Celebrity Faces Generation and Optimization using DCGAN

(Deep convolutional Generative Adversarial Network)

AUTHORS: ZHIXIA ZHANG AND LINGFENG ZHAO

ECE-GY 7143 A: ADVANCED MACHINE LEARNING

PROF. ANNA CHOROMANSKA

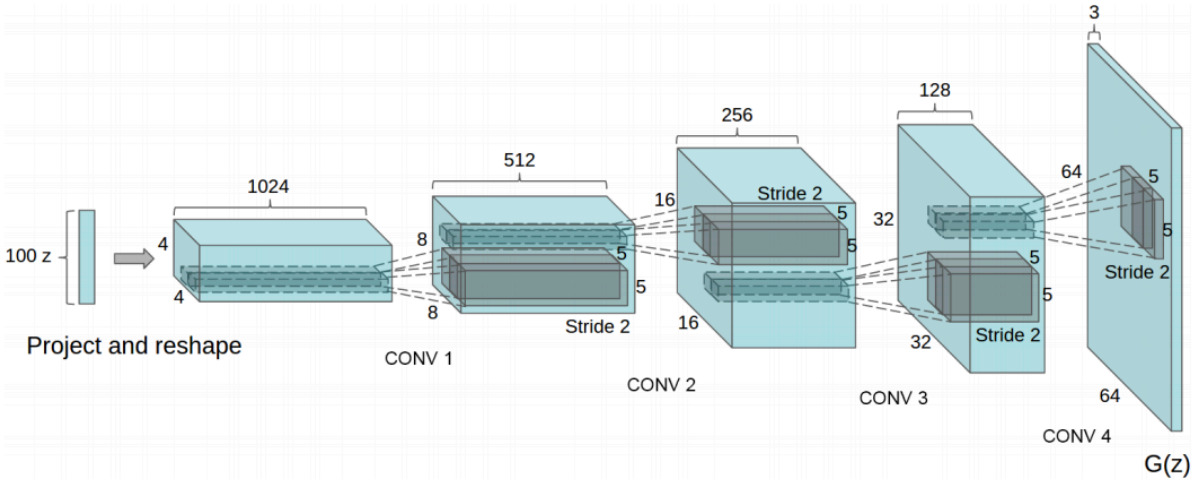
NYU TANDON SCHOOL OF ENGINEERING

Outline

- Project Introduction
- Demo on process how DCGAN works
- Experimental Network establishment and Loss function application
- Initial Training and examples
- Optimization
- Conclusion

Introduction

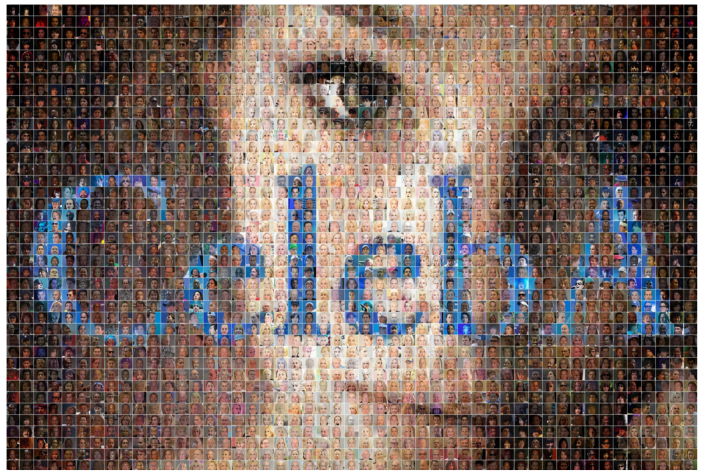
- DCGAN is one of the popular and successful network design for GAN.
- It mainly composes of convolution layers without max pooling or fully connected layers.
- It uses convolutional stride and transposed convolution for the downsampling and the upsampling.



GAN generator used for Large-scale Scene Understanding (LSUN).

Dataset and Application

- Small-scale(hundreds or thousands) dataset cannot reach Generalization and generate photos close to training samples
- Large-scale Celebrity Faces Attributes (CelebA) Dataset
- To generate some human faces from DCGAN which trained via CelebA (**202,599** number of **face images**)



True image

Fake image

Outline

- Project Introduction and purpose.
- Demo on process how DCGAN works
- Experimental Network establishment and Loss function application
- Initial Training and examples
- Optimization
- Conclusion

Simple Example using DCGAN

- Given some true data (artist works) distribution.

  upper bound: 2x**2+1

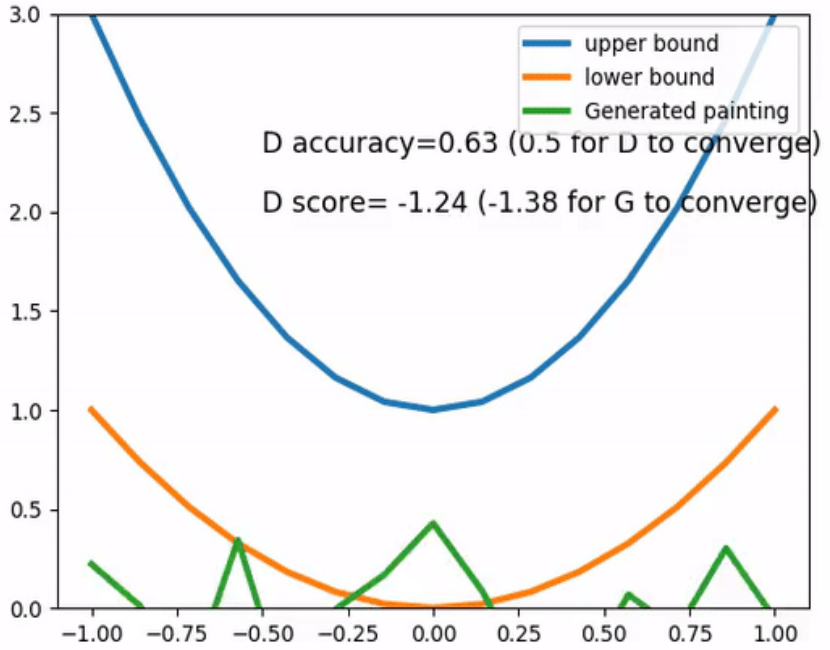
  lower bound: x**2

- Want to generate some points (painter's paintings) to fit the distribution:

$$y \approx f(x) \Rightarrow ax^2 + c$$

  a belongs to [1,2], c belongs to [0,1]

- Plot the Learning process.



Outline

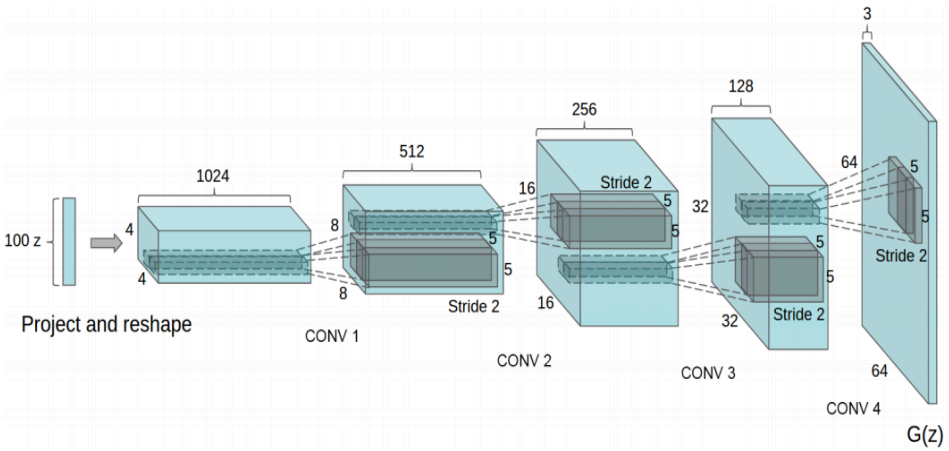
- Project Introduction and purpose.
- Demo on process how DCGAN works
- Experimental Network establishment and Loss function application
- Initial Training and examples
- Optimization
- Conclusion

Deep convolutional Network

- Learn from the paper

  Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
  - Alec Radford & Luke Metz, Soumith Chintala , 2016

- The Generator:



Upsampling !

```
Generator(
 (main): Sequential(
   (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
   (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (2): ReLU(inplace=True)
   (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (5): ReLU(inplace=True)
   (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (8): ReLU(inplace=True)
   (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (11): ReLU(inplace=True)
   (12): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (13): Tanh()
)
```

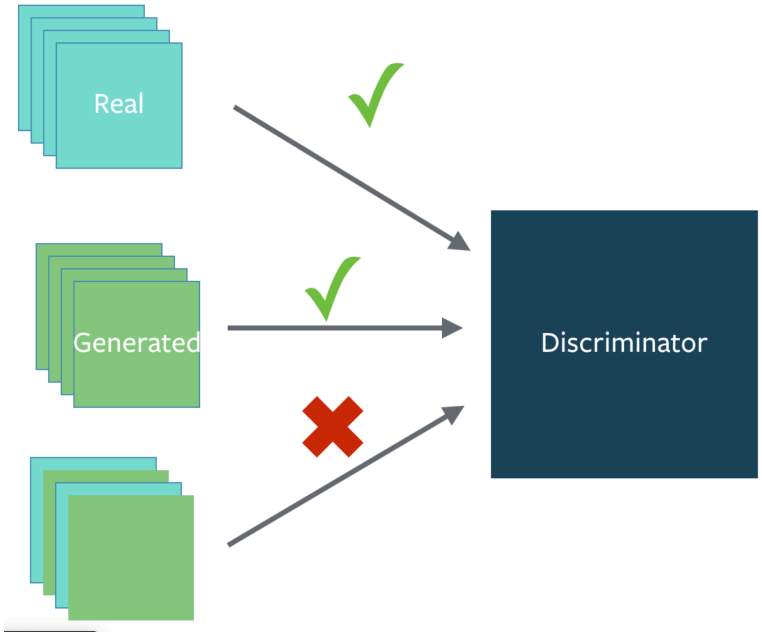
Deep convolutional Network

- The Discriminator : for each sample, subtract mean and divide by standard deviation(**Batch Normalization**)
- Construct different mini-batches for real and fake,

i.e. each mini-batch needs to contain only all real images or all generated images.

Downsampling !

```
Discriminator(
 (main): Sequential(
   (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (1): LeakyReLU(negative_slope=0.2, inplace=True)
   (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   (4): LeakyReLU(negative_slope=0.2, inplace=True)
   (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   (7): LeakyReLU(negative_slope=0.2, inplace=True)
   (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   (10): LeakyReLU(negative_slope=0.2, inplace=True)
   (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
   (12): Sigmoid()
 )
```



Some tricks and Loss function

- ❑ Normalize the inputs
- ❑ Avoid Sparse Gradients: ReLU, MaxPool

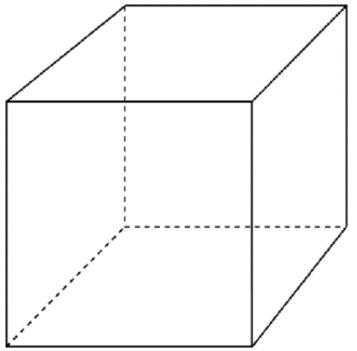
  Using LeakyReLU = good (in both G and D)
- ❑ Use the ADAM Optimizer

  or Use SGD for discriminator and ADAM for generator
- ❑ Use a spherical Z
- ❑ Loss function:

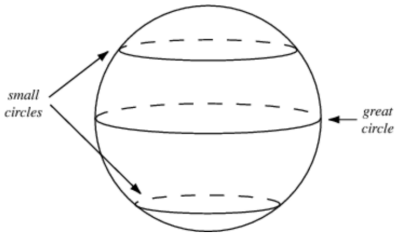
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

```
D_loss = - torch.mean(torch.log(prob_artist0) + torch.log(1. - prob_artist1))
G_loss = torch.mean(torch.log(1. - prob_artist1))
```

- Dont sample from a Uniform distribution



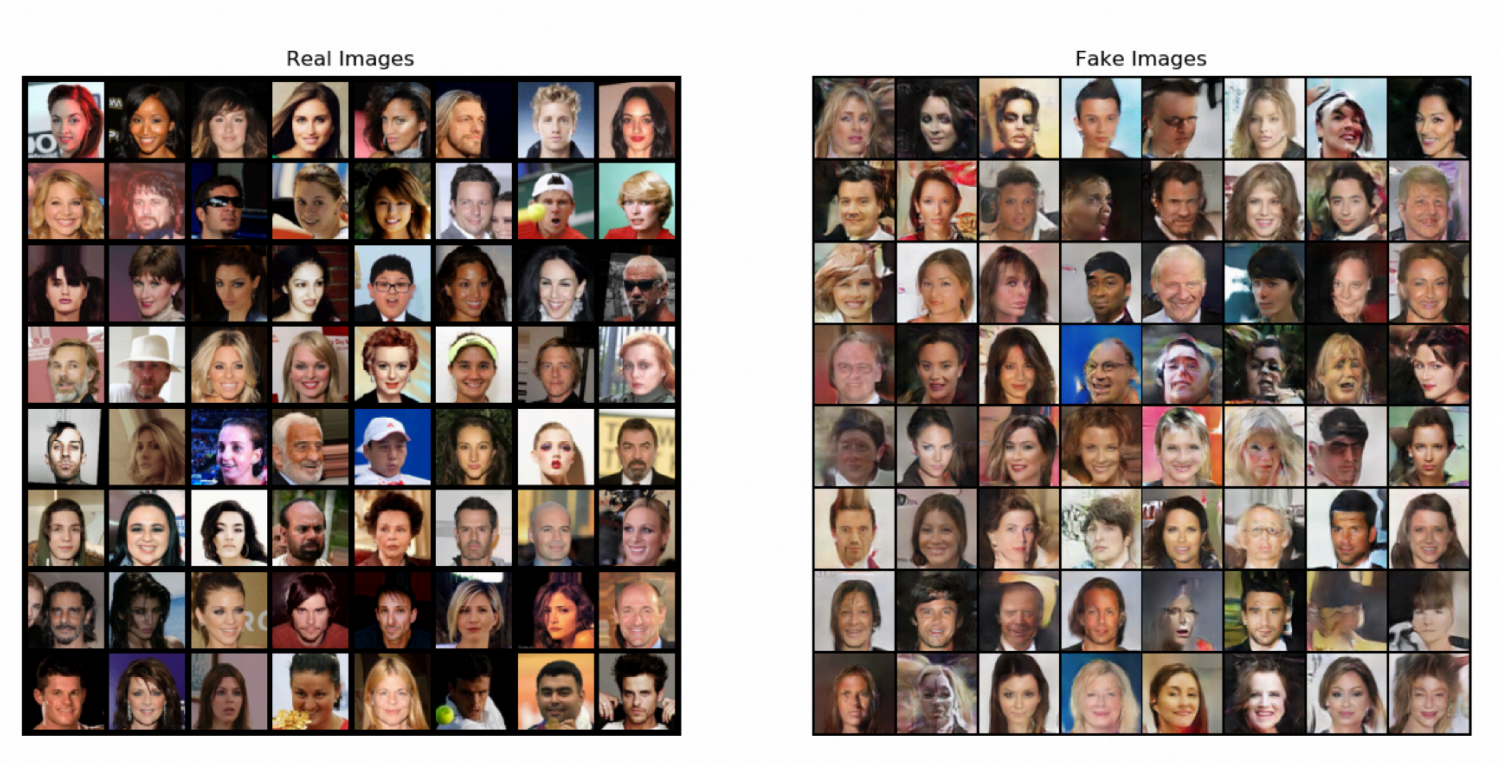
- Sample from a gaussian distribution



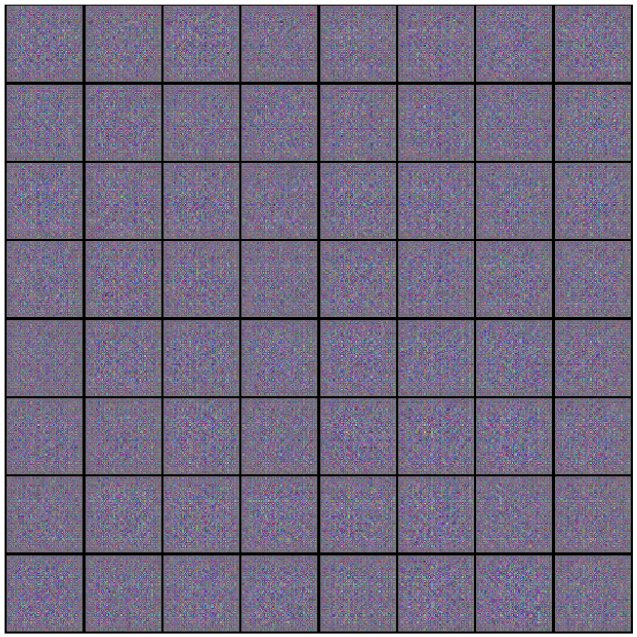
Outline

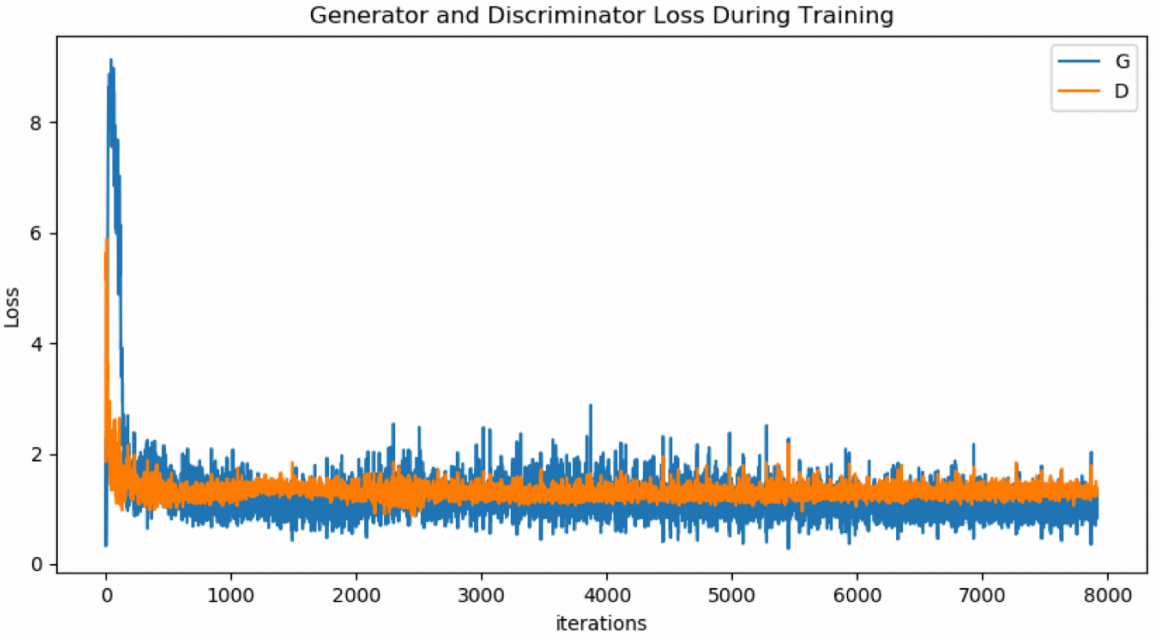
- ❑ Project Introduction and purpose.
- ❑ Demo on process how DCGAN works
- ❑ Experimental Network establishment and Loss function application
- ❑ Initial Training and examples
- ❑ Optimization
- ❑ Conclusion

Some Outputs from Initial Training

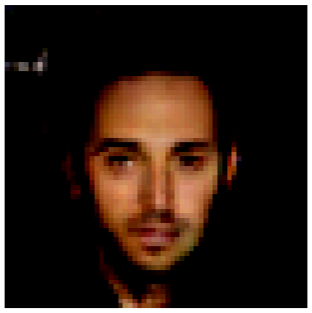
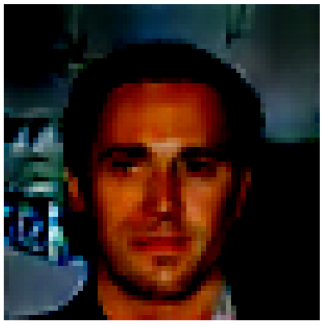


Some Outputs from Initial Training

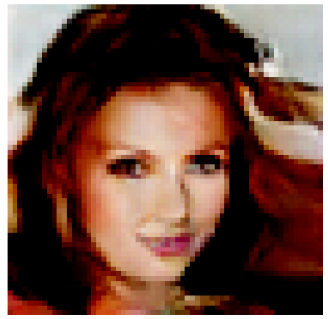




Some Outputs from Initial Training





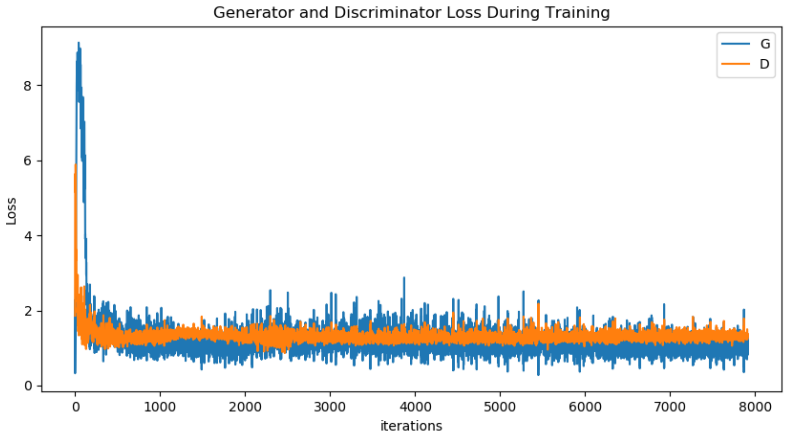
Outline

- ❑ Project Introduction and purpose.
- ❑ Demo on process how DCGAN works
- ❑ Experimental Network establishment and Loss function application
- ❑ Initial Training and examples
- ❑ Optimization
- ❑ Conclusion

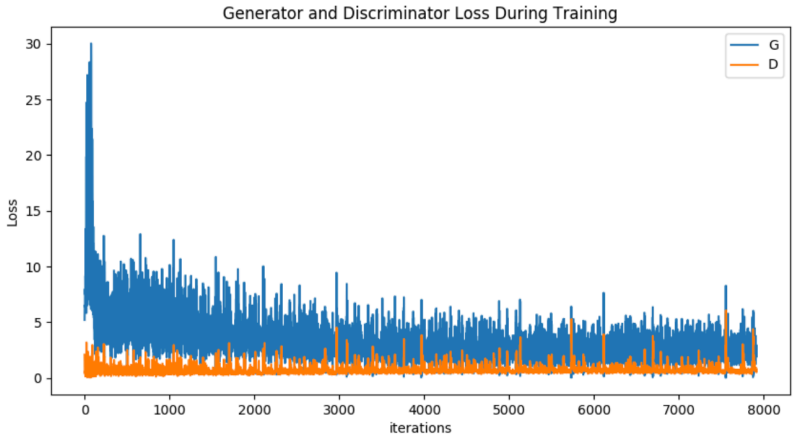
Optimization

- Compare to use same learning rate on G and D. We changed our strategy to use different lrs on G and D, respectively.
- Compare to Original method, we find out the best learning rates for G and D is 0.0002 and 0.00005 respectively.

Different learning rate



Same learning rate



Outline

- Project Introduction and purpose.
- Demo on process how DCGAN works
- Experimental Network establishment and Loss function application
- Initial Training and examples
- Optimization
- Conclusion

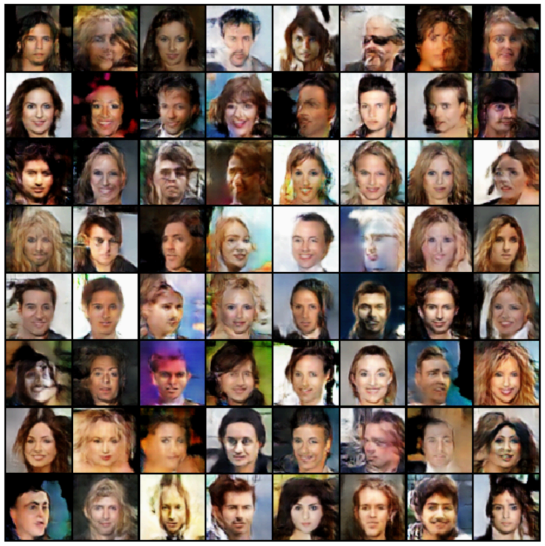
Conclusion

- We got a better output quality on same model but in different learning rate.
- In addition that we also tried to use other different strategies:

  like hybrid network, using pooling

  using other input noise distribution….
- G: 0.0002 D: 0.00005

Better !



Thank you…..