Lab 7 - st121413

1. Reproduce the vanilla GAN and DCGAN results on MNIST and CIFAR. Get the training and test loss for the generator and discriminator over time, plot them, and interpret them.

2. Develop your own GAN to model data generated as follows: You should create a PyTorch DataSet that generates the 2D data in the **init**() method, outputs a sample in the **getitem**() method, and returns the dataset size in the **len**() method. Use the vanilla GAN approach above with an appropriate structure for the generator. Can your GAN generate a convincing facsimile of a set of samples from the actual distribution?

1. Use the DCGAN (or an improvement to it) to build a generator for a face image set of your choice. Can you get realistic faces that are not in the training set?

1. Reproduce the VanillaGAN and DGAN

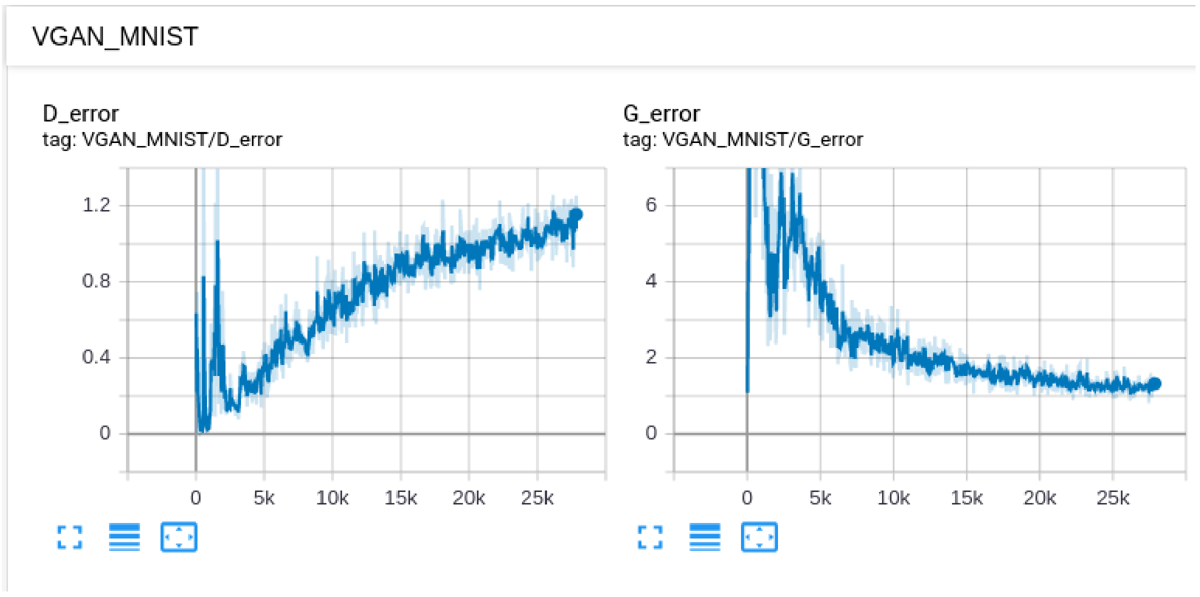
The proof of reproducing the lab manual can be seen from the series of images from TensorboardX below.

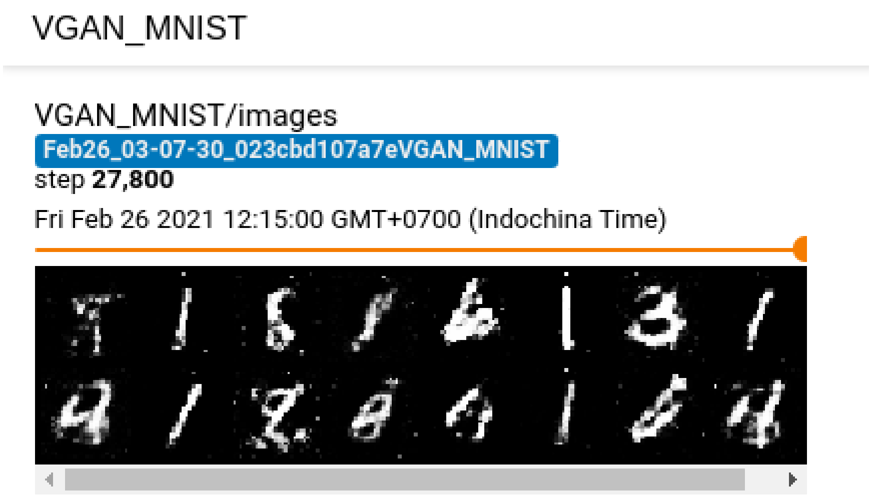
VGAN - MNIST

Loss: Discriminator Loss is growning while the Generator loss is decreasing.

This could be that the objective of Discriminator is trying to classify a real data from fake one. Once the generator gets better at faking the data, a Discriminator could not classify the real fron the fake anymore.

The fake data generated at iteration 27,800 is also shown below.




VGAN_MNIST

VGAN_MNIST/images
Feb26_03-07-30_023cbd107a7eVGAN_MNIST
step **27,800**
Fri Feb 26 2021 12:15:00 GMT+0700 (Indochina Time)

DCGAN - CIFAR10

My DCGAN has totally different story from VanillaGAN.

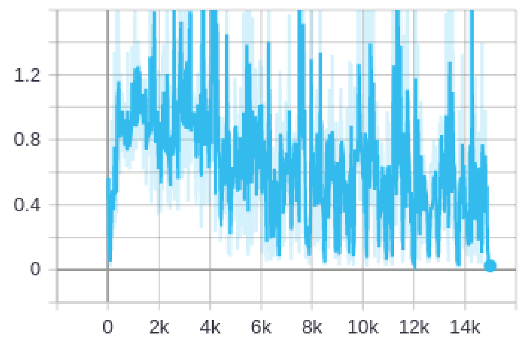
The losses of Discriminator is all over the place. There is one iteration that the loss is very high and after that is alternate between super high and super low (0.9 or 0.0001)

The same case is applied to Generator Loss. It is reduced until that iteration and start to jump around.

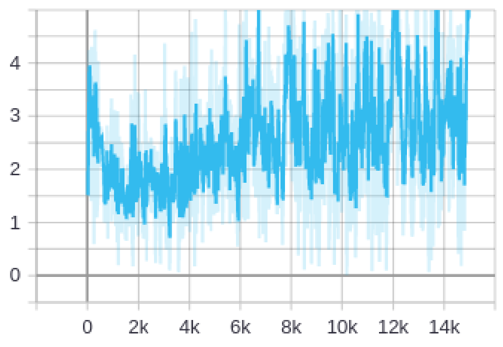
This could the unstable problem we talked about in WGAN.

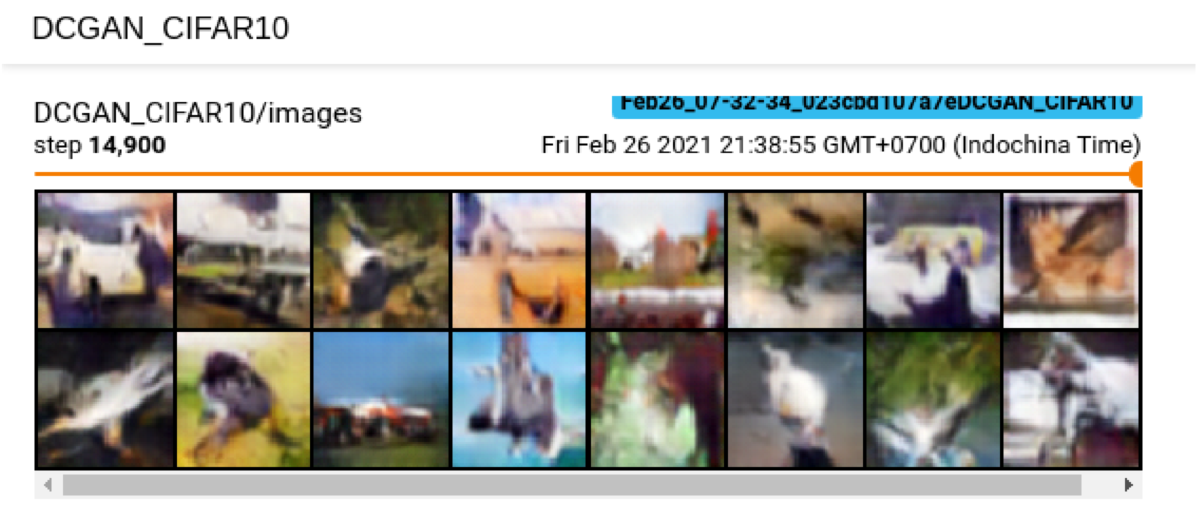
DCGAN_CIFAR10

D_error
tag: DCGAN_CIFAR10/D_error



G_error
tag: DCGAN_CIFAR10/G_error




DCGAN_CIFAR10
DCGAN_CIFAR10/images
step 14,900
Feb26_07-32-34_023cbd107a7eDCGAN_CIFAR10
Fri Feb 26 2021 21:38:55 GMT+0700 (Indochina Time)

2. VanillaGAN on SNAKE

First, here is my snake dataset

In [ ]:

```python
import torch
class Snake(torch.utils.data.Dataset):
    def __init__(self, size = 100000):
        import numpy as np
        theta = torch.rand(size) * 2 * np.pi
        # torch.RAND
        # torch.rand(*size, *, out=None, dtype=None, layout=np.strided, devic
        # Returns a tensor filled with random numbers from a uniform distribu
        r = torch.randn(size)
        # np.RANDN
        # np.randn(*size, *, out=None, dtype=None, layout=torch.strided, devi
        # Returns a tensor filled with random numbers from a normal distribut
        self.size = size
        self.x = (10 + r) * torch.cos(theta)
        mask = (theta >= (0.5 * np.pi)) & (theta <= (1.5 * np.pi))
        offset = torch.where(mask, torch.tensor(10.0), torch.tensor(-10.0))
        self.y = (10 + r) * torch.sin(theta) + offset

        self.x = torch.reshape(self.x, (1,-1))
        self.y = torch.reshape(self.y, (1,-1))
        self.data = torch.cat([ self.x, self.y], 0 ).T

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        return self.size
```

Second, about fixing VanillaGAN.

At first, I change the number of `in feature` of Discriminator to 2 and Generator `out feature` to 2.

I ran it for a while and observed that my Generator can only output a certain range of number.

I analyzed the Generator and decided to remove the last `tanh` . This function clipped out the output and I don't want that.

After doing so, the training goes well.

This is the setup where the z (input noise) space is 100-dimension.

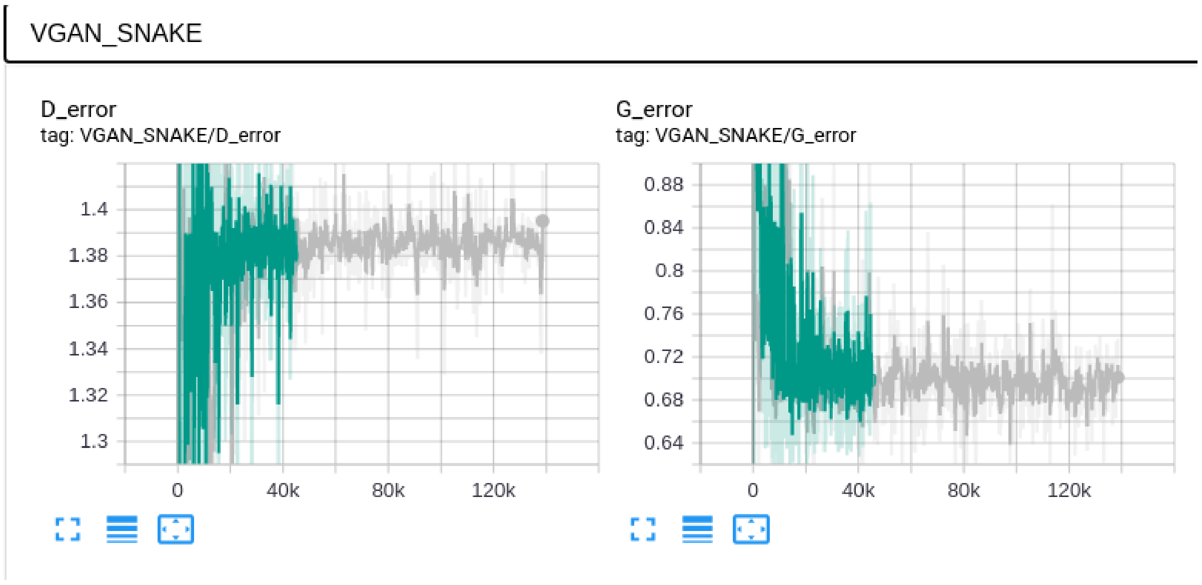
Later on, I found out from Chichan that I suppose to use z space of 2-dimension.

Therefore, I changed the noise function to generate noise of size 2 and generator to accept data of size 2.

The loss graph can be found below.

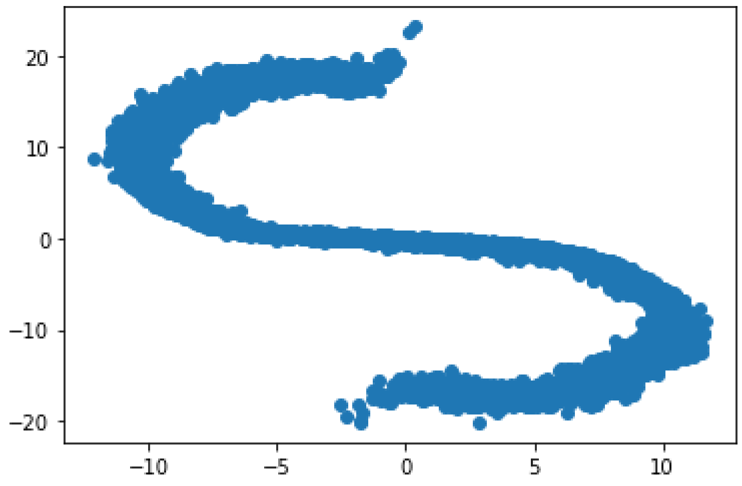
The green line: 100-dimension z

The grey line: 2-dimension z

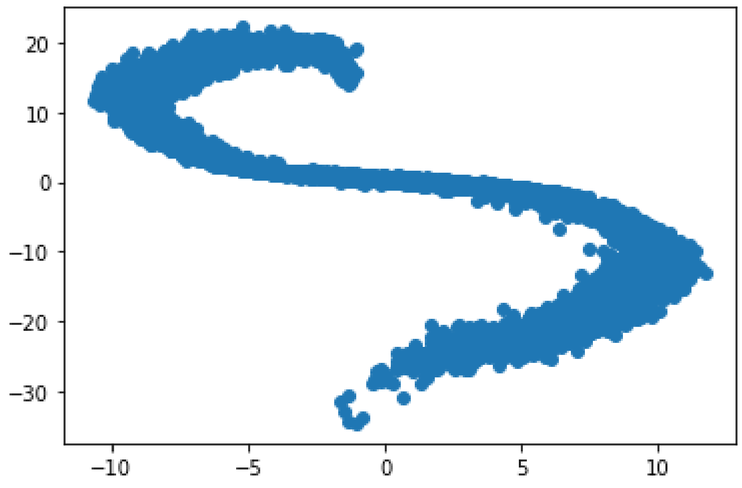


The last 2 results of generator

```
Epoch: [44/200], Batch Num: [0/1000]
Discriminator Loss: 1.3794, Generator Loss: 0.6757
D(x): 0.5167, D(G(z)): 0.5107
(10000, 2)
```



```
Epoch: [45/200], Batch Num: [0/1000]
Discriminator Loss: 1.3879, Generator Loss: 0.6789
D(x): 0.5087, D(G(z)): 0.5089
(10000, 2)
```



```
Epoch: [46/200], Batch Num: [0/1000]
Discriminator Loss: 1.3859, Generator Loss: 0.7205
D(x): 0.4892, D(G(z)): 0.4878
```

3. DCGAN on Face (CelebA)

I choosed to use torchvision dataset named CelebA because it already there in the library.

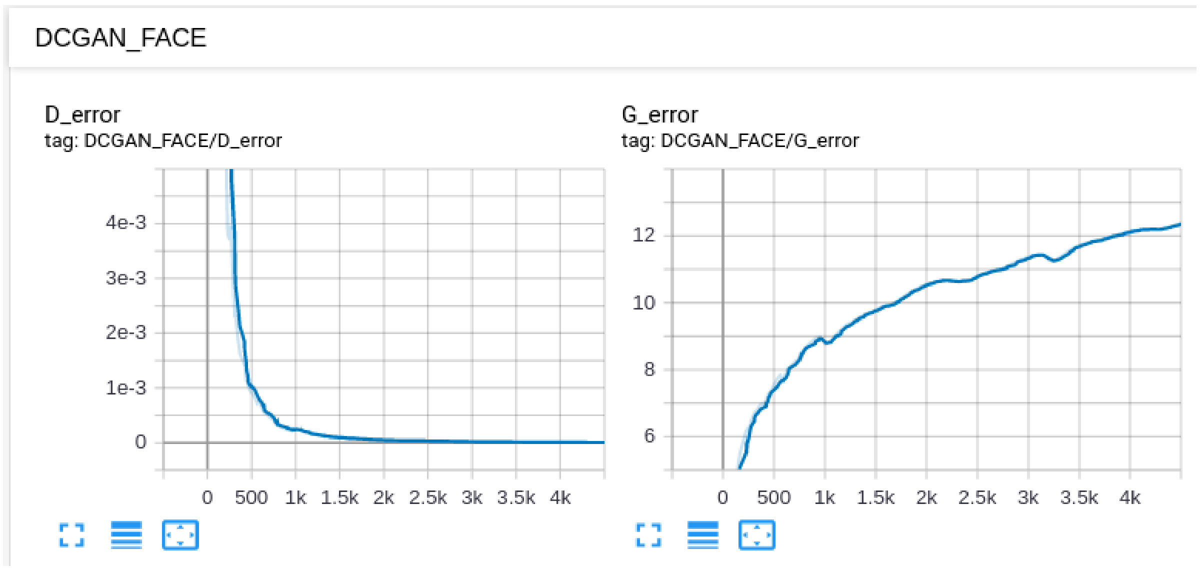
The downloading procedure has some issue but I managed to make it working.

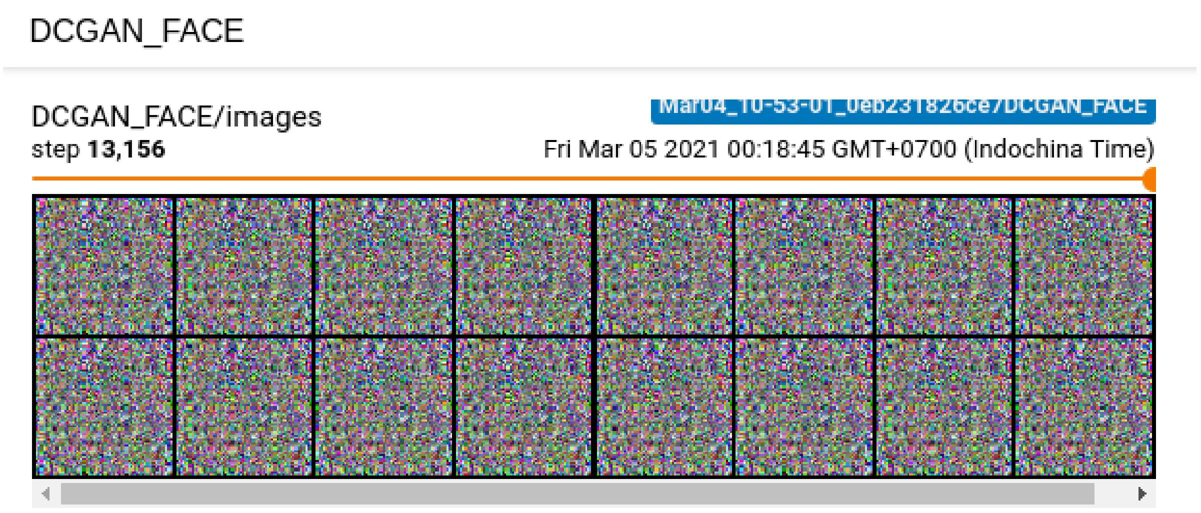
I modify not thing from DCGAN and just run the training because I want to get this result as a baseline.

My docker decided to not cooperate and vanished multiple times.

At the time of writing, my model is still training very slowly.

I share you my current result.




DCGAN_FACE
DCGAN_FACE/images
Mar04_10-53-01_0eb231826ce7DCGAN_FACE
step 13,156
Fri Mar 05 2021 00:18:45 GMT+0700 (Indochina Time)

From my analysis, I think the dataset is not only contain faces but rather an image of Celebrity with poses. The varity is just to big for this small Generator to learn.

Since the Generator failed to output the image, the discriminator has en easy time seperate real data from fake data.