

Improved training of Wasserstein Generative Adversarial Networks (GANs): Experimental Results and Insights

Jalaj Bhandari
jb3618@columbia.edu

Aditya Garg
ag3741@columbia.edu

Moorissa Tjokro
mmt2167@columbia.edu

Abstract—Generative Adversarial Networks (GANs) have recently emerged as one of the most popular and widely used generative modeling approach. It leverages recent success of deep learning models and can be trained end to end using back-propagation to generate fake samples from a given data distribution. However, the loss function and the training procedure proposed in the original paper [1] makes the training unstable. Recently, [2] proposed an alternative loss function based on Wasserstein distance between two distributions (instead of the KL distance as proposed originally [1]) and [3] proposed a method to train this loss meaningfully. The idea of this project is to be able to train Wasserstein GANs (hereafter WGANs) based on the proposals made in [2], [3] to different datasets; and understand the pitfalls that still make training GANs hard. GAN output is quite sensitive to the architecture of the discriminator and generator networks, which was the main challenge we faced. By experimenting with different architectures, we were able to get results for three sets of data.

I. INTRODUCTION

GANs are a powerful class of generative models that cast generative modeling as a min-max game between two networks: a generator network, which produces synthetic/fake data and a discriminator network, which evaluates the generators output data and tries to discriminate it from the true data (i.e. the given data). This game is continued until the discriminator can no longer distinguish between the fake data generated by the generator and the true data. A very nice way of understanding this is given in the original paper [1] - “*The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles*”

In GANs, generative model generates samples by passing random noise (gaussian or uniform) through either a deep multilayer perceptron (MLP) or a deep convolution based architecture (DCGAN paper [4]), and the discriminative model is a Convolution neural network (CNN). Both networks can be trained using the back propagation algorithm and only a forward pass is needed to sample from the generative model.

The goal of our project is to be able to apply the approach proposed in [3] and generate fake samples of high quality (for visual inspection). Despite a lot of recent progress, GANs are known to be very hard to train and can be sensitive

to architectures. We see that in our experimental results, GANs are also known to be susceptible to a phenomenon called “mode collapse”, where the generator concentrates on a very small region of the manifold of the true data distribution and is not able to learn at all. In one of our experiments, we see this happening, particularly for deeper architectures.

II. LITERATURE REVIEW AND SUMMARY OF ORIGINAL PAPER

In this section, we will summarize briefly the original GAN method, which forms the basis. We then go on to describe the Wasserstein GAN approach [2], and some of its problems that can be partially alleviated using the approach in [3]. We provide the final algorithm we implement and show some generated samples from the experiment done in [3].

A. Methodology:

1) *GANs*: Let $p(z)$ be some prior distribution over noise variable z , typically gaussian or uniform noise is used. The generator is then a mapping from this random noise to the data space defined as: $g(\theta_g, z)$, where θ_g represent the parameters of the generator network. Let p_r be the true data distribution and p_g be the model distribution implicitly defined by:

$$\tilde{x} = g(z; \theta_g) \text{ where } z \sim p(z) \quad (1)$$

Let $d(x; \theta_d)$ be the discriminator, parameterized by θ_d . The discriminator network receives either a generated sample or a true data sample and must distinguish between the two; while the generator is trained to fool the discriminator. This game is formalized as min-max game between the generator $g(\cdot)$ and the discriminator $d(\cdot)$.

$$\min_{\theta_g} \max_{\theta_d} E_{x \sim p_r} [\log d_{\theta_d}(x)] + E_{z \sim p_z} [\log (1 - d_{\theta_d}(g(\theta_g, z)))] \quad (2)$$

$$\Leftrightarrow \min_{\theta_g} \max_{\theta_d} E_{x \sim p_r} [\log d_{\theta_d}(x)] - E_{z \sim p_z} [\log d_{\theta_d}(g(\theta_g, z))]$$

Note that both the objectives are equivalent. See Figure (1) for an intuitive explanation of the GAN approach.

Problems with GAN: If the discriminator is trained to optimality before each generator parameter update, i.e. completing the inner maximization loop, then minimizing the generator loss (the min part of Equation (2)) amounts to minimizing the Jensen-Shannon divergence between the true data distribution, p_r , and the model distribution learned by the generator, p_g . In practice, optimizing D to completion in the inner loop of

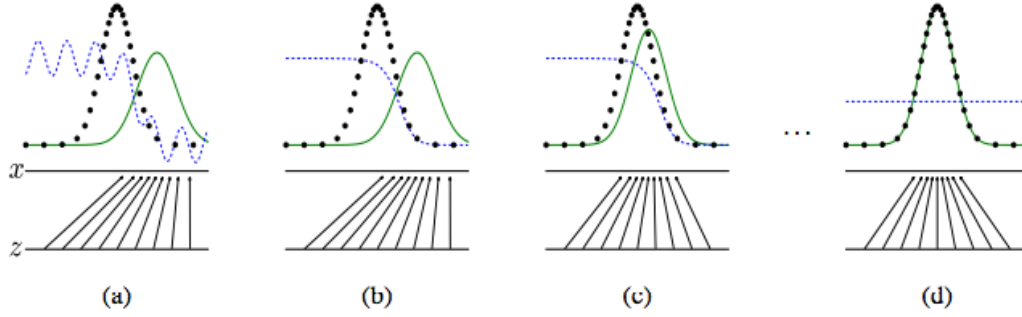


Figure 1: Generative adversarial nets are trained by simultaneously updating the discriminative distribution (D , blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_{data} from those of the generative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x . The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g . (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$. (c) After an update to G , gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$.

Fig. 1: An intuitive understanding of GANs, taken from [1]

training is computationally prohibitive, and on finite datasets results in overfitting. Instead, practitioners alternate between k steps of optimizing d and one step of optimizing g . Tuning k can be really hard, and even then for many examples one gets vanishing gradients initially as the discriminator saturates.

2) *Wasserstein GANs (WGANs)*:: A recent paper, [2], proposes a new loss function to alleviate the problems in the original GAN formulation. They argue that the divergence between the true data distribution and the generated sample distribution, which GANs typically minimize, i.e. $KL(p_r||p_g)$, $JS(p_r||p_g)$ or any other f -divergence, are potentially not continuous with respect to the generators parameters. This leads to vanishing gradients and training difficulty. They propose instead using the Earth-Mover (also called Wasserstein-1) distance $W(p_r, p_g)$, which is informally defined as the minimum cost of transporting mass in order to transform the distribution the p_g into the distribution p_r . Under mild assumptions, $W(p_r, p_g)$ is continuous everywhere and differentiable almost everywhere and this can potentially improve the *learnability* of the GAN approach.

The Wasserstein distance between two distributions, p_r and p_g , can be formally computed as:

$$W(p_r, p_g) = \sup_{\|d\|_L \leq 1} E_{x \sim p_r} [d(x)] - E_{\tilde{x} \sim p_g} [d(\tilde{x})] \quad (3)$$

$$\Leftrightarrow W(p_r, p_g) = \sup_{\|d\|_L \leq 1} E_{x \sim p_r} [d(x)] - E_{z \sim p(z)} [d(g(\theta_g, z))]$$

here $\sup_{\|d\|_L \leq 1}$ is the supremum over the set of all 1-Lipschitz functions. In other words, the Wasserstein distance is parameterized by some function $d(\cdot) \in \mathcal{L}$ where \mathcal{L} is the set of

1-Lipschitz functions. It's important to note that the $d(\cdot)$ does not have to be 1-Lipschitz. It can be k -Lipschitz in which case, the Wasserstein distance is just scaled by k which does not affect training.

Problems with WGANs: The idea of WGANs is to optimize the Wasserstein loss with respect to the generator's parameters. In terms of the min-max game, this results in the following loss function:

$$\min_{\theta_g} W(p_r, p_g) \quad (4)$$

$$\Leftrightarrow \min_{\theta_g} \sup_{\|d\|_L \leq k} E_{x \sim p_r} [d(x)] - E_{z \sim p(z)} [d(g(\theta_g, z))] \quad (5)$$

Thus, when we use the Wasserstein distance metric for GANs, we need to impose the Lipschitz constraint on the function $d(\cdot)$, which in our case is the discriminator. As the discriminator is parameterized by θ_d , the authors in [2] propose to clip the weights of the discriminator to lie within a compact space $[c, c]$ which ensures that the discriminator lies in the set of k -Lipschitz functions; for some k which depends on c and the architecture of the discriminator. But even this approach is problematic in training as illustrated by the authors in [3] who propose an alternative loss function by imposing a penalty on the norm of the gradient of the discriminator's parameters.

3) *Improved training of WGANs:* To mitigate the problems with weight-clipping, authors in [3] prove an important property of the optimal discriminator $d_{\theta_g^*}(\cdot)$: consider a point $x \sim p_g$, then there is a point $y \sim p_r$ such that the gradient of the optimal discriminator $\nabla d(\theta_d^*)$ at all points $x_t = (1-t) \cdot x + t \cdot y$ on a

straight line between x and y has unit norm. In other words,

$$\nabla d_{\theta_d^*}(x_t) = \frac{y - x_t}{\|y - x_t\|} \quad (6)$$

They use this key property and propose a modified loss function with a penalty on the norm of the gradient of the discriminator:

$$L = E_{x \sim p_r} [d(x)] - E_{z \sim p_g(z)} [d(g(\theta_g, z))] + \lambda \cdot E_{\hat{x} \sim p_{\hat{x}}} [(\|\nabla d(\hat{x})\|_2 - 1)^2] \quad (7)$$

here λ is a penalty parameter and $\hat{x} \sim p_{\hat{x}}$ is sampled uniformly along straight lines between pairs of points x, y samples from the data distribution p_r and generated distribution p_g respectively. The algorithm used and implemented in our project is described below in Figure (2).

B. Key results:

Improved training of WGAN paper [3] shows excellent results in terms of the sample quality for generated samples. Figure (3) and (4) shows the results for CIFAR-10 and LSUN bedroom image dataset. [3] also shows the inception score on CIFAR 10 across a wide array of different GAN models.

For our project here; we do not implement the Inception score module. This is computationally exhaustive as we need to pass our samples to a pre-trained publicly available model. We found this model to give an *exhaust-resource error*, and since this is only a heuristic, we ignored this. We only collect samples from the generator, show how it compares to the samples from the original dataset, and plot the generator and discriminator loss.

III. METHODOLOGY:

We start by understanding the basic idea of GANs as proposed by Ian Goodfellow. We explore further by understanding the fundamental concepts behind Wasserstein GANs, which proposes to use the Wasserstein distance as a distance metric between two distributions.

Wasserstein GANs alleviates vanishing gradient problem of the regular GANs, although it still suffers from training issues due to weight clipping. For this reason, understanding the use of gradient penalty in the recently proposed improved training on GANs paper was helpful in helping us yield a reliable set of results on both MNIST digits and fashion dataset, and avoiding problems that GANs implementation alone would encounter.

For this paper, we use two different architectures in our methodology, using DCGAN and MLP generators. We implement our generators to MNIST digits and fashion dataset for 3000 and 5000 iterations respectively, with parameters discussed in the Results section. We also implemented the models to UT Zappos50k shoes dataset with three channels.

We previously tried implementing more architectures on a number of datasets such as CelebA faces and Amazon bags. However, since our implementation works best in MNIST digits, MNIST fashion, we decided to use these for the report with robust hyper parameters.

We also initially planned to compare the generated image quality in both architectures using inception score. However,

as we previously discussed, we will keep this in the future work as we have limited computational resources.

IV. IMPLEMENTATION:

We experiment with two architectures for the generator - 512 4-Layer MLP and DCGAN, while the discriminator in each case is the DCGAN network. As can be seen in Fig. 5a, the MLP generator consists of four feed forward dense layers that take a noise vector as input and generates an image with original image dimensions.

The DCGAN generator Fig. 5b involves 4 deconvolution layers in a pyramidal structure, with filters size of 1024, 512, 256 and 128. Again, it takes the noise vector as input and deconvolves it to create an image with original image dimensions.

The DCGAN discriminator, on the other hand, consists of four convolution layers with 64, 128, 256 and 512 layers respectively that generates a 1 dimension output, given an input image.

V. RESULTS:

We evaluate the results of our improved training function of WGANs by training it on two datasets - MNIST digits and MNIST fashion. For each dataset we experiment with two architectures for the generator: a) 4-layer 512 ReLU MLP and b) DCGAN architecture. For each of these experiments, we keep the discriminator architecture the same.

A. Datasets and Hyper parameters:

Below we describe the datasets we used and the hyper parameters used during training:

- For MNIST digits dataset, we choose the following parameters: a mini-batch size of 64, $z \in \mathbb{R}^{100}$ where z is the input noise fed to the generator, $n_{\text{disc}} = 5$ which is the number of iterations for which the discriminator is trained within each epoch, $\lambda = 10$ where λ is the penalty on the norm of the gradient (see the loss function) and 3000 training epochs. The MNIST images were 28×28 grey-scaled images.
- For MNIST fashion dataset, all inputs are the same except we trained for 5000 training epochs.
- For the UT Zappos50K shoes dataset, we choose the following parameters: a mini-batch size of 32, $z \in \mathbb{R}^{100}$ where z is the input noise fed to the generator, $n_{\text{disc}} = 10$ which is the number of iterations for which the discriminator is trained within each epoch, $\lambda = 10$ where λ is the penalty on the norm of the gradient (see the loss function) and 2000 training epochs. The shoe images were processed to be 32×32 RGB images.

B. Project results:

Instead of implementing Inception score module as mentioned above, we collect samples from the generator, show how it compares to the samples from the original dataset. We also plot the generator and discriminator loss which behaves in a meaningful way, the generator loss decreases over time.

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ , the number of critic iterations per generator iteration n_{critic} , the batch size m , Adam hyperparameters α, β_1, β_2 .

Require: initial critic parameters w_0 , initial generator parameters θ_0 .

```

1: while  $\theta$  has not converged do
2:   for  $t = 1, \dots, n_{\text{critic}}$  do
3:     for  $i = 1, \dots, m$  do
4:       Sample real data  $\mathbf{x} \sim \mathbb{P}_r$ , latent variable  $\mathbf{z} \sim p(\mathbf{z})$ , a random number  $\epsilon \sim U[0, 1]$ .
5:        $\tilde{\mathbf{x}} \leftarrow G_{\theta}(\mathbf{z})$ 
6:        $\hat{\mathbf{x}} \leftarrow \epsilon \mathbf{x} + (1 - \epsilon) \tilde{\mathbf{x}}$ 
7:        $L^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) - D_w(\mathbf{x}) + \lambda(\|\nabla_{\hat{\mathbf{x}}} D_w(\hat{\mathbf{x}})\|_2 - 1)^2$ 
8:     end for
9:      $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$ 
10:   end for
11:   Sample a batch of latent variables  $\{\mathbf{z}^{(i)}\}_{i=1}^m \sim p(\mathbf{z})$ .
12:    $\theta \leftarrow \text{Adam}(\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m -D_w(G_{\theta}(\mathbf{z})), \theta, \alpha, \beta_1, \beta_2)$ 
13: end while

```

Fig. 2: Algorithm for improved training of GANs, taken from [3]

We also plot the Wasserstein distance between the true data distribution and the empirical distribution of the generated samples. This compares well with the training curves as shown in the papers [2] and [3].

C. MNIST Results:

In Figure (6) we display a sample of images from the original MNIST dataset. In Figures (7) and (8), we display generated samples using MLP as the generator architecture and the training loss curves - for both the generator and discriminator loss. Figures (9) and (10) are the corresponding figures using DCGAN as the generator.

D. MNIST Fashion Images:

In Figure (11) we display a sample of images for original MNIST fashion dataset. In Figures (12) and (13), we display generated samples using MLP as the generator architecture and the training loss curves - for both the generator and discriminator loss. Figures (14) and (15) are the corresponding figures using DCGAN as the generator.

E. UT Zappos50K Shoe Images:

In Figure (16) we display a sample of images for original shoe dataset and in Figure (17) we display generated samples using DCGAN as the generator architecture. As can be clearly seen, the generator seems to collapse and we do not get meaningful samples of shoes. We suspect this is a problem with generator architecture; somehow the generator is not able to deal with color images. It could just as well be a failure

mode for the GANs; this is something we want to investigate for future work as this could potentially be a research idea.

F. Comparison of results:

As we are working different datasets, we can't really compare in terms of sample quality. But as a simple sanity check for the results, in Figure (18), we can see the Wasserstein distance estimate (between the true data distribution and the empirical distribution of the generated samples) decreasing with iterations. We see similar behavior for MNIST and MNIST-Fashion dataset.

G. Discussions and insights gained:

We can summarize the results as follows:

- WGANs with gradient penalty for both MLP and DCGAN architectures gave good results which look very much like the true data fed to the model. However, training failed for the shoe dataset. We suspect this happened because the input were RGB images instead of grey-scale images. It seems like there was some issue with the generator architecture. As future work, we would like to experiment with different architectures and be able to reproduce results for color images too.
- For both MNIST and MNIST fashion dataset, the generator loss and Wasserstein distance go down as should be the case.

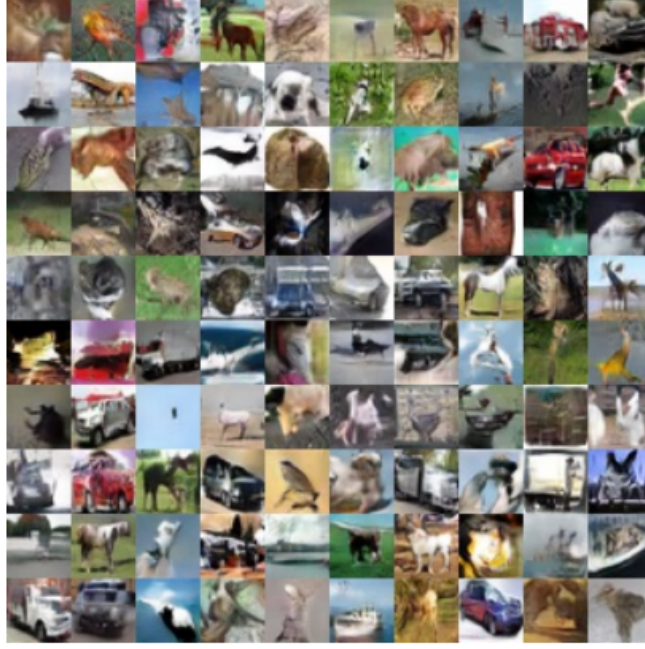


Fig. 3: Generated samples from Improved training of WGANs using a deep convolutional network as generator (DCGAN architecture [4]), taken from [3]



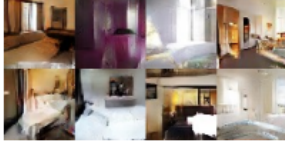





DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G : DCGAN, D : DCGAN)			
			
G : 4-layer 512-dim ReLU MLP, D : DCGAN			
			

Fig. 4: Generated samples from two different generator architectures: using DCGAN and MLP as generators. There were 4 different models trained: DCGAN [4], LSGAN [5], Wasserstein GAN [2] and Wasserstein GAN with gradient penalty (Improved training of WGANs [3]). The image has been taken from [3]

- We experimented with different values of the training hyper parameters like λ and dimension of the input noise but found the results to be robust to these.

VI. CONCLUSIONS:

We conclude with some future work directions. Our project highlights potential research opportunities in GANs: GAN training and outputs seem to be quite sensitive to the architectures of the discriminator and generator. There is also the

problem of mode collapse which has been widely documented in the literature (we did not see it in our experiments); because of which the generator is only able to sample from a part of the true data manifold. We think these are important directions to pursue.

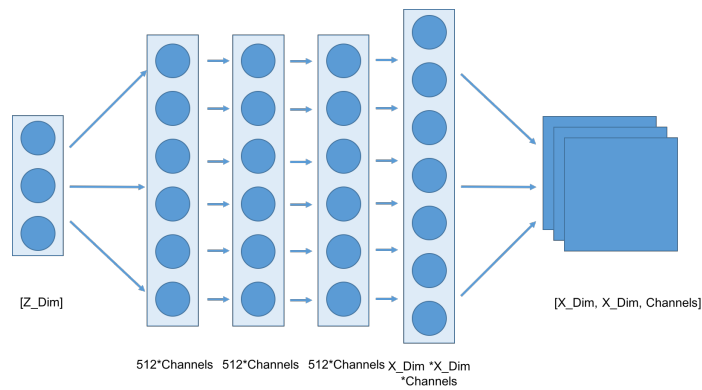
VII. APPENDIX

TABLE I: Individual Contributions

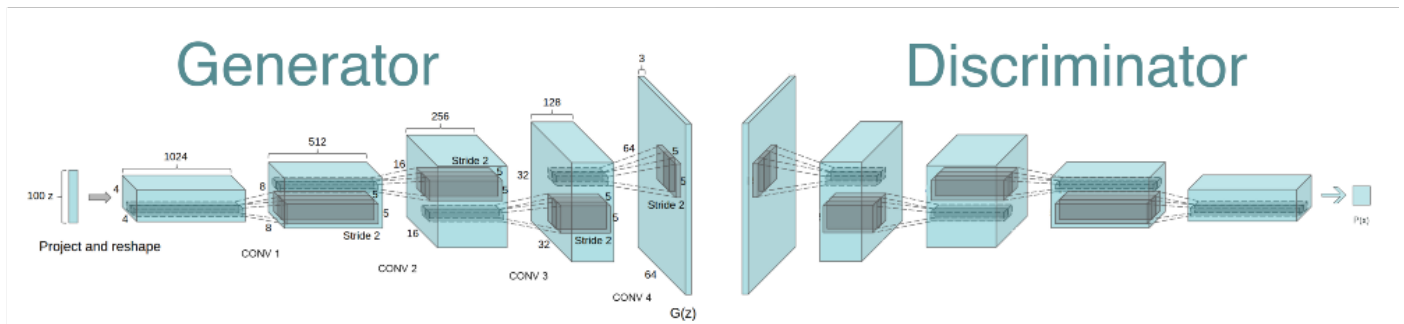
	JB3618	AG3741	MMT2167
Last Name	Bhandari	Garg	Tjokro
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Literature research	DCGAN Implementation	Data Collection and Processing
What I did 2	MLP Implementation	Final module and notebook implementation	Inception Score Implementation
What I did 3	Introduction, Literature, Methodology	Results, Comparison, Conclusion	Results, Comparison, Conclusion

REFERENCES

- [1] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.
- [2] Arjovsky, Martin, Soumith Chintala, and Lon Bottou. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017).
- [3] Gulrajani, Ishaan, et al. "Improved training of wasserstein gans." arXiv preprint arXiv:1704.00028 (2017).
- [4] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).
- [5] Mao, Xudong, et al. "Least squares generative adversarial networks." arXiv preprint ArXiv:1611.04076 (2016).



(a) Architecture for 512 4-Layer MLP Generator



(b) Architecture for DCGAN Generator and Discriminator

Fig. 5: Architectures for MLP And DCGAN (Image Source: <http://gluon.mxnet.io/chapter14/generative-adversarial-networks/dcgan.html>)

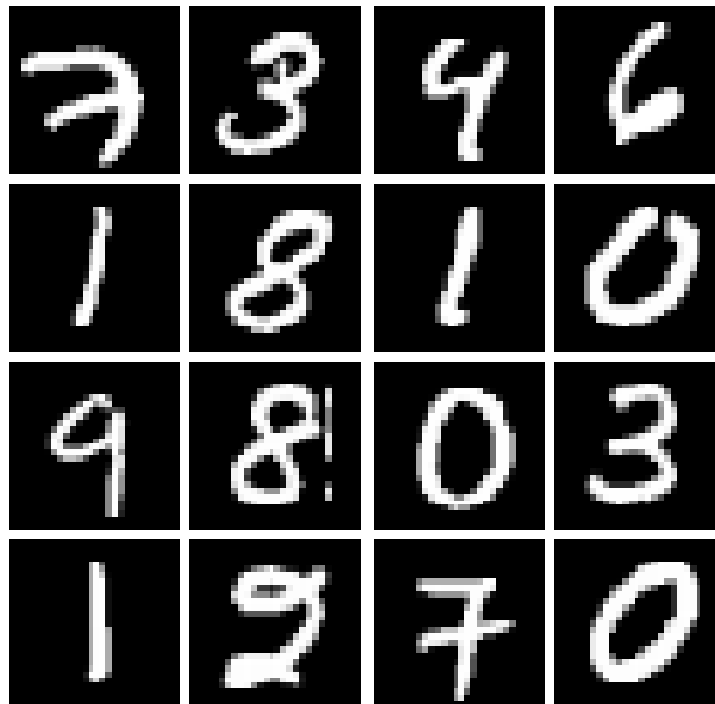
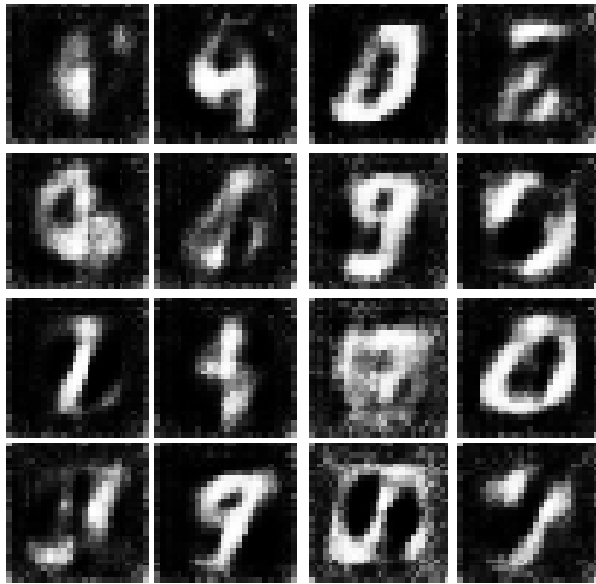
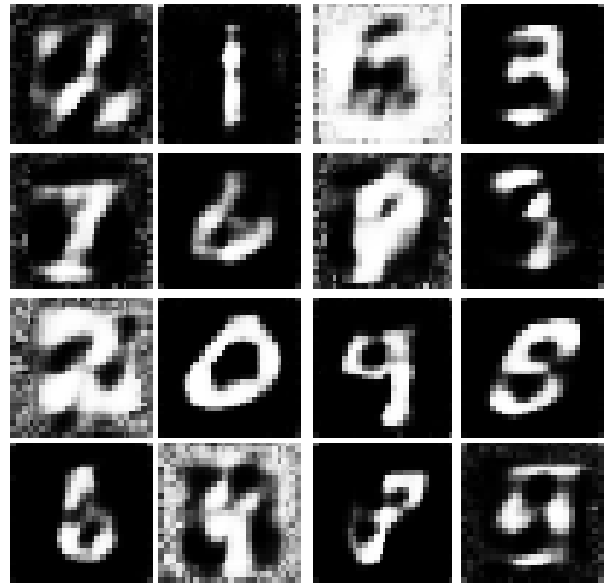


Fig. 6: Actual MNIST digit images

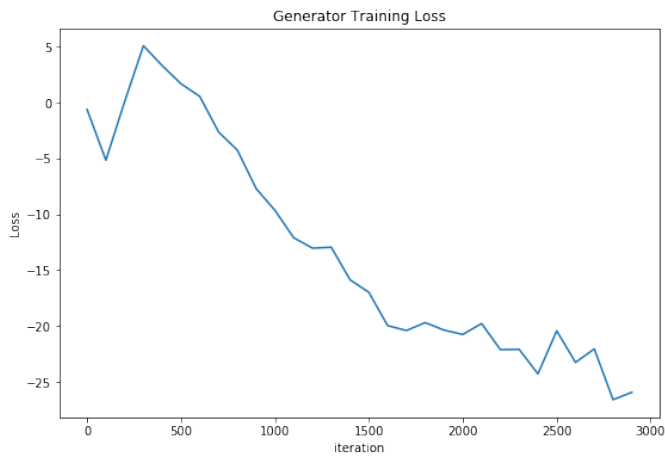


(a) Iteration number = 1500

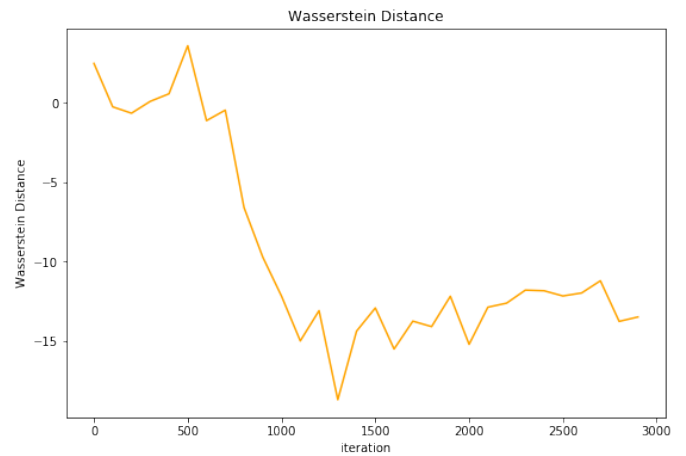


(b) Iteration number = 3000

Fig. 7: MLP generator with MNIST digits dataset: generated samples

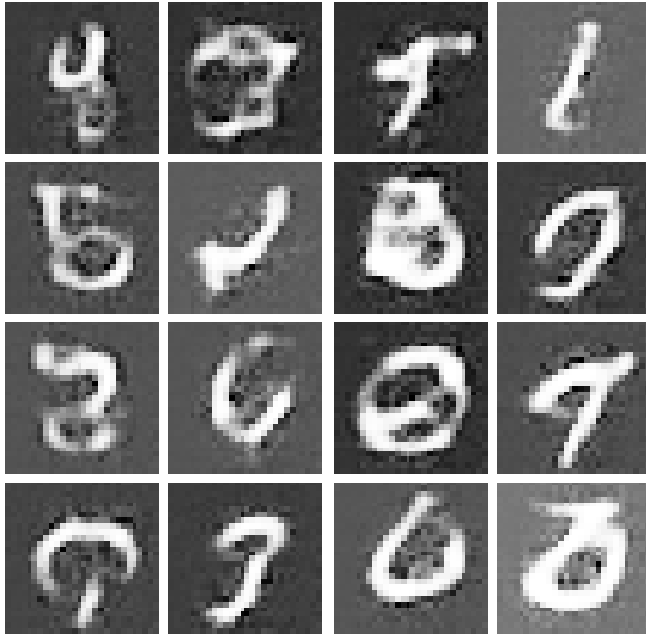


(a) Generator training loss for 3000 iterations

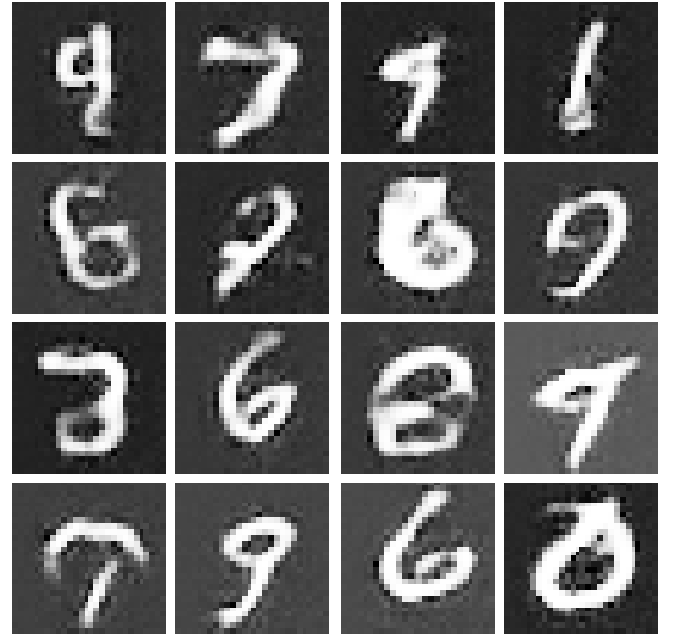


(b) Wasserstein distance for 3000 iterations

Fig. 8: MLP generator with MNIST digits dataset: both generator loss and Wasserstein distance go down as should be the case

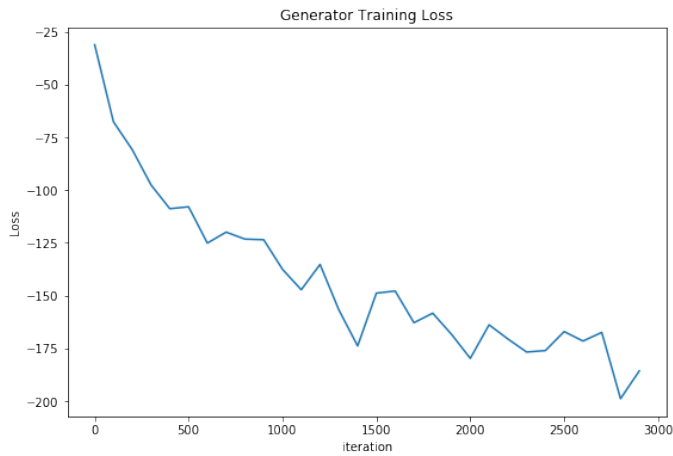


(a) Iteration number = 1500

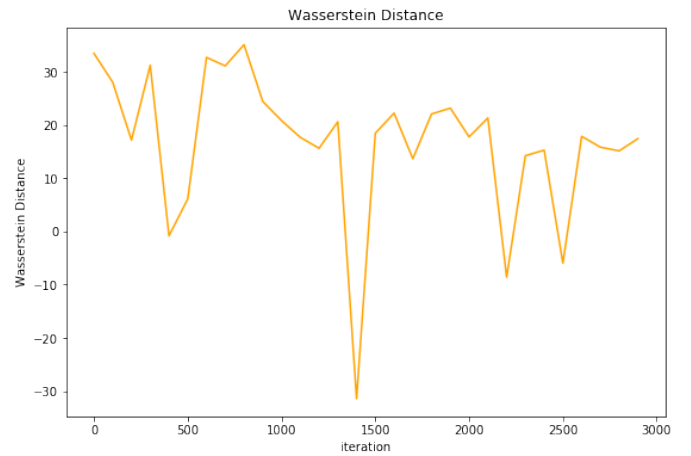


(b) Iteration number = 3000

Fig. 9: DCGAN generator with MNIST digits dataset: generated samples



(a) Generator training loss for 3000 iterations

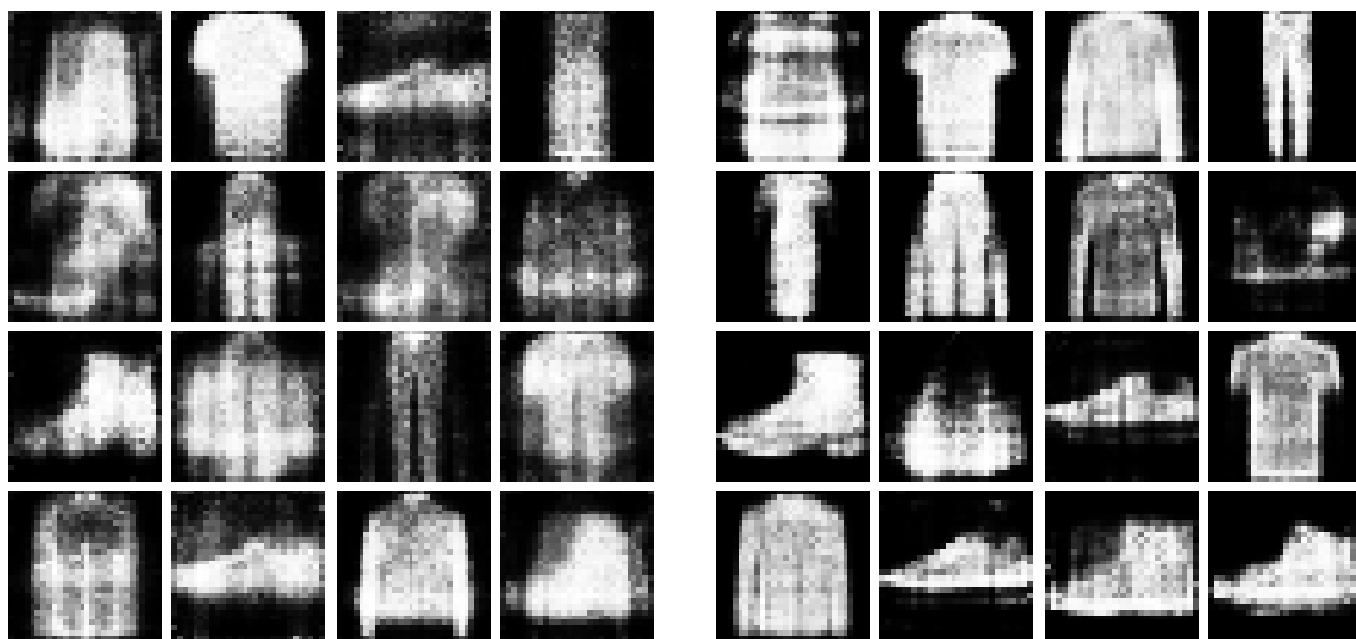


(b) Wasserstein distance for 3000 iterations

Fig. 10: DCGAN generator with MNIST digits dataset: both generator loss and Wasserstein distance go down as should be the case



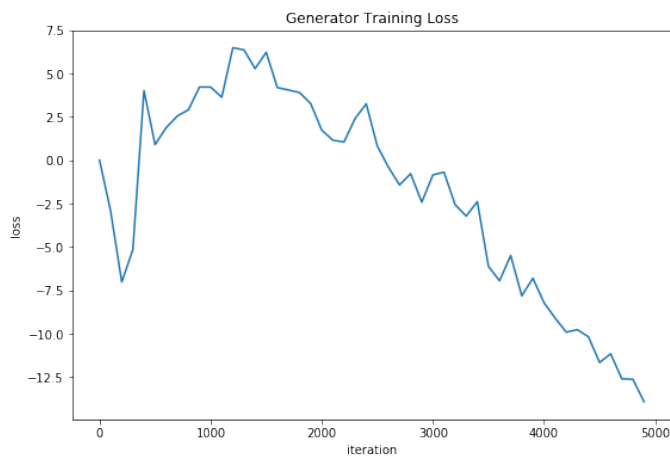
Fig. 11: Actual MNIST fashion images



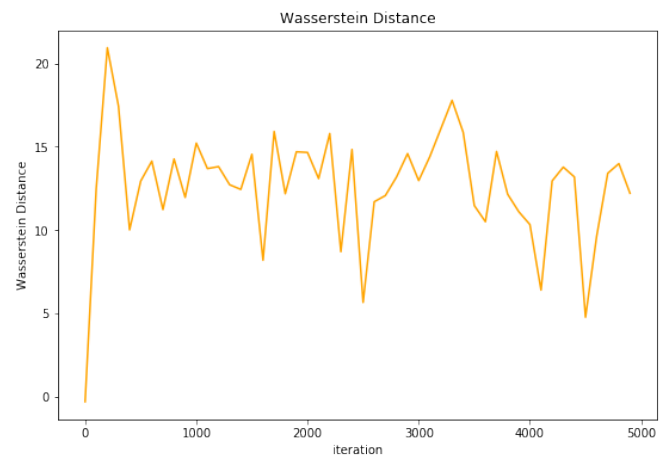
(a) Iteration number = 10

(b) Iteration number = 45

Fig. 12: MLP generator with MNIST fashion dataset: generated samples

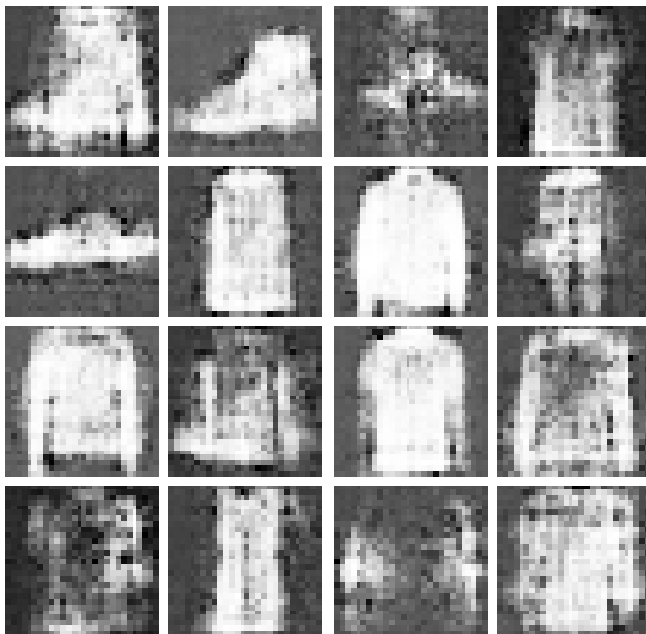


(a) Generator training loss for 5000 iterations

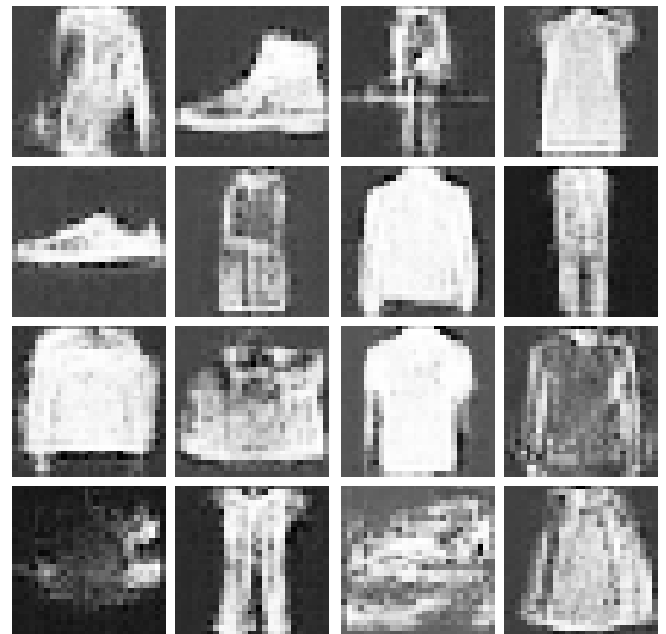


(b) Wasserstein distance for 5000 iterations

Fig. 13: MLP generator with MNIST fashion dataset: both generator loss and Wasserstein distance go down as should be the case

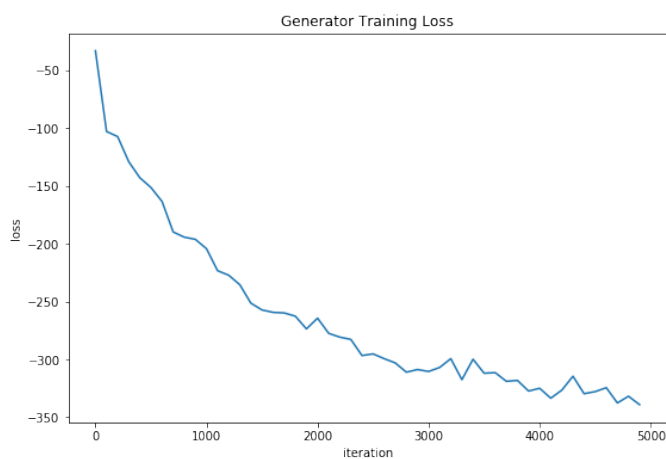


(a) Iteration number = 10

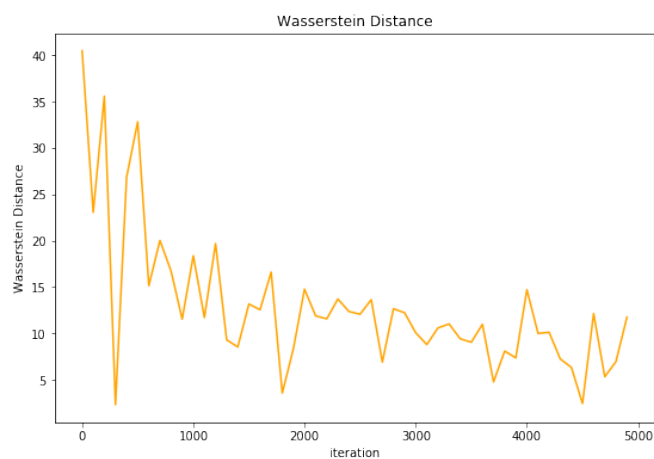


(b) Iteration number = 40

Fig. 14: DCGAN generator with MNIST fashion dataset: generated samples



(a) Generator training loss for 5000 iterations



(b) Wasserstein distance for 5000 iterations

Fig. 15: DCGAN generator with MNIST fashion dataset: both generator loss and Wasserstein distance go down as should be the case



Fig. 16: Actual High-Knee shoes images

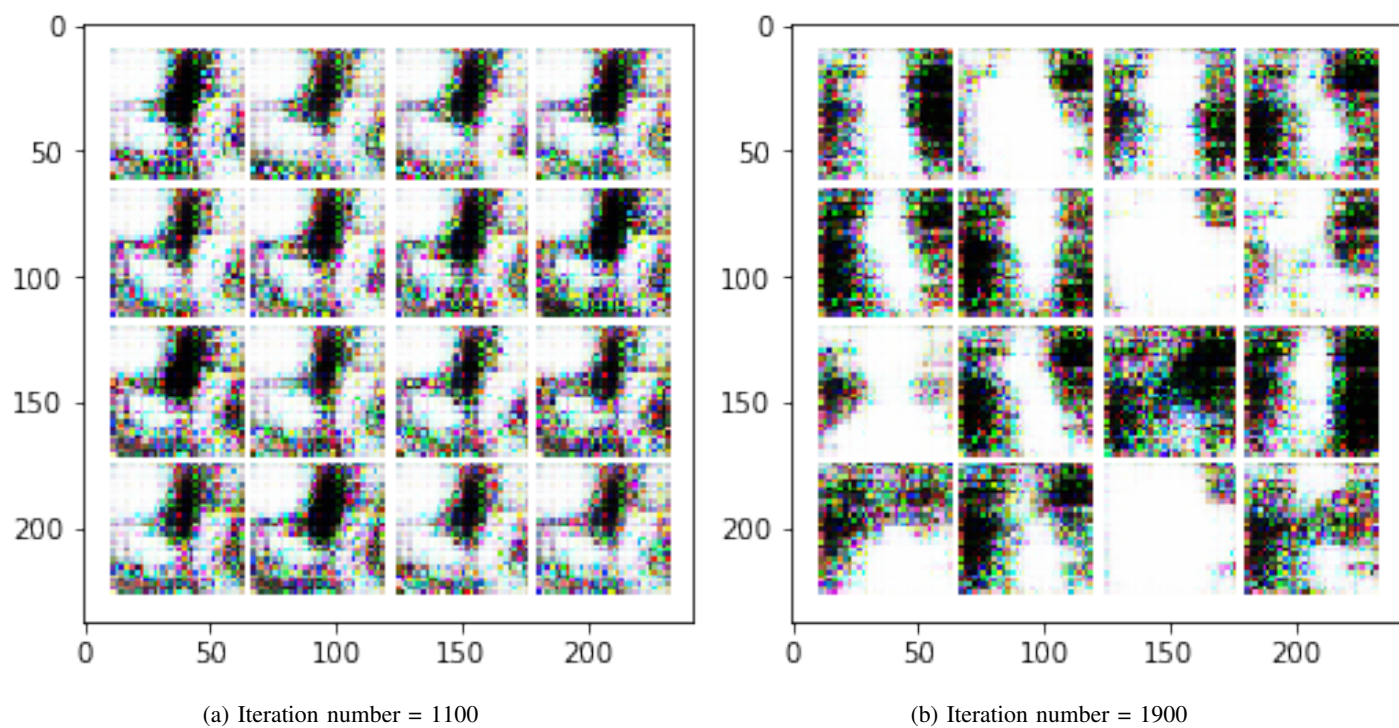


Fig. 17: DCGAN with Zappos High-Knee shoes dataset: generated samples

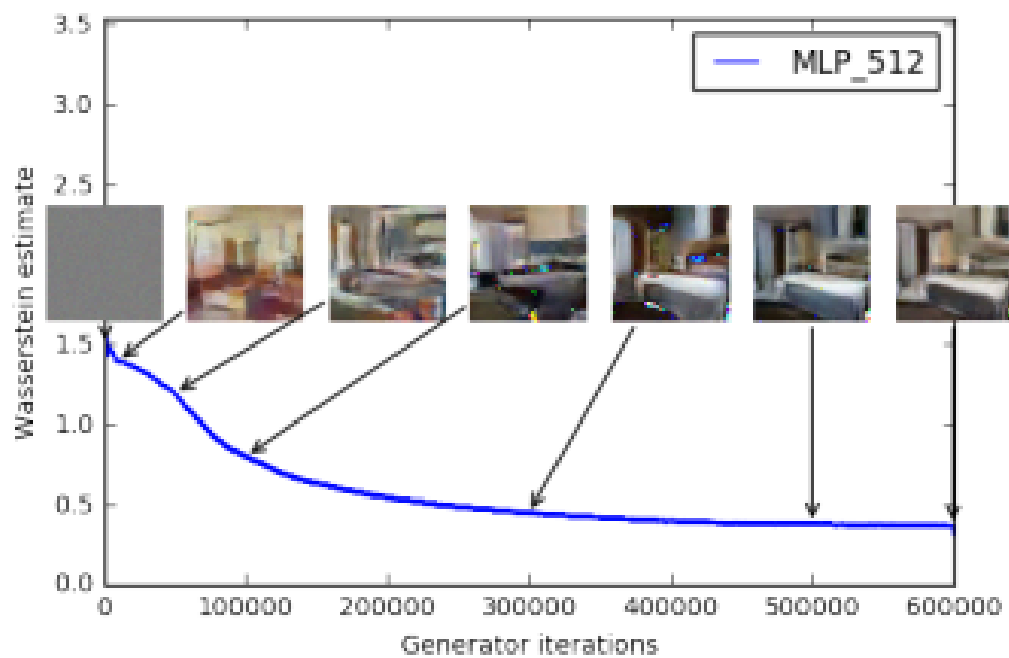


Fig. 18: Wasserstein distance estimate (between the true data distribution and the empirical distribution of the generated samples) decreasing with iterations. Taken from WGAN paper [2]