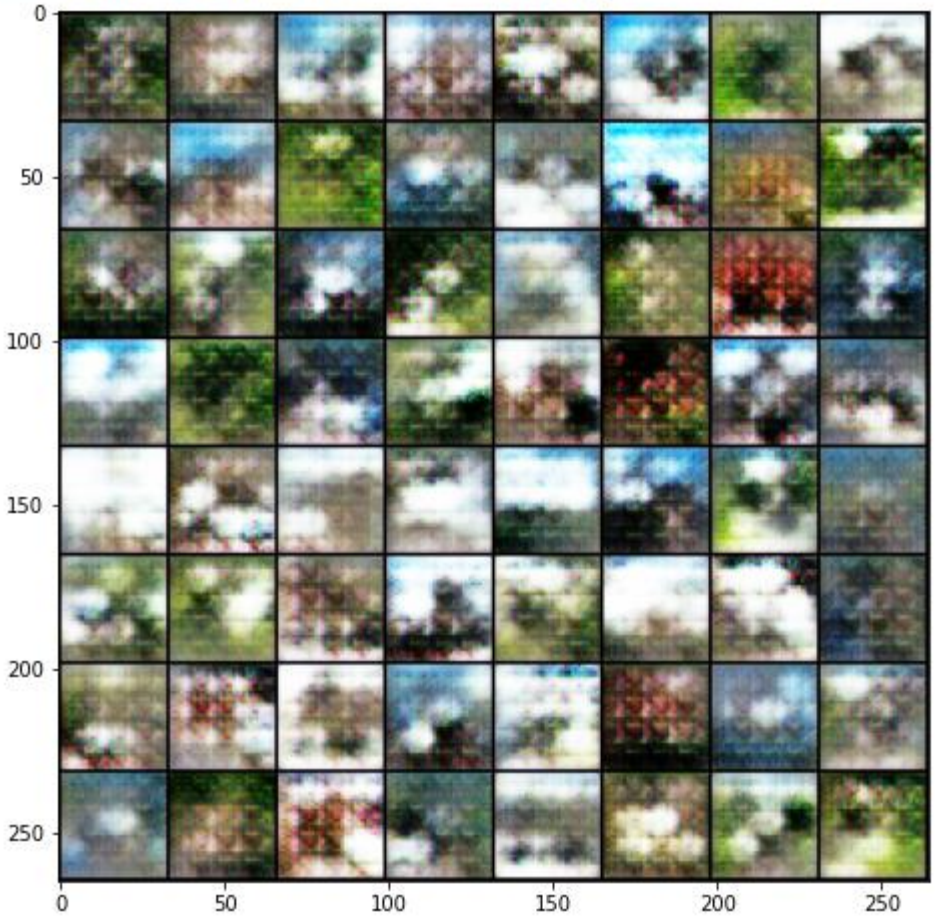
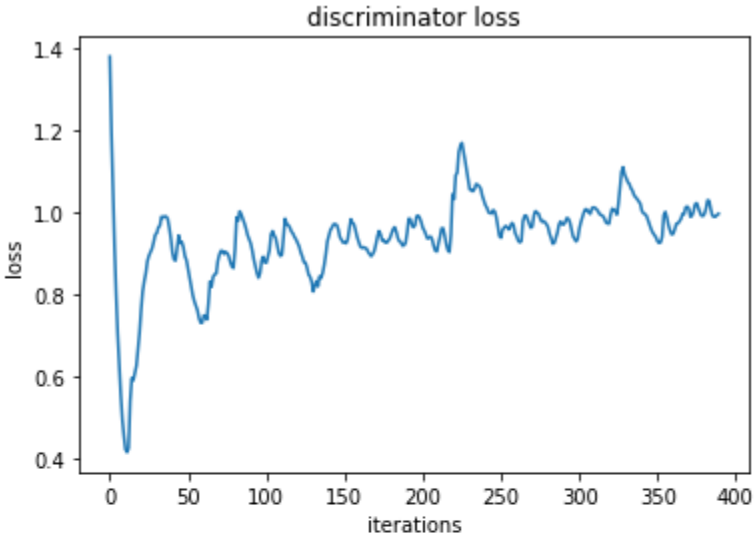
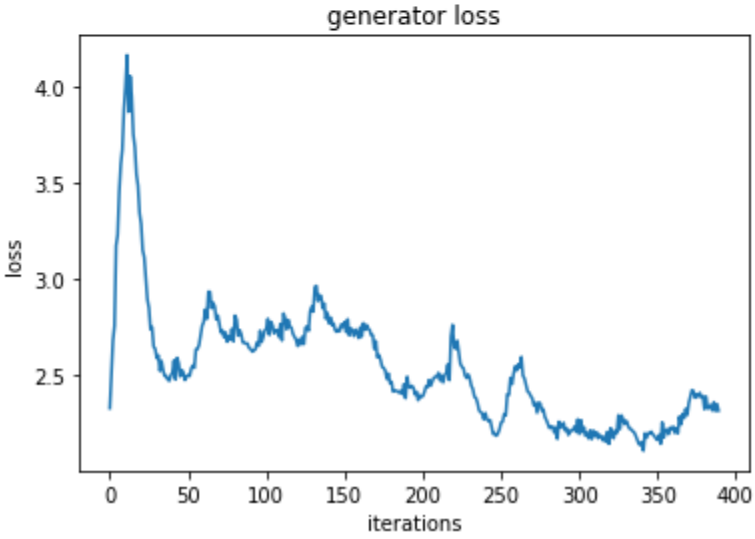
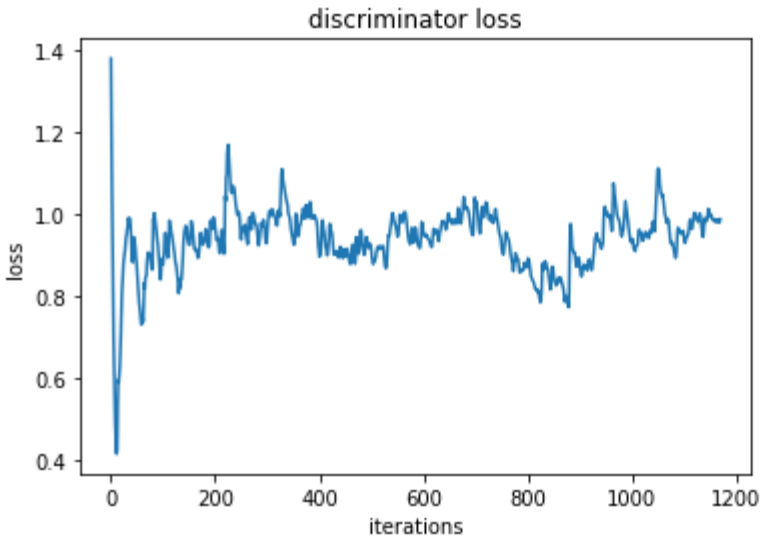
```
Start training ...
Iteration 100/9750: dis loss = 0.6176, gen loss = 3.3591
Iteration 200/9750: dis loss = 0.7728, gen loss = 2.2248
Iteration 300/9750: dis loss = 1.0694, gen loss = 3.5096
```

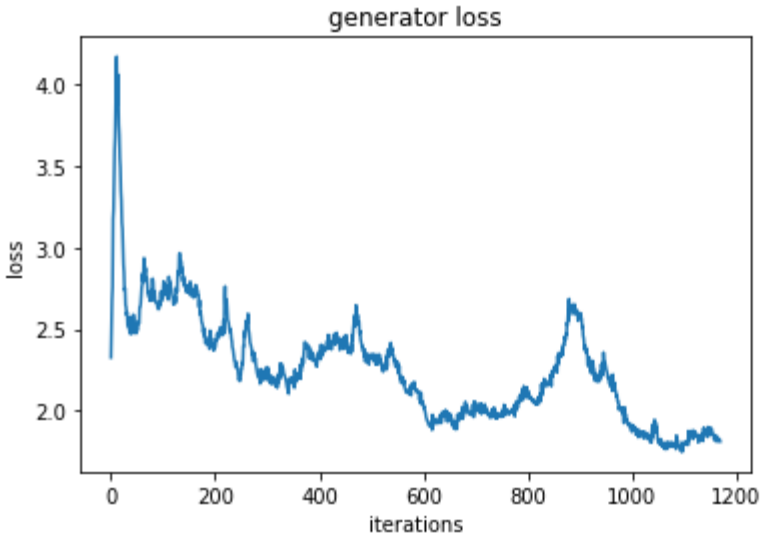



discriminator loss
loss
iterations
1.4
1.2
1.0
0.8
0.6
0.4
0
50
100
150
200
250
300
350
400

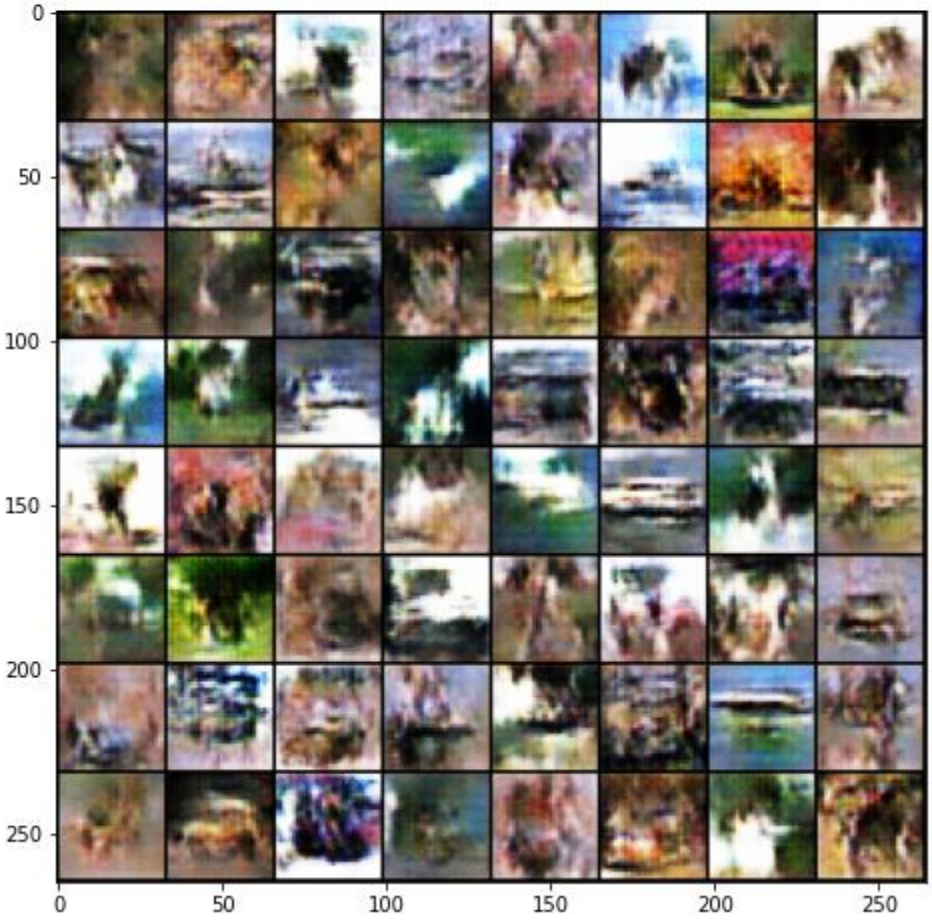

generator loss
loss
iterations
4.0
3.5
3.0
2.5
0
50
100
150
200
250
300
350
400

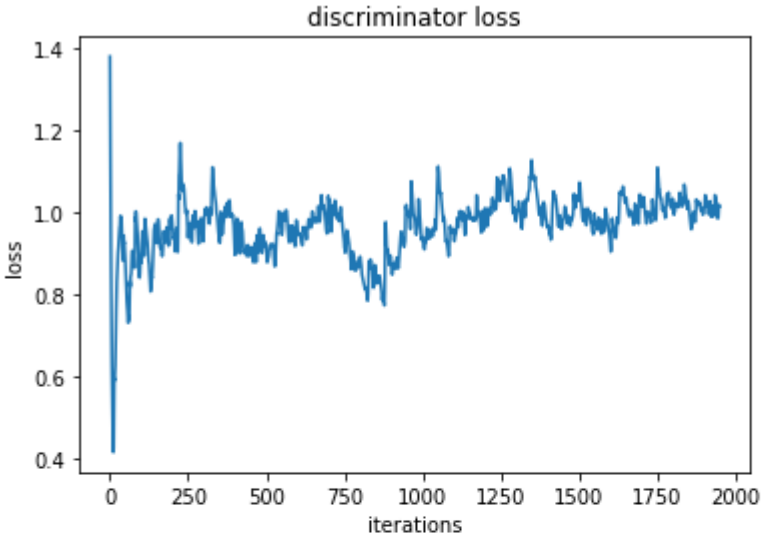
```
Iteration 400/9750: dis loss = 0.7253, gen loss = 3.1222
Iteration 500/9750: dis loss = 0.8099, gen loss = 1.9054
Iteration 600/9750: dis loss = 0.7218, gen loss = 1.7982
Iteration 700/9750: dis loss = 0.8156, gen loss = 1.3703
Iteration 800/9750: dis loss = 1.1387, gen loss = 0.9477
Iteration 900/9750: dis loss = 0.8134, gen loss = 3.0272
Iteration 1000/9750: dis loss = 1.0616, gen loss = 1.0838
Iteration 1100/9750: dis loss = 0.7368, gen loss = 2.1944
```

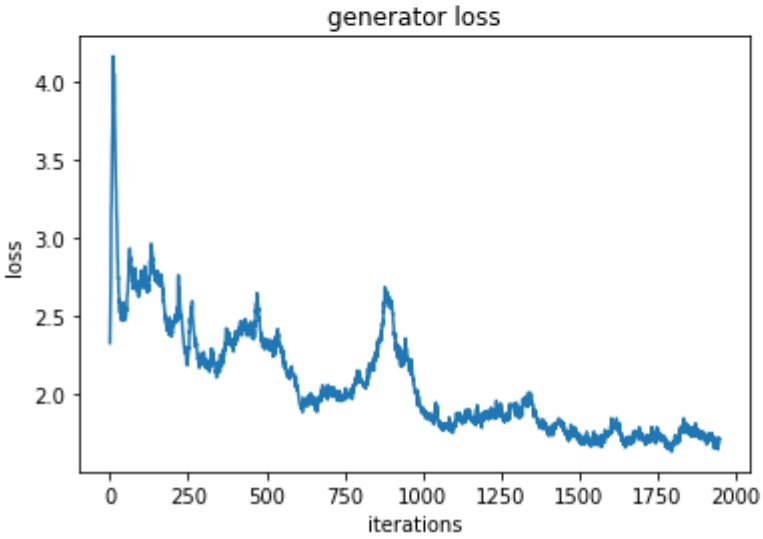

discriminator loss
loss
iterations
1.4
1.2
1.0
0.8
0.6
0.4
0
200
400
600
800
1000
1200


generator loss
loss
iterations
4.0
3.5
3.0
2.5
2.0
0
200
400
600
800
1000
1200

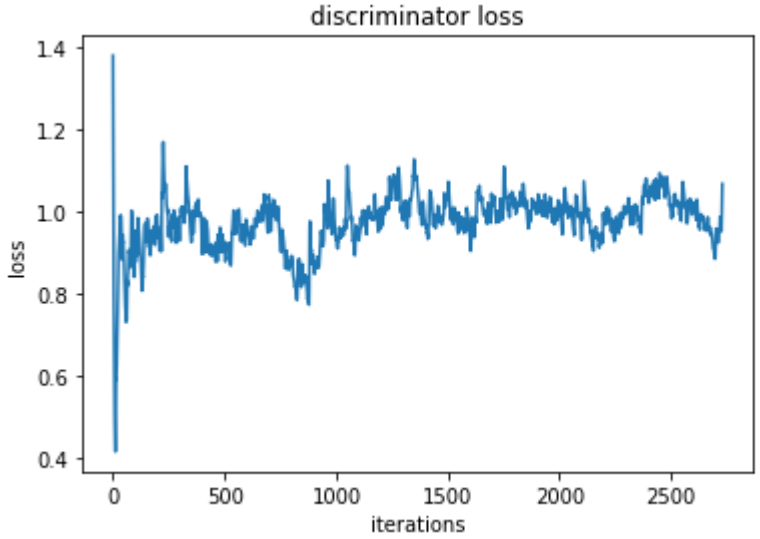
```
Iteration 1200/9750: dis loss = 1.3789, gen loss = 0.9708
Iteration 1300/9750: dis loss = 0.9279, gen loss = 3.5074
Iteration 1400/9750: dis loss = 0.9147, gen loss = 1.5573
Iteration 1500/9750: dis loss = 1.1441, gen loss = 0.7582
Iteration 1600/9750: dis loss = 0.7866, gen loss = 2.0613
Iteration 1700/9750: dis loss = 1.0532, gen loss = 1.5888
Iteration 1800/9750: dis loss = 1.2520, gen loss = 0.8263
Iteration 1900/9750: dis loss = 1.1044, gen loss = 2.4969
```

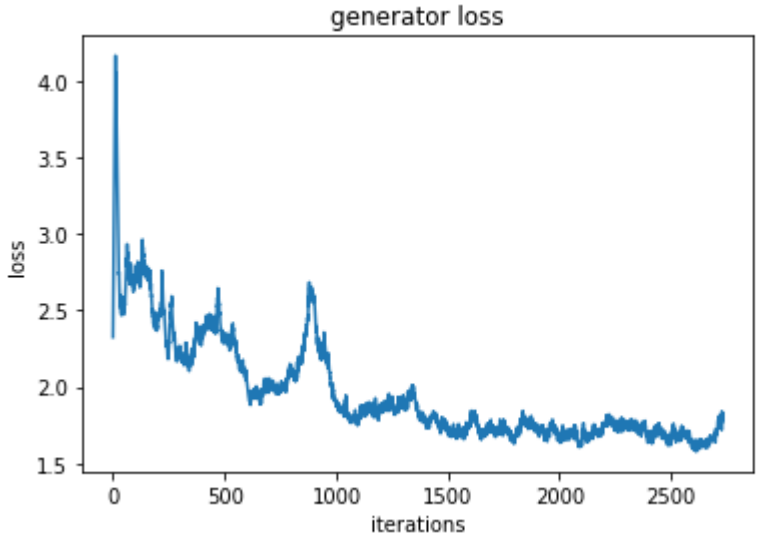



discriminator loss
loss
iterations
1.4
1.2
1.0
0.8
0.6
0.4
0
250
500
750
1000
1250
1500
1750
2000

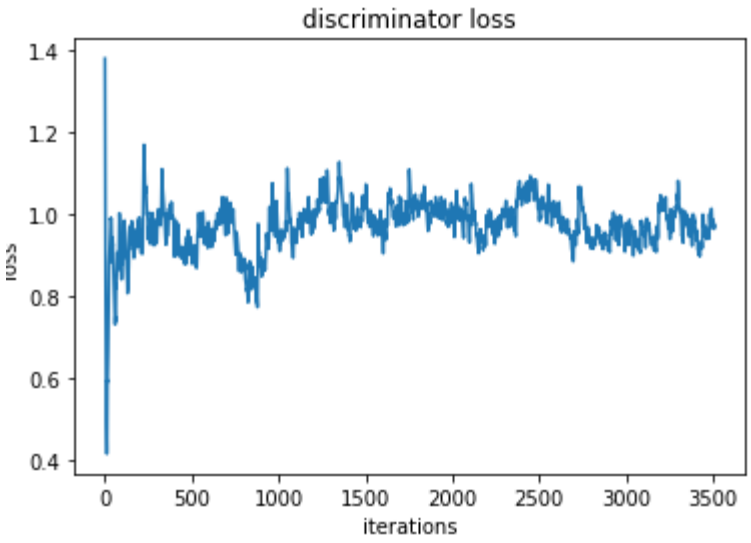

generator loss
loss
iterations
4.0
3.5
3.0
2.5
2.0
0
250
500
750
1000
1250
1500
1750
2000

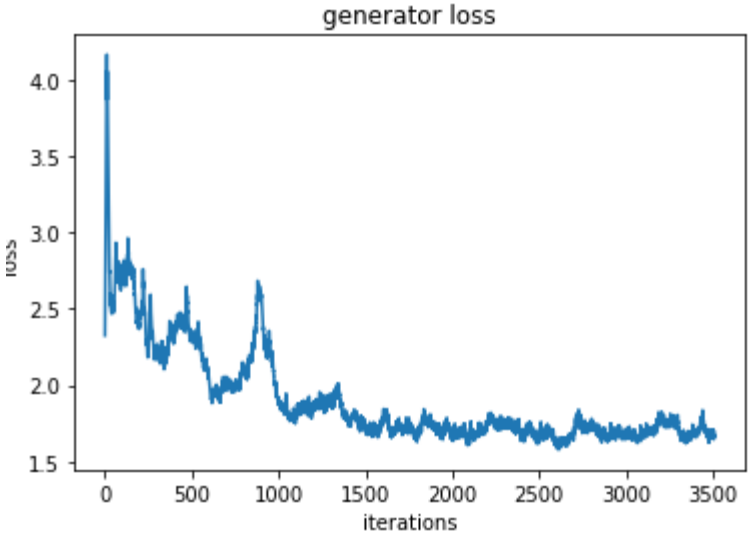
```
Iteration 2000/9750: dis loss = 0.9951, gen loss = 2.2865
Iteration 2100/9750: dis loss = 0.8373, gen loss = 0.8464
Iteration 2200/9750: dis loss = 0.8522, gen loss = 1.9641
Iteration 2300/9750: dis loss = 1.0282, gen loss = 1.3583
Iteration 2400/9750: dis loss = 0.8552, gen loss = 2.3759
Iteration 2500/9750: dis loss = 0.8866, gen loss = 2.1508
Iteration 2600/9750: dis loss = 0.9115, gen loss = 2.2170
Iteration 2700/9750: dis loss = 1.3092, gen loss = 3.3554
```


discriminator loss
loss
iterations
1.4
1.2
1.0
0.8
0.6
0.4
0
500
1000
1500
2000
2500

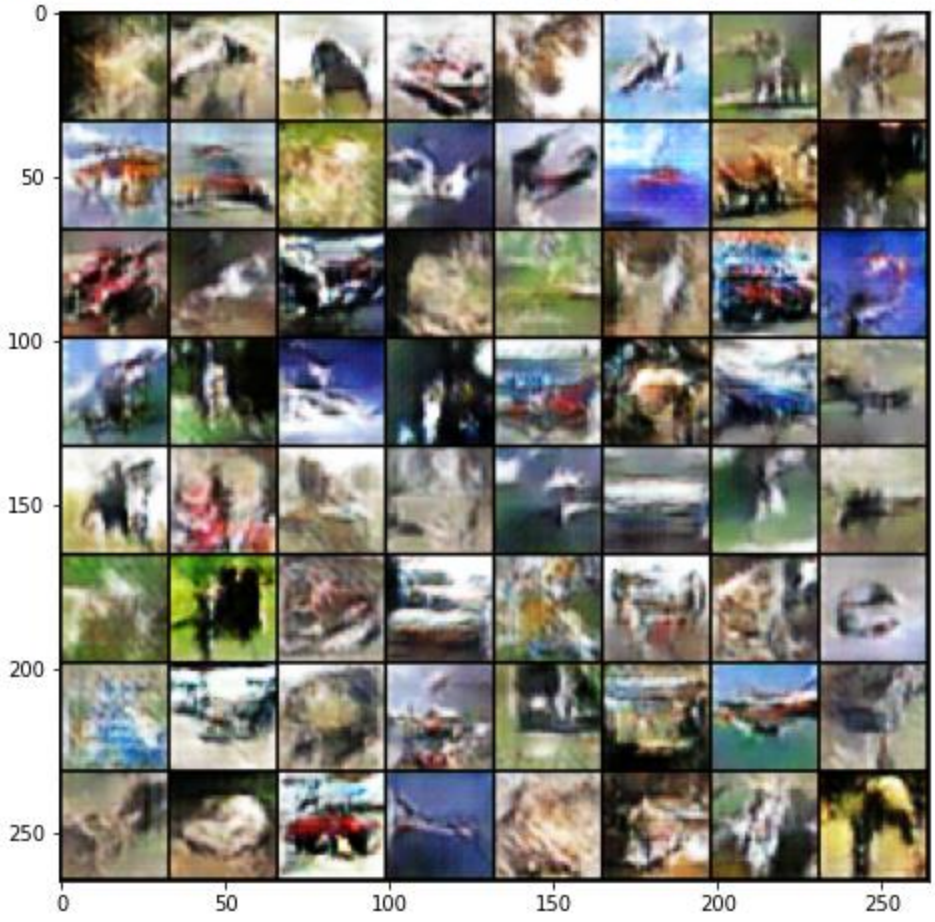

generator loss
loss
iterations
4.0
3.5
3.0
2.5
2.0
1.5
0
500
1000
1500
2000
2500

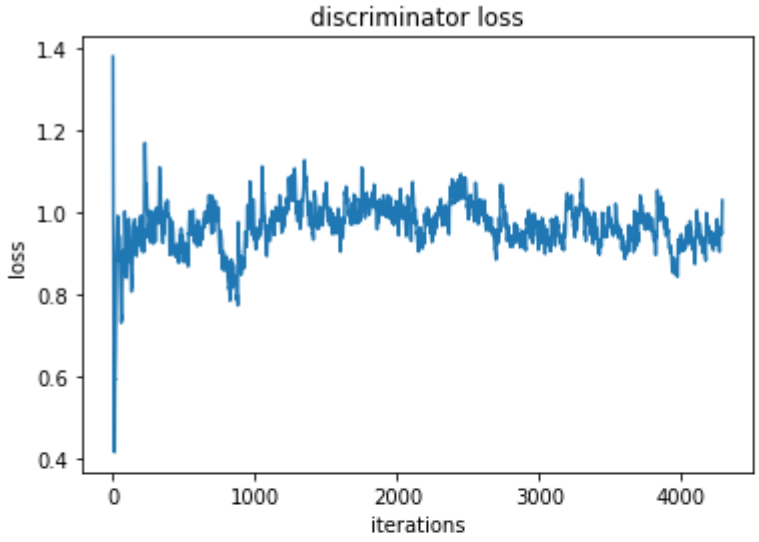
```
Iteration 2800/9750: dis loss = 0.9345, gen loss = 1.3365
Iteration 2900/9750: dis loss = 0.9152, gen loss = 1.2298
Iteration 3000/9750: dis loss = 1.3663, gen loss = 2.2866
Iteration 3100/9750: dis loss = 0.9149, gen loss = 1.9631
Iteration 3200/9750: dis loss = 0.8958, gen loss = 1.7609
Iteration 3300/9750: dis loss = 1.4451, gen loss = 2.1472
Iteration 3400/9750: dis loss = 0.8012, gen loss = 1.7414
Iteration 3500/9750: dis loss = 0.9470, gen loss = 2.3720
```

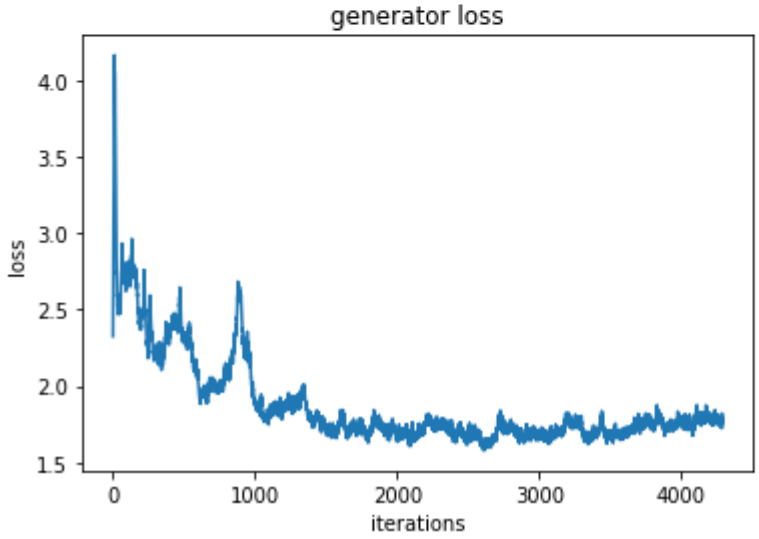

discriminator loss
1.4
1.2
1.0
0.8
0.6
0.4
loss
0
500
1000
1500
2000
2500
3000
3500
iterations


generator loss
4.0
3.5
3.0
2.5
2.0
1.5
loss
0
500
1000
1500
2000
2500
3000
3500
iterations

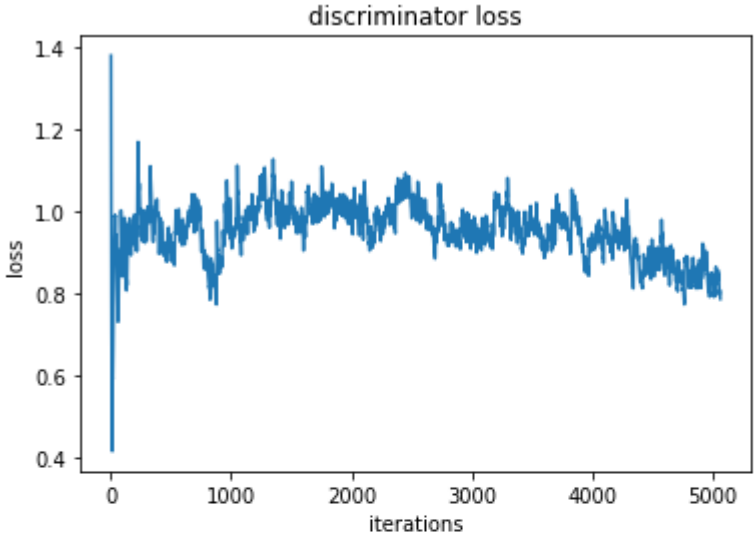
```
Iteration 3600/9750: dis loss = 0.7953, gen loss = 1.5127
Iteration 3700/9750: dis loss = 1.5730, gen loss = 0.7771
Iteration 3800/9750: dis loss = 0.8641, gen loss = 1.6214
Iteration 3900/9750: dis loss = 0.8270, gen loss = 1.3545
Iteration 4000/9750: dis loss = 1.0703, gen loss = 0.9913
Iteration 4100/9750: dis loss = 0.9495, gen loss = 0.9926
Iteration 4200/9750: dis loss = 0.8231, gen loss = 1.8931
```

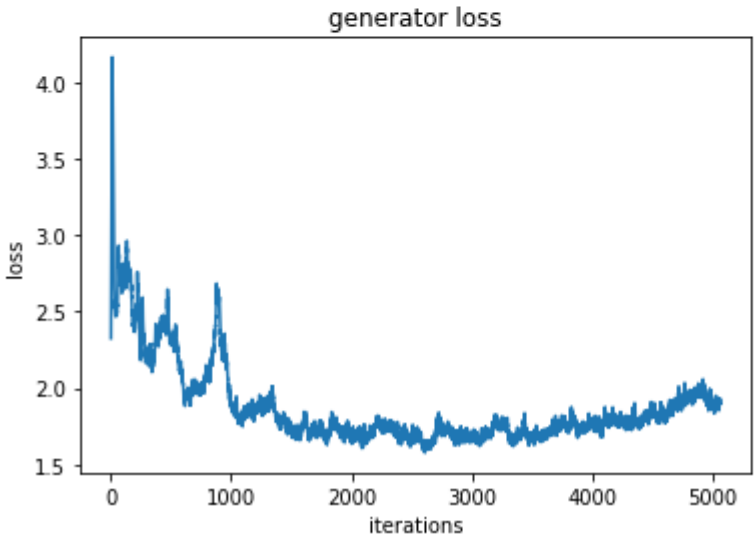



discriminator loss
1.4
1.2
1.0
0.8
0.6
0.4
loss
0
1000
2000
3000
4000
iterations

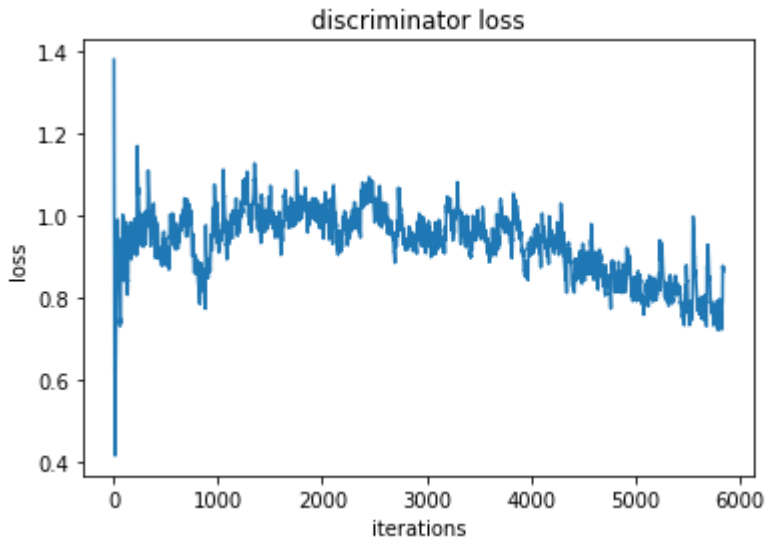

generator loss
4.0
3.5
3.0
2.5
2.0
1.5
loss
0
1000
2000
3000
4000
iterations

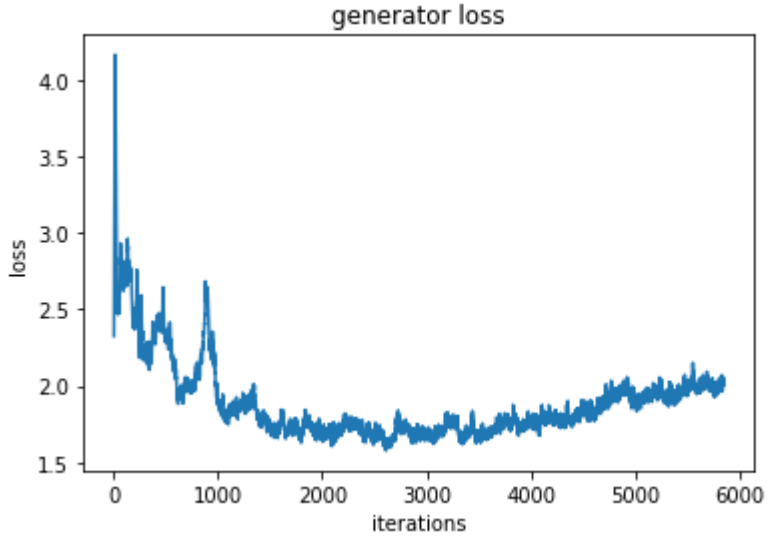
```
Iteration 4300/9750: dis loss = 0.8429, gen loss = 1.2768
Iteration 4400/9750: dis loss = 0.6928, gen loss = 2.2999
Iteration 4500/9750: dis loss = 1.2724, gen loss = 3.0335
Iteration 4600/9750: dis loss = 0.8487, gen loss = 1.5321
Iteration 4700/9750: dis loss = 0.7542, gen loss = 1.1428
Iteration 4800/9750: dis loss = 0.8766, gen loss = 1.3558
Iteration 4900/9750: dis loss = 0.7083, gen loss = 1.8890
Iteration 5000/9750: dis loss = 0.9247, gen loss = 1.1264
```


discriminator loss
loss
iterations
1.4
1.2
1.0
0.8
0.6
0.4
0
1000
2000
3000
4000
5000

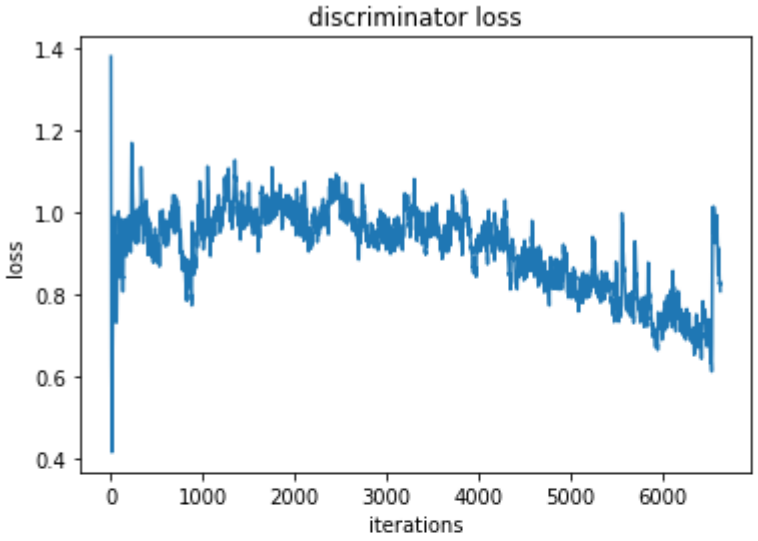

generator loss
loss
iterations
4.0
3.5
3.0
2.5
2.0
1.5
0
1000
2000
3000
4000
5000

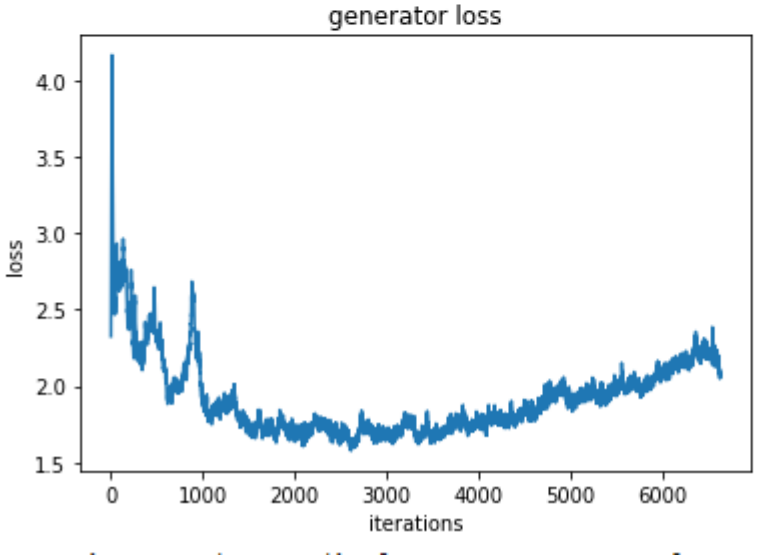
```
Iteration 5100/9750: dis loss = 0.9083, gen loss = 1.4153
Iteration 5200/9750: dis loss = 0.7424, gen loss = 1.6530
Iteration 5300/9750: dis loss = 0.7576, gen loss = 1.8695
Iteration 5400/9750: dis loss = 0.6786, gen loss = 1.8967
Iteration 5500/9750: dis loss = 0.6980, gen loss = 2.1189
Iteration 5600/9750: dis loss = 0.6798, gen loss = 2.0432
Iteration 5700/9750: dis loss = 1.0818, gen loss = 2.6546
Iteration 5800/9750: dis loss = 0.6516, gen loss = 2.4423
```


discriminator loss
1.4
1.2
1.0
0.8
0.6
0.4
loss
0
1000
2000
3000
4000
5000
6000
iterations

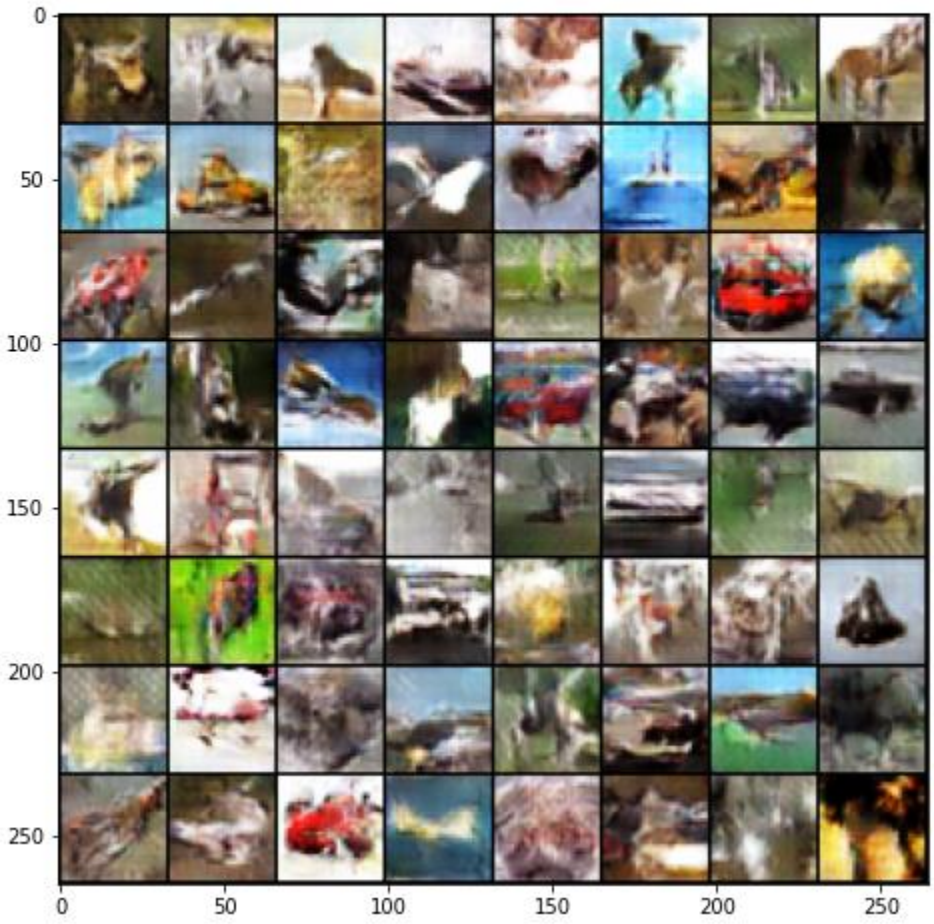

generator loss
4.0
3.5
3.0
2.5
2.0
1.5
loss
0
1000
2000
3000
4000
5000
6000
iterations

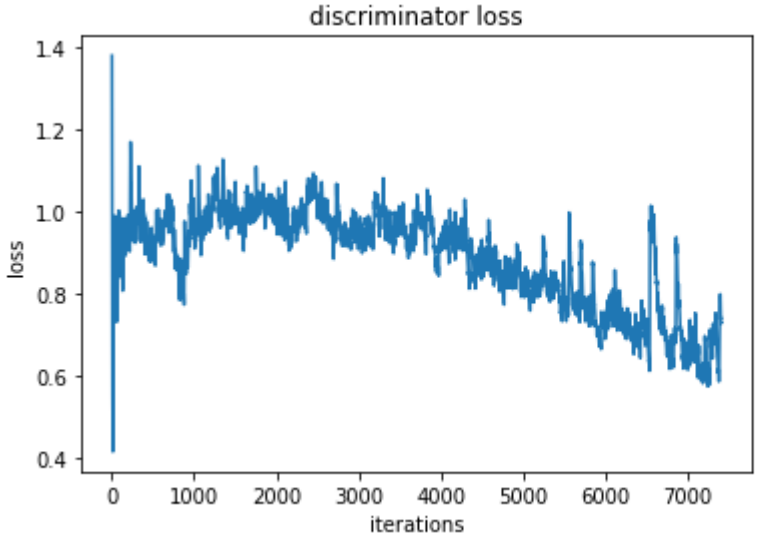
```
Iteration 5900/9750: dis loss = 0.6463, gen loss = 2.3959
Iteration 6000/9750: dis loss = 0.9332, gen loss = 3.0897
Iteration 6100/9750: dis loss = 0.5324, gen loss = 2.0448
Iteration 6200/9750: dis loss = 0.8510, gen loss = 3.2740
Iteration 6300/9750: dis loss = 0.5818, gen loss = 2.8412
Iteration 6400/9750: dis loss = 0.7204, gen loss = 2.3535
Iteration 6500/9750: dis loss = 0.5099, gen loss = 2.3576
Iteration 6600/9750: dis loss = 0.8342, gen loss = 1.3438
```

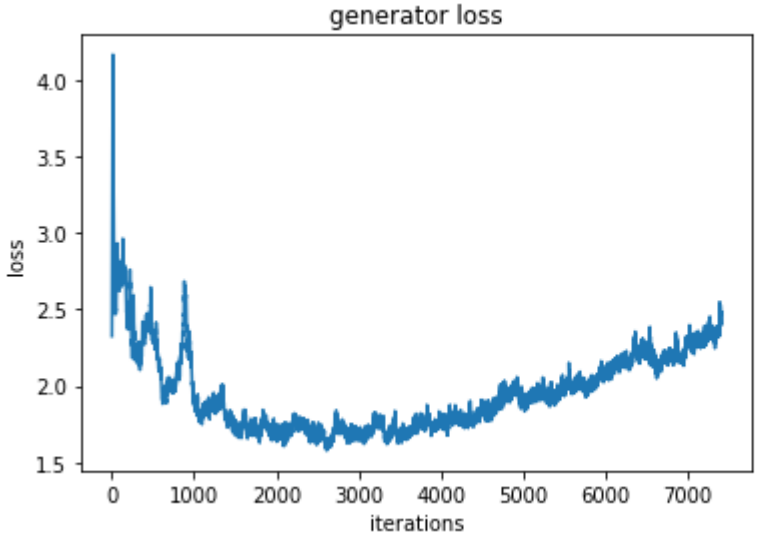

discriminator loss
1.4
1.2
1.0
0.8
0.6
0.4
loss
0
1000
2000
3000
4000
5000
6000
iterations


generator loss
4.0
3.5
3.0
2.5
2.0
1.5
loss
0
1000
2000
3000
4000
5000
6000
iterations

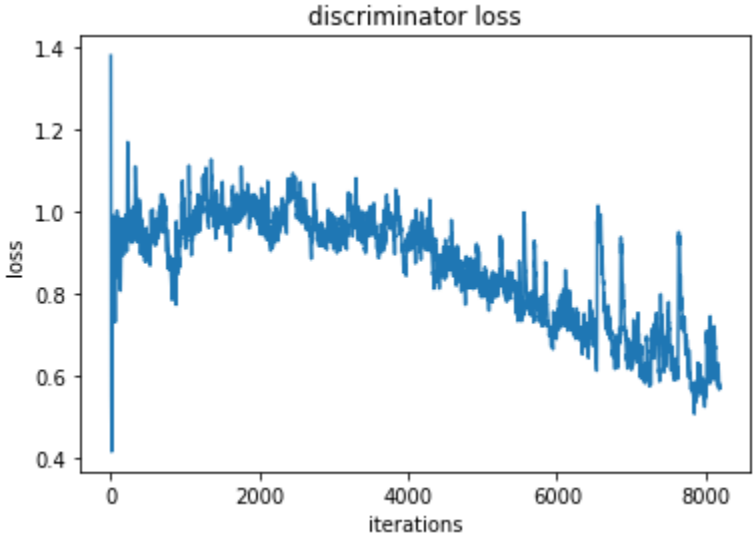
```
Iteration 6700/9750: dis loss = 0.5820, gen loss = 2.1532
Iteration 6800/9750: dis loss = 0.7510, gen loss = 1.2818
Iteration 6900/9750: dis loss = 0.8111, gen loss = 2.7199
Iteration 7000/9750: dis loss = 0.5075, gen loss = 2.4950
Iteration 7100/9750: dis loss = 0.5839, gen loss = 2.4756
Iteration 7200/9750: dis loss = 0.5638, gen loss = 3.2594
Iteration 7300/9750: dis loss = 0.7451, gen loss = 1.8735
Iteration 7400/9750: dis loss = 0.5293, gen loss = 3.1257
```

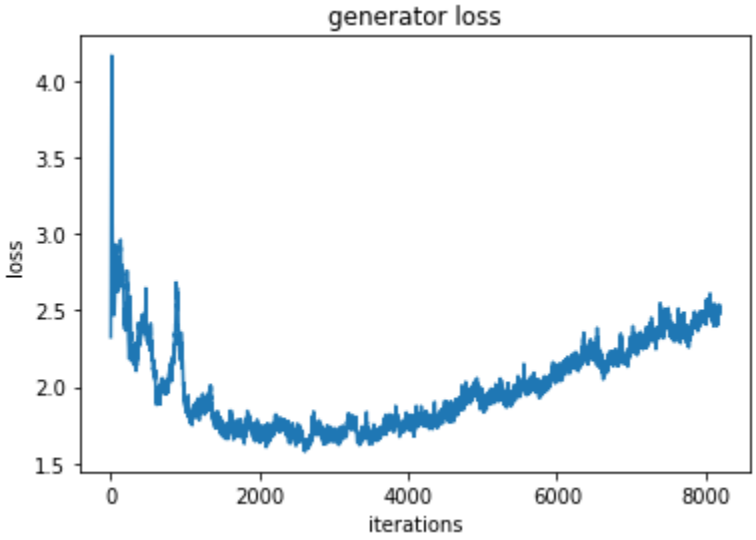



discriminator loss
loss
iterations
1.4
1.2
1.0
0.8
0.6
0.4
0
1000
2000
3000
4000
5000
6000
7000

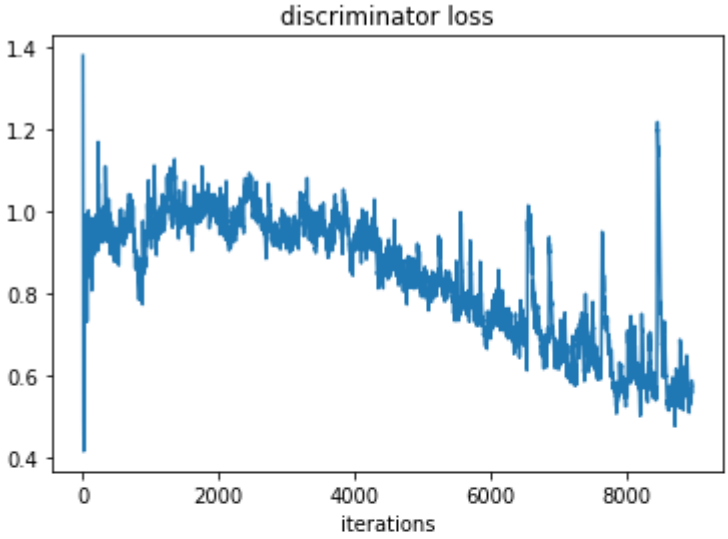

generator loss
loss
iterations
4.0
3.5
3.0
2.5
2.0
1.5
0
1000
2000
3000
4000
5000
6000
7000

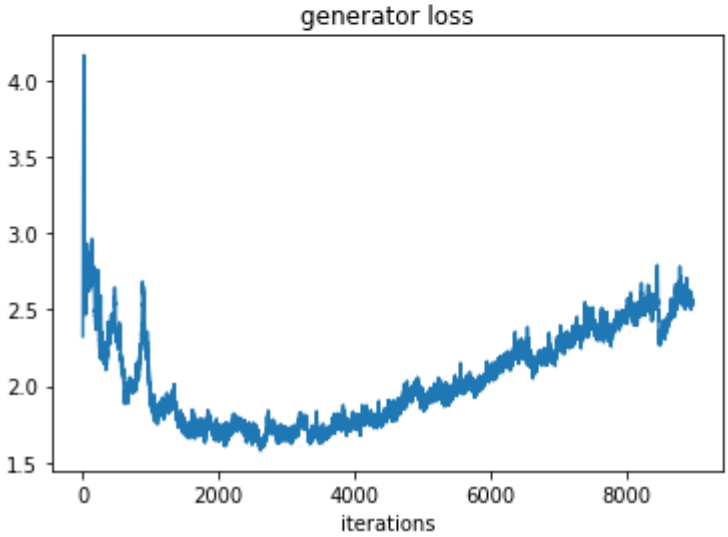
```
Iteration 7500/9750: dis loss = 0.5952, gen loss = 2.0246
Iteration 7600/9750: dis loss = 0.5388, gen loss = 1.8958
Iteration 7700/9750: dis loss = 0.8036, gen loss = 1.7507
Iteration 7800/9750: dis loss = 0.4952, gen loss = 2.5151
Iteration 7900/9750: dis loss = 0.5680, gen loss = 2.3129
Iteration 8000/9750: dis loss = 0.4826, gen loss = 1.8311
Iteration 8100/9750: dis loss = 0.7628, gen loss = 1.0754
```


discriminator loss
loss
iterations
1.4
1.2
1.0
0.8
0.6
0.4
0
2000
4000
6000
8000

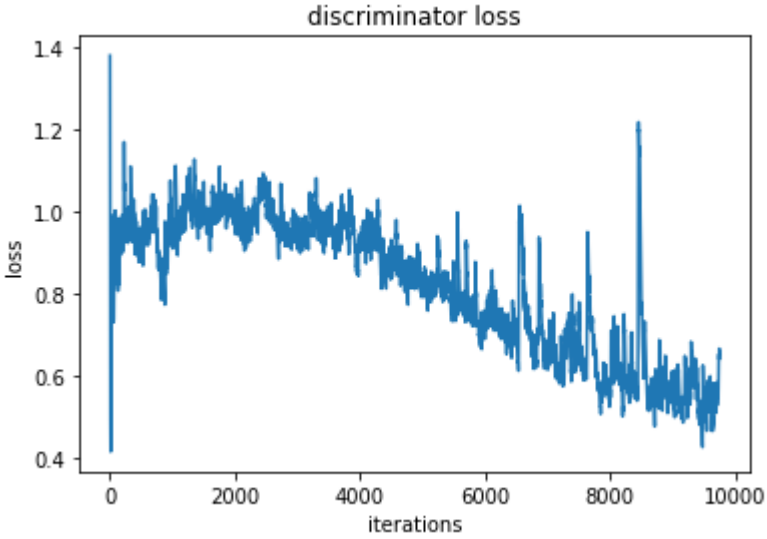

generator loss
loss
iterations
4.0
3.5
3.0
2.5
2.0
1.5
0
2000
4000
6000
8000

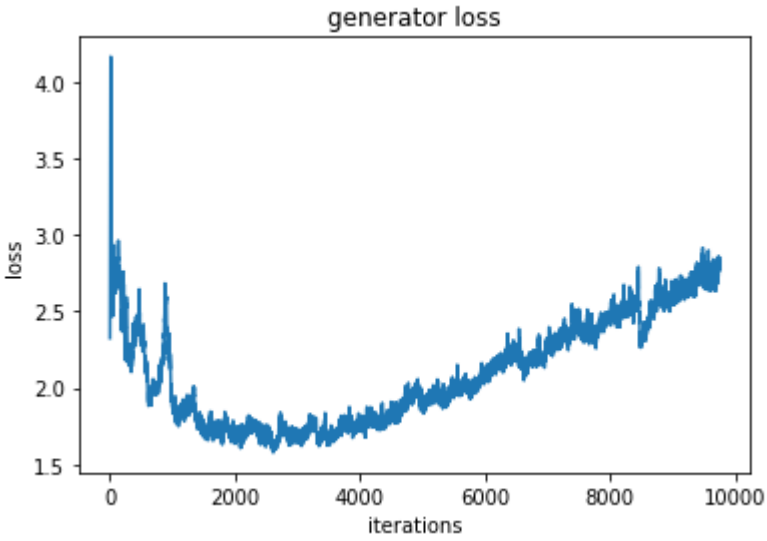
```
Iteration 8200/9750: dis loss = 0.2874, gen loss = 3.1753
Iteration 8300/9750: dis loss = 0.4607, gen loss = 2.6333
Iteration 8400/9750: dis loss = 0.4047, gen loss = 2.6633
Iteration 8500/9750: dis loss = 0.4476, gen loss = 2.4346
Iteration 8600/9750: dis loss = 0.3951, gen loss = 2.9262
Iteration 8700/9750: dis loss = 0.5880, gen loss = 2.7518
Iteration 8800/9750: dis loss = 1.1914, gen loss = 1.0682
Iteration 8900/9750: dis loss = 0.5835, gen loss = 3.0882
```


discriminator loss
1.4
1.2
1.0
0.8
0.6
0.4
0
2000
4000
6000
8000
iterations


generator loss
4.0
3.5
3.0
2.5
2.0
1.5
0
2000
4000
6000
8000
iterations

```
Iteration 9000/9750: dis loss = 0.5811, gen loss = 3.5919
Iteration 9100/9750: dis loss = 0.3838, gen loss = 2.5818
Iteration 9200/9750: dis loss = 0.5255, gen loss = 2.6332
Iteration 9300/9750: dis loss = 0.4628, gen loss = 2.7718
Iteration 9400/9750: dis loss = 0.3673, gen loss = 3.2775
Iteration 9500/9750: dis loss = 0.5223, gen loss = 3.7155
Iteration 9600/9750: dis loss = 0.4908, gen loss = 3.3572
Iteration 9700/9750: dis loss = 1.0866, gen loss = 1.0066
```





Problem 2-2: The forger versus the police, revisited (2 pts)

> **Question**: In the forger versus police story, we made part of it hand-wavy to hide a flaw that makes the story improbable to actually happen and makes it a bad analogy of how the training works in a GAN. Now that you have implemented a GAN, can you spot the flaw?

> Specifically, when we consider one of the two parties, the other is treated as a black box. They know their opponent's result but not how they works. What is wrong here?

> **Your answer: The forger will not know how the police differetiate the bill he created is not real. Thus, he will not be able to modify the bad feature which is picked by the discriminator(police).

> **Question**: By removing the first batch normalization layer, for two different distributions to get confused with each other they must produce two distributions after dis_lrelu1 such that one can be obtained by applying an isotropic scaling and a translation to the other. Such a case is still possible but extremely unlikely to happen.

> Propose a different way of feeding the samples to solve the problem in the second question without omitting any batch normalization layers or changing their mode of operation.

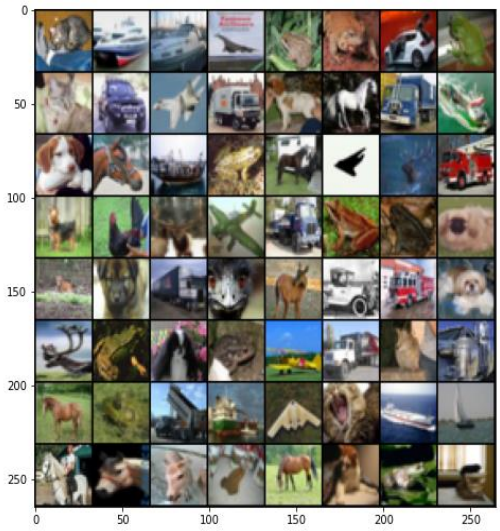
> **Your answer: After reading the paper "On the Effects of Batch and Weight Normalization in Generative Adversarial Networks", I think instead of using batch normalization, we used weight normalization for every layer. we use affine weight-normalized for the last layer while for every other layer we use strict weight-normalized. Strict weight-normalizaed layer is layer without affine transformations.

```
set_seed(241)

dcgan = DCGAN()
dcgan.load_state_dict(torch.load("dcgan.pt", map_location=device))

actmax_results = dcgan.actmax(np.random.normal(size=(64, dcgan.code_size)))
fig = plt.figure(figsize = (8, 8))
ax1 = plt.subplot(111)
ax1.imshow(make_grid(actmax_results, padding=1, normalize=True).numpy().transpose((1, 2, 0)))
plt.show()
```

+ Code + Text

```
dcgan = DCGAN()
dcgan.load_state_dict(torch.load("dcgan.pt", map_location=device))

avg_loss, reconstructions = dcgan.reconstruct(test_samples[0:64])
print('average reconstruction loss = {0:.4f}'.format(avg_loss))
fig = plt.figure(figsize = (8, 8))
ax1 = plt.subplot(111)
ax1.imshow(make_grid(torch.from_numpy(test_samples[0:64]), padding=1).numpy().transpose((1, 2, 0)))
plt.show()
fig = plt.figure(figsize = (8, 8))
ax1 = plt.subplot(111)
ax1.imshow(make_grid(reconstructions, padding=1, normalize=True).numpy().transpose((1, 2, 0)))
plt.show()
```

```
average reconstruction loss = 0.0143
```




0
50
100
150
200
250
0
50
100
150
200
250