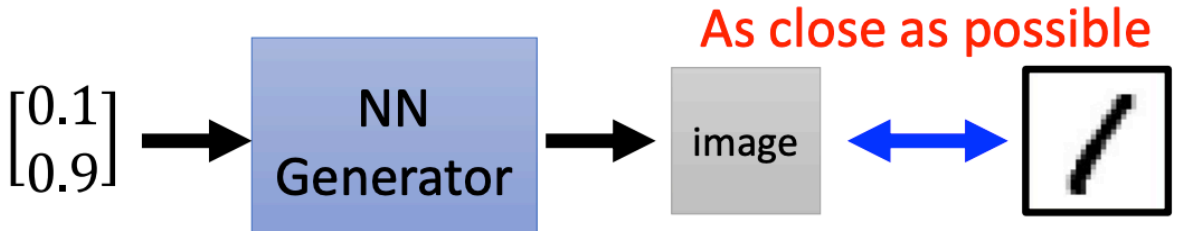
Discriminator:
Evaluating the whole object, and find the best one

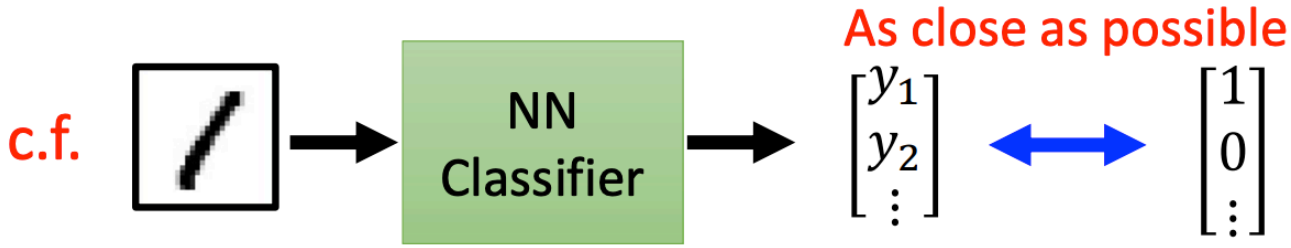
Can Generator learn by itself?



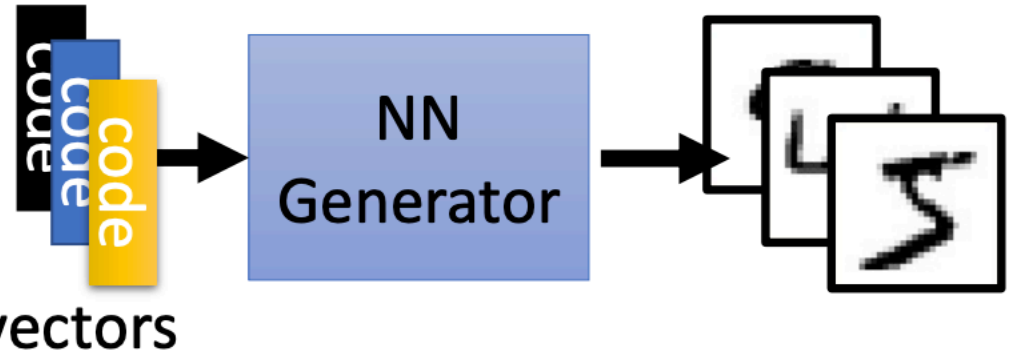
code: $\begin{bmatrix} 0.1 \\ -0.5 \end{bmatrix}$ $\begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}$ $\begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$ $\begin{bmatrix} 0.3 \\ 0.2 \end{bmatrix}$

Image:



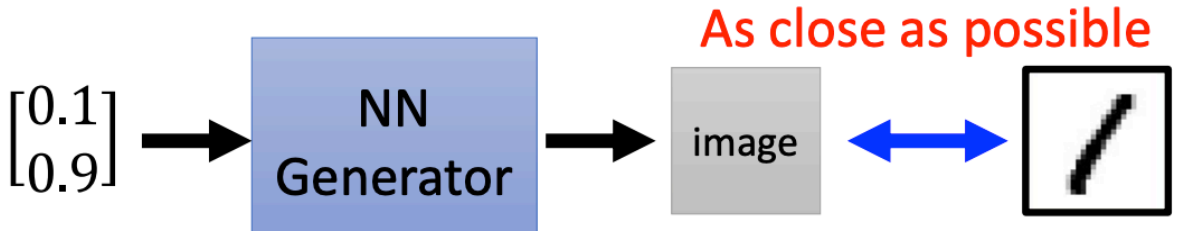


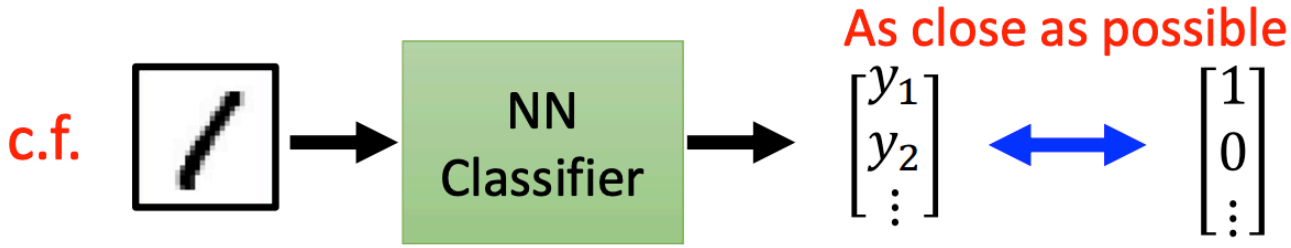
Can Generator learn by itself?



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

Image:





GAN advantage

- The GAN model only uses back propagation
- The parameter update of G is not directly from the data sample
- GAN is a semi-supervised learning model, which does not require much labeled data for the training set

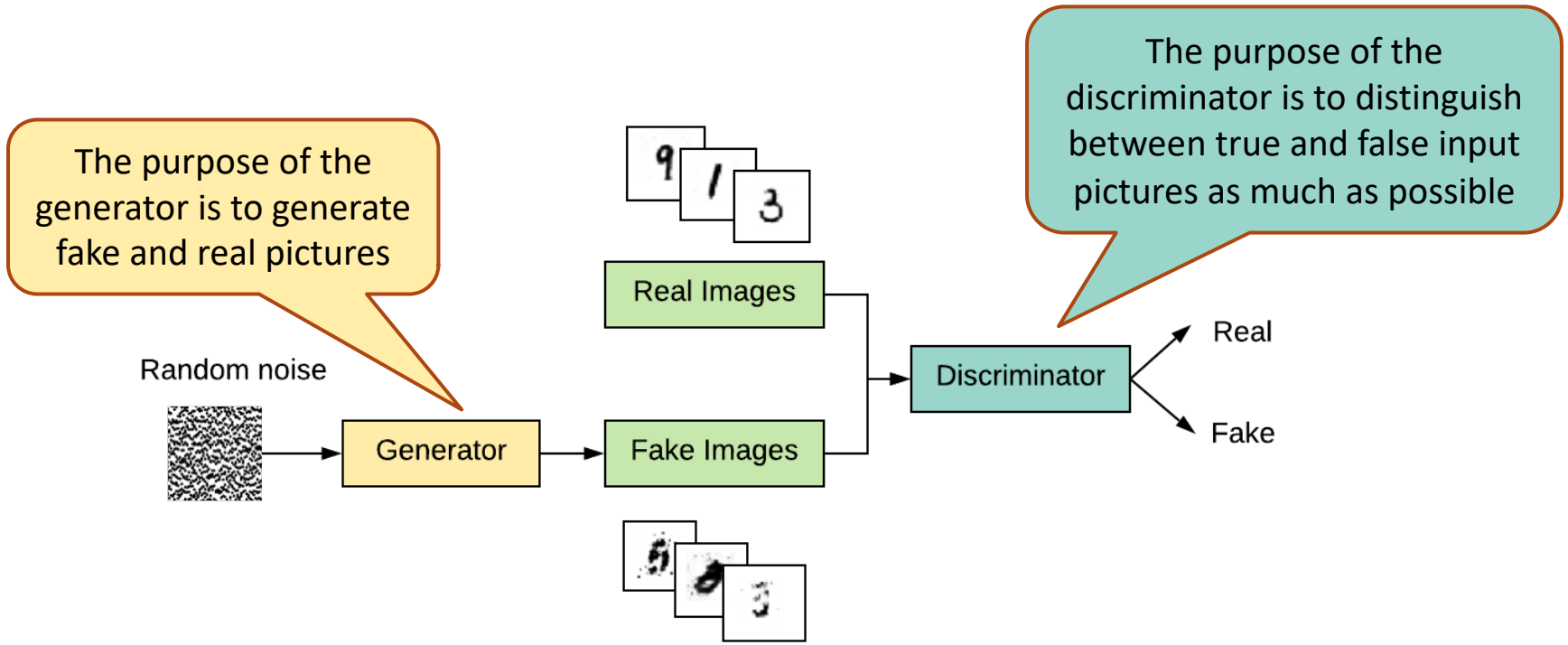
GAN disadvantage

- Poor interpretability
- Harder to train, good synchronization between D and G is required
- It is difficult to learn to generate discrete data, like text
- The training is unstable, G and D are difficult to converge, and will encounter the problem of gradient disappearance and mode collapse
- Lack of effective methods for evaluating models

Build a Generative Adversarial Network

- https://github.com/kevinsuo/CS7357/blob/master/gan/gan.py

- Review: GAN structure: a Generator + a Discriminator



Build a Generative Adversarial Network

- Import package

```
1   from keras.datasets import mnist
2   from keras.layers import Dense, Dropout, Input
3   from keras.models import Model,Sequential
4   from keras.layers.advanced_activations import LeakyReLU
5   from keras.optimizers import Adam
6   from tqdm import tqdm
7   import numpy as np
8   import matplotlib.pyplot as plt
9   #matplotlib inline
10  #from google.colab import drive
11
12
```

Build a Generative Adversarial Network

- Read the mnist data set provided by Keras. Here we use load_data() function

```
15
16  # Load the dataset
17  def load_data():
18      (x_train, y_train), (_, _) = mnist.load_data()
19      x_train = (x_train.astype(np.float32) - 127.5)/127.5
20
21      # Convert shape from (60000, 28, 28) to (60000, 784)
22      x_train = x_train.reshape(60000, 784)
23      return (x_train, y_train)
24
25  X_train, y_train = load_data()
26  print(X_train.shape, y_train.shape)
27
```

- Output:

```
(60000, 784) (60000,)
```

Build a Generative Adversarial Network

- Use the MLP fully connected layer to build the generator

```
30
31    def build_generator():
32        model = Sequential()
33
34        model.add(Dense(units=256, input_dim=100))
35        model.add(LeakyReLU(alpha=0.2))
36
37        model.add(Dense(units=512))
38        model.add(LeakyReLU(alpha=0.2))
39
40        model.add(Dense(units=1024))
41        model.add(LeakyReLU(alpha=0.2))
42
43        model.add(Dense(units=784, activation='tanh'))
44
45        model.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
46        return model
47
48    generator = build_generator()
49    generator.summary()
50
```

Build a Generative Adversarial Network

```
51    generator.summary()
```

- Output:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               25856
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 784)               803600
=================================================================
Total params: 1,486,352
Trainable params: 1,486,352
Non-trainable params: 0
```

Generator structure

Build a Generative Adversarial Network

- Use an MLP fully connected neural network to build a discriminator

```
57  def build_discriminator():
58      model = Sequential()
59
60      model.add(Dense(units=1024 ,input_dim=784))
61      model.add(LeakyReLU(alpha=0.2))
62      model.add(Dropout(0.3))
63
64      model.add(Dense(units=512))
65      model.add(LeakyReLU(alpha=0.2))
66      model.add(Dropout(0.3))
67
68      model.add(Dense(units=256))
69      model.add(LeakyReLU(alpha=0.2))
70      model.add(Dropout(0.3))
71
72      model.add(Dense(units=1, activation='sigmoid'))
73
74      model.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
75      return model
76
77  discriminator = build_discriminator()
78
```

Build a Generative Adversarial Network

```
80    discriminator.summary()
```

- Output:

```
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 1024)              803840
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               524800
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 257
=================================================================
Total params: 1,460,225
Trainable params: 1,460,225
Non-trainable params: 0
```

Build a Generative Adversarial Network

- Build a GAN network consisting of discriminator and generator

```
86
87  def build_GAN(discriminator, generator):
88      discriminator.trainable=False
89      GAN_input = Input(shape=(100,))
90      x = generator(GAN_input)
91      GAN_output= discriminator(x)
92      GAN = Model(inputs=GAN_input, outputs=GAN_output)
93      GAN.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
94      return GAN
95
96  GAN = build_GAN(discriminator, generator)
```

Build a Generative Adversarial Network

```
99    GAN.summary()
```

- Output: the structure of GAN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
sequential (Sequential)      (None, 784)               1486352
_________________________________________________________________
sequential_1 (Sequential)    (None, 1)                 1460225
=================================================================
Total params: 2,946,577
Trainable params: 1,486,352
Non-trainable params: 1,460,225
```

Build a Generative Adversarial Network

- The function of drawing the image, used to draw the fake picture generated by the generator

```
104    def draw_images(generator, epoch, examples=25, dim=(5,5), figsize=(10,10)):
105        noise= np.random.normal(loc=0, scale=1, size=[examples, 100])
106        generated_images = generator.predict(noise)
107        generated_images = generated_images.reshape(25,28,28)
108        plt.figure(figsize=figsize)
109        for i in range(generated_images.shape[0]):
110            plt.subplot(dim[0], dim[1], i+1)
111            plt.imshow(generated_images[i], interpolation='nearest', cmap='Greys')
112            plt.axis('off')
113        plt.tight_layout()
114        plt.savefig('Generated_images %d.png' %epoch)
115
```

Build a Generative Adversarial Network

- Train function to train the GAN network
  - Step 1: Import data set
  - Step 2: Build a GAN network by connecting two neural networks (generator, discriminator)

```
119
120   def train_GAN(epochs=1, batch_size=128):
121
122     #Loading the data
123     X_train, y_train = load_data()
124
125     # Creating GAN
126     generator= build_generator()
127     discriminator= build_discriminator()
128     GAN = build_GAN(discriminator, generator)
```

Build a Generative Adversarial Network

- Train function to train the GAN network
  - Step 3: Create a loop (400 iterations). tqdm is used to dynamically display the progress of each iteration

```
129
130     for i in range(1, epochs+1):
131       print("Epoch %d" %i)
132
133       for _ in tqdm(range(batch_size)):
```

Build a Generative Adversarial Network

- Train function to train the GAN network
  - Step 3.2: Generate noise with Gaussian distribution, and then use the generator to generate batch_size (128) pictures. The input of each picture is a 1*100 noise matrix

```
134         # Generate fake images from random noiset
135         noise= np.random.normal(0,1, (batch_size, 100))
136         fake_images = generator.predict(noise)
```

Build a Generative Adversarial Network

- Train function to train the GAN network
  - Step 3.3: We randomly select 128 real pictures from the Mnist data set. We label the real pictures 1 and the fake pictures 0, and then mix 256 real and fake pictures together.

```
138         # Select a random batch of real images from MNIST
139         real_images = X_train[np.random.randint(0, X_train.shape[0], batch_size)]
140
141         # Labels for fake and real images
142         label_fake = np.zeros(batch_size)
143         label_real = np.ones(batch_size)
144
145         # Concatenate fake and real images
146         X = np.concatenate([fake_images, real_images])
147         y = np.concatenate([label_fake, label_real])
```

Build a Generative Adversarial Network

- Train function to train the GAN network
  - Step 3.4: Use the 256 labeled true and false pictures mentioned above to train the discriminator. After training, the weights of discriminator have been updated.

```
149         # Train the discriminator
150         discriminator.trainable=True
151         discriminator.train_on_batch(X, y)
```

Build a Generative Adversarial Network

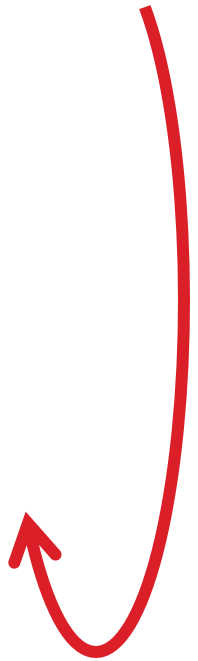
- Train function to train the GAN network
  - Step 3.5: Freeze the discriminator's weights so that the discriminator no longer changes. Then start training the generator.
  - In this process, the generator continues to generate fake pictures, send them to the discriminator for inspection, and get the test results. If it is identified as false, it will continue to update its weight until the discriminator identifies the added image as a true picture.

```
153         # Train the generator/chained GAN model (with frozen weights in discriminator)
154         discriminator.trainable=False
155         GAN.train_on_batch(noise, label_real)
```

Build a Generative Adversarial Network

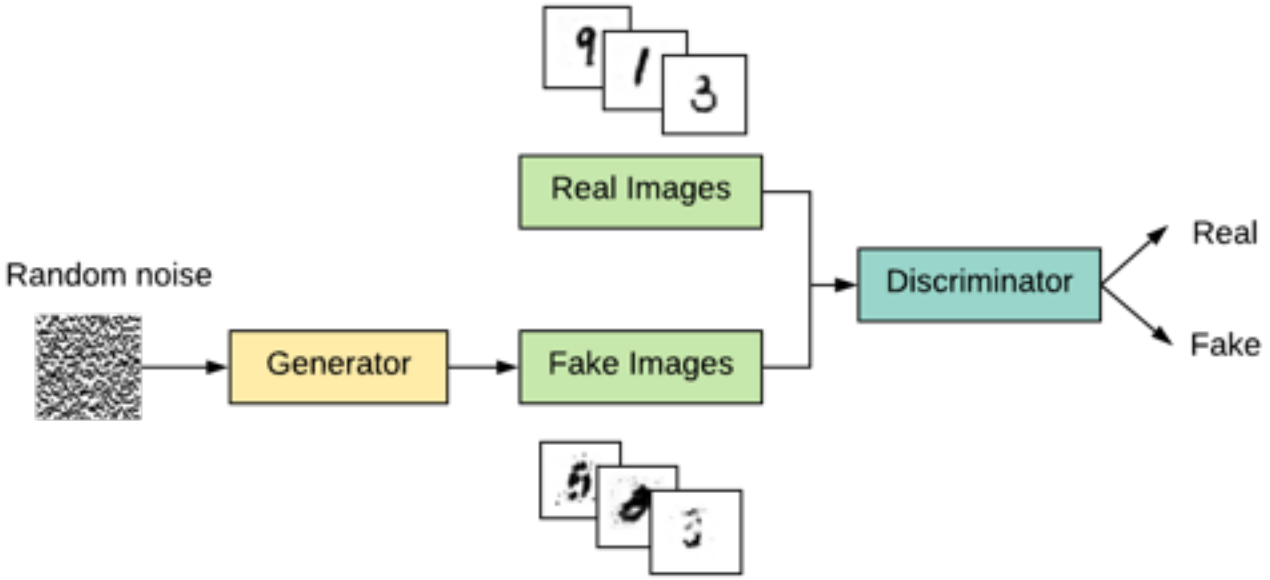
- Train function to train the GAN network
  - Step 3.6: Iterate repeatedly until the program is completed

```
130     for i in range(1, epochs+1):
131       print("Epoch %d" %i)
132
133       for _ in tqdm(range(batch_size)):
134         # Generate fake images from random noiset
135         noise= np.random.normal(0,1, (batch_size, 100))
136         fake_images = generator.predict(noise)
137
138         # Select a random batch of real images from MNIST
139         real_images = X_train[np.random.randint(0, X_train.shape[0], batch_size)]
140
141         # Labels for fake and real images
142         label_fake = np.zeros(batch_size)
143         label_real = np.ones(batch_size)
144
145         # Concatenate fake and real images
146         X = np.concatenate([fake_images, real_images])
147         y = np.concatenate([label_fake, label_real])
148
149         # Train the discriminator
150         discriminator.trainable=True
151         discriminator.train_on_batch(X, y)
152
153         # Train the generator/chained GAN model (with frozen weights in discriminator)
154         discriminator.trainable=False
155         GAN.train_on_batch(noise, label_real)
156
```



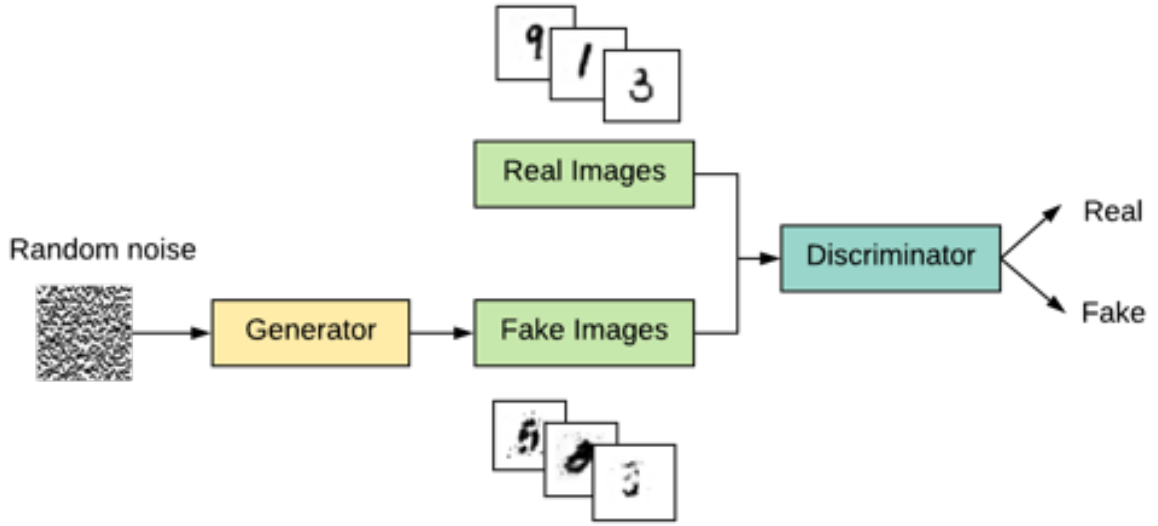
Build a Generative Adversarial Network

- What happened in each iteration?
  1. Generator uses its latest weight to generate a bunch of fake pictures
  2. Discrminator continuously trains and updates its weight according to the real label of the true and false pictures until it can successfully identify the true and false pictures



Build a Generative Adversarial Network

- What happened in each iteration?
  3. The discriminator weight is fixed and no longer changes. The generator uses the latest discrimintor to continuously train its own weights, and finally makes the discriminator identify fake pictures as real pictures
  4. Through each iteration, discrimintor and generator become more and more mature... and later reached a dynamic balance



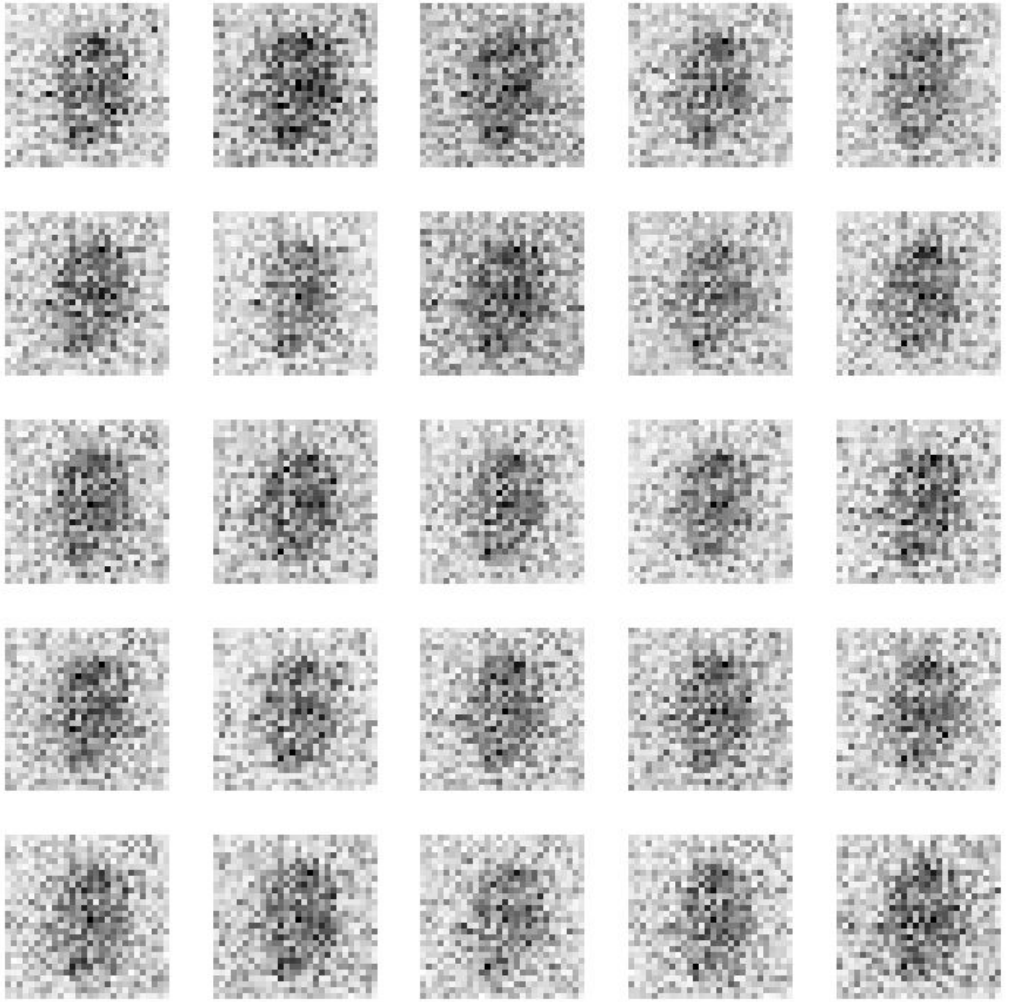
Build a Generative Adversarial Network

- Run the program

```
100%|██████████| 128/128 [00:02<00:00, 42.95it/s]
  4%|▍         | 5/128 [00:00<00:02, 43.35it/s]Epoch 3
100%|██████████| 128/128 [00:02<00:00, 42.75it/s]
  4%|▍         | 5/128 [00:00<00:02, 43.01it/s]Epoch 4
100%|██████████| 128/128 [00:02<00:00, 43.07it/s]
  4%|▍         | 5/128 [00:00<00:02, 44.25it/s]Epoch 5
100%|██████████| 128/128 [00:03<00:00, 42.61it/s]
  4%|▍         | 5/128 [00:00<00:02, 43.43it/s]Epoch 6
100%|██████████| 128/128 [00:02<00:00, 43.03it/s]
```

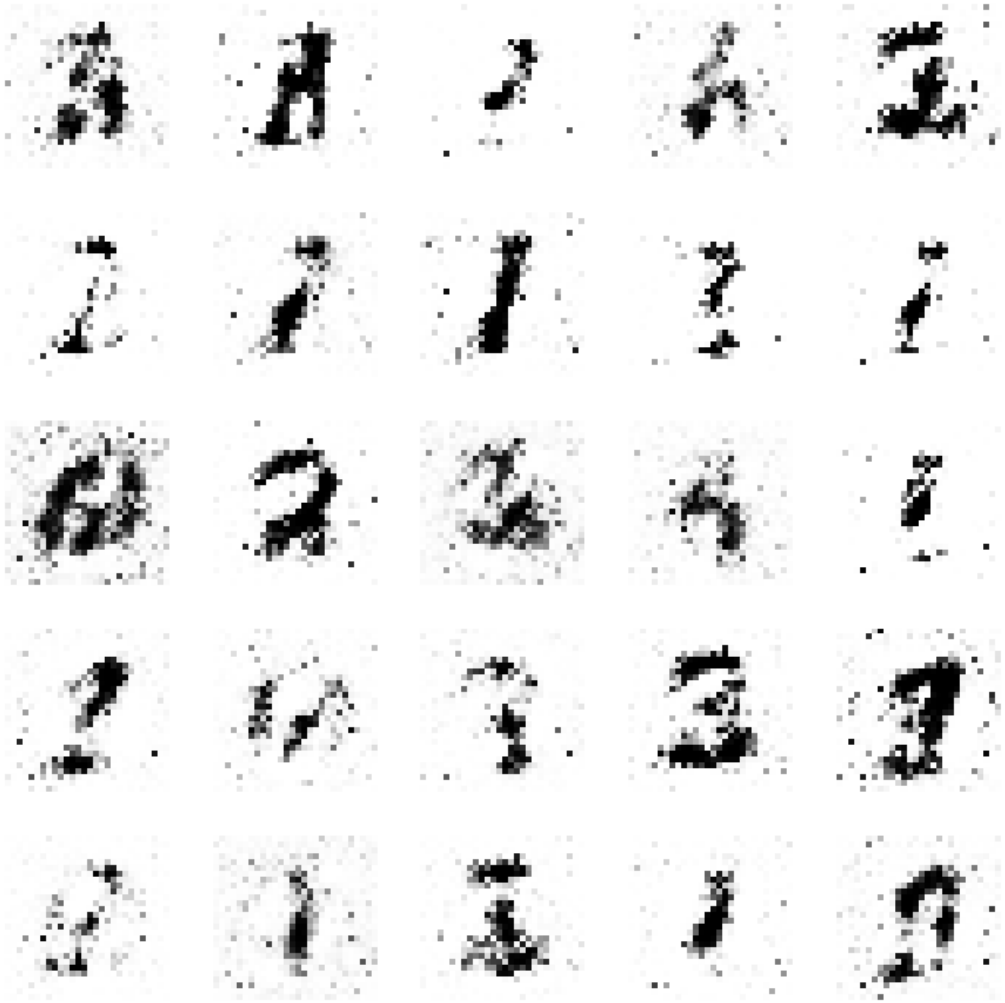
Build a Generative Adversarial Network

- Iteration 1



Build a Generative Adversarial Network

- Iteration 10



Build a Generative Adversarial Network

- Iteration 50

Build a Generative Adversarial Network

- Iteration 100