EE5239 : NONLINEAR OPTIMIZATION

PROJECT REPORT

Generative Adversarial Nets (GANs): Training Algorithms

Akshit Goyal

A report submitted in partial fulfilment of the requirements
for the academic course EE5239

in the

Department of Electrical & Computer Engineering

December 18, 2019

Contents

1 Introduction

Generative Adversarial Nets (GANs) are a class of generative models which aim to learn the probability density implicitly from the data itself. This means that this approach towards learning probability distributions does not require a prior density function to be defined on data in contrast to approaches like Maximum Likelihood (ML) estimation, Maximum a Posteriori (MAP) estimation, etc.

Furthermore, GANs have become recently popular and have found applications in tasks such as image generation of human faces (Deepfakes media), 3D object generation, photograph editing applications (e.g. FaceApp, Figure 1), image to image translation (Figure 2), text to image translation, etc. It is also believed that GANs can be exploited to generate huge training data for reinforcement learning (RL) models i.e. simulate a large number of real-world situations for a RL model to learn better which otherwise is quite difficult.

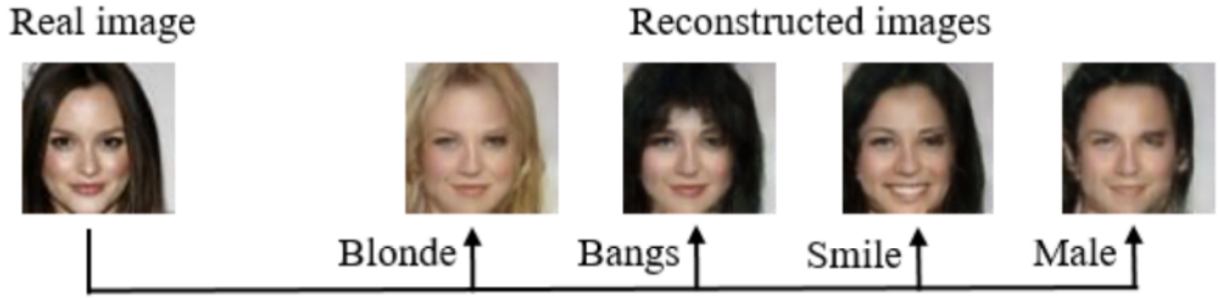


Figure 1: Example of Face Photo Editing with IcGAN. Taken from Invertible Conditional GANs For Image Editing, 2016.

Figure 2: Example of Sketches to Color Photographs With pix2pix. Taken from Image-to-Image Translation with Conditional Adversarial Networks, 2016.

2 Background

Given this huge success of GANs, it becomes very important to understand the underlying background of these models. The first success in formulating GANs was achieved by [1] on image generation which considers two networks, a generator and a discriminator. The goal of the generator is to generate fake images from random noise data and the goal of the discriminator is to distinguish between a real & a fake image. This can be seen as some sort of a police and counterfeit game where generator (G) is a counterfeiter and discriminator (D) is a police. In mathematical terms, it is interpreted as a minimax game where either one of G or D maximizes some value function and other minimizes it. For example,

$$\min_G \max_D \; V(G, D)$$

where $V(G, D)$ represents some value function. For practical implementation, [1] proposes to train a deep neural network for both G and D, and therefore gave an algorithm to solve the above optimization problem for some value function (will be discussed later). With further work on empirically trying different deep network architectures, deep convolution GANs (DCGANs) [2] became quite popular.

Since its inception, the interest surged towards understanding the theoretical properties of original mathematical formulation given by [1]. From the point of view of distance between two probability distributions, the authors in [3] made an attempt to highlight issues pertaining to original GANs arising from the value (or loss) function used for training. The loss function suffered from the problems of discontinuity and vanishing gradients. Further, the authors in [3] proposed an alternative distance measure called Earth-Mover or Wasserstein-1 distance and thus formulated a new class of GANs called Wasserstein GANs (WGANs) [4]. This formulation was posed with a requirement of a Lipschitz continuity on discriminator function (called critic function in the paper), which they enforced using a clipping on discriminator network weights.

This weight clipping on D was further investigated by [5]. After running experiments on toy datasets for various clipping values, [5] showed that for WGANs with weight clipping, the gradient norm of D across its layers either vanishes or explodes (see Figure 3). Thus, [5] extended the WGAN formulation of [4] by adding a gradient penalty term to it (called as WGAN-GP). The penalty term directly constrains the gradient norm of discriminator to be strictly 1, but for some distribution lying between the original and the generator distribution (will be discussed further).

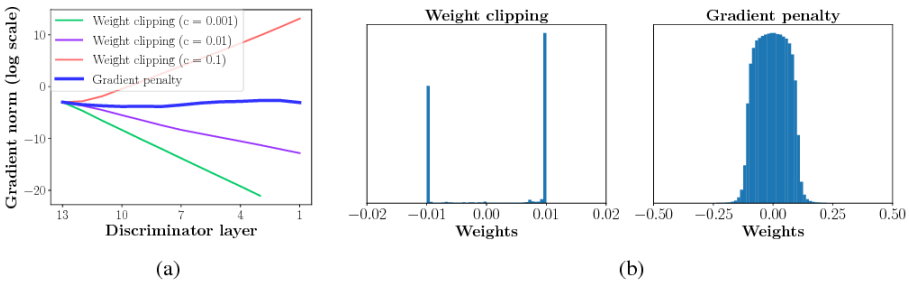


Figure 3: Experiment results from [5]. (a) Gradient norms of deep WGAN critics during training on Swiss Roll dataset. (b) (left) Weight clipping pushes weights towards extremes of clipping range unlike gradient penalty (right).

3 Main Contributions

The main idea behind this project is to test different GAN formulations such as DCGANs, WGANs, WGAN-GP, and reproduce the results as given in their seminal work. This exercise enables to think about GANs from the first principles and at the same time, it also helps to gain insights about their practical performance. It also helps to draw comparison between different formulations on given datasets (which are MNIST Handwriting dataset and MNIST Fashion dataset, in our case).

Moreover, the knowledge gained from this exercise can be useful to exploit existing formulations for different research problems such as protein folding problem. Recently, Google's Deepmind used GANs for predicting protein structure from vast genomic data, a step towards understanding how DNA sequence chains folds into complex 3-D protein structure (AlphaFold). This is a clickable link. You can click if you wish to know more about the topic.
Click here to see the animation of protein structure predicted through a generative network.

4 GANs Training Algorithm Formulations

4.1 Original GANs (or DCGAN)

The value function for original GANs given by [1] is,

$$V(G, D) = \mathbb{E}_{x\sim\mathbb{P}_r}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

where $\mathbb{P}_r$ is the real data distribution, z is a noise variable and p_z is the noise distribution. Here $D(\cdot)$ gives the probability of the input to discriminator being real. Therefore for the optimal G, D we would like to solve the following problem,

$$\min_G \max_D \; V(G, D) \tag{1}$$

To learn functions D and G, we parameterize them with a deep neural network (θ_d and θ_g respectively) and update the weights of each network one by one. The optimization problem with respect to D keeping G fixed is,

$$\max_D \; \mathbb{E}_{x\sim\mathbb{P}_r}[\log D_{\theta_d}(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D_{\theta_d}(G(z)))]$$

$$\Updownarrow$$

$$\min_D \; -\mathbb{E}_{x\sim\mathbb{P}_r}[\log D_{\theta_d}(x)] - \mathbb{E}_{z\sim p_z(z)}[\log(1 - D_{\theta_d}(G(z)))] \tag{2}$$

The optimization problem with respect to G keeping D fixed is,

$$\min_G \; \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G_{\theta_g}(z)))]$$

$$\Updownarrow$$

$$\max_G \; \mathbb{E}_{z\sim p_z(z)}[\log D(G_{\theta_g}(z))]$$

$$\Updownarrow$$

$$\min_G \; -\mathbb{E}_{z\sim p_z(z)}[\log D(G_{\theta_g}(z))] \tag{3}$$

The equations (1) and (2) can be implemented using binary cross entropy loss in tensorflow. Moreover, if $V(G, D)$ is solved to optimality for D given any G i.e. $D^*_G = \frac{\mathbb{P}_r}{\mathbb{P}_r + \mathbb{P}_g}$ where $\mathbb{P}_g$ is the generator's distribution, then equation (1) can be re-written as

$$\min_G \; C(G) = V(G, D^*_G) = \mathbb{E}_{x\sim\mathbb{P}_r}\Big[\log \frac{\mathbb{P}_r}{\mathbb{P}_r + \mathbb{P}_g}\Big] + \mathbb{E}_{x\sim\mathbb{P}_g}\Big[\log \frac{\mathbb{P}_g}{\mathbb{P}_r + \mathbb{P}_g}\Big]$$

. We can rewrite $C(G)$ as,

$$C(G) = -\log(4) + KL\Big(\mathbb{P}_r||\frac{\mathbb{P}_r + \mathbb{P}_g}{2}\Big) + KL\Big(\mathbb{P}_g||\frac{\mathbb{P}_r + \mathbb{P}_g}{2}\Big) = -\log(4) + 2 \cdot JSD(\mathbb{P}_r||\mathbb{P}_g)$$

For optimal D, solving equation (1) corresponds to minimizing Jensen-Shannon divergence (JSD) between $\mathbb{P}_r$ and $\mathbb{P}_g$. Theoretically, the optimal solution is $\mathbb{P}^*_r = \mathbb{P}^*_g$.

Note: Although DCGAN [2] is just a neural network architecture for generator and discriminator networks of GANs but often used to refer original GAN formulation given by [1]. Note that the DCGAN architecture can be used with other formulations as well which are described below.

4.2 Wasserstein GANs (WGAN)

Previously, we saw for optimal D, DCGANs minimizes JSD between $\mathbb{P}_r$ and $\mathbb{P}_g$. The authors in [4] studied the theoretical properties of various distance measures between distributions including KL divergence, JSD and Wasserstein distance. [4] concluded that Wasserstein distance induces weaker topology than JSD, and therefore makes easier for the distributions to converge. The Earth-Mover (EM) or Wasserstein-1 distance is given by,

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\; \|x - y\| \;] \tag{4}$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively, $\mathbb{P}_r$ and $\mathbb{P}_g$ i.e. $\int \gamma \; dy = \mathbb{P}_r$ and $\int \gamma \; dx = \mathbb{P}_g$. Intuitively, $\gamma(x, y)$ denotes the amount of mass to be transported from x to y in order to convert distribution $\mathbb{P}_r$ into $\mathbb{P}_g$. So, calculating W corresponds to finding the optimal cost of transport.
Further, [4] argues that infimum in equation (4) is intractable and the dual problem is given by,

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|D\|_L\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[D(x)]$$

where the supremum is over all the 1-Lipschitz functions D. If we parameterize D with a neural network having parameters θ_d and introduce G which generates data according to $\mathbb{P}_g$ through a neural network (parameters θ_g), then for finding optimal D we consider solving,

$$\max_{\theta_d\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[D_{\theta_d}(x)] - \mathbb{E}_{z\sim p_z(z)}[D_{\theta_d}(G_{\theta_g}(z))]$$

where weights θ_d lie in a compact space $\mathcal{W}$. For finding optimal G that minimizes W, we consider solving,

$$\min_{\theta_g} \left\{ \max_{\theta_d\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[D_{\theta_d}(x)] - \mathbb{E}_{z\sim p_z(z)}[D_{\theta_d}(G_{\theta_g}(z))] \right\} \tag{5}$$

The fact that $\mathcal{W}$ is compact implies D_{θ_d} will be Lipschitz. To enforce that θ_d lie in a compact space, [4] introduces clipping them to a box after each update. For example, $\mathcal{W} \in [-0.01, 0.01]^l$.

4.3 WGAN with Gradient Penalty (WGAN-GP)

Although, according to [4], WGANs produced samples of quality which correlated better with generator training loss compared to original GANs, but the idea of weight clipping was a terrible way of enforcing a Lipschitz constraint. The authors in [5] showed experimentally that weight clipping actually fails to maintain the gradient norm of D across layers[1] (see figure 3). Therefore, [5] relaxes constraining the weights of the network by penalizing the gradient norm of D to be strictly 1. The new objective/loss function is given as,

$$L = \mathbb{E}_{x\sim\mathbb{P}_g}[D(x)] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \lambda \underbrace{\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Gradient Penalty (GP) term}} \tag{6}$$

where $\hat{x}$ is some random distribution, which means enforcing a softer version of the penalty. As usual, after parameterizing with θ_d and θ_g, the optimization problem is formulated by,

$$\max_{\theta_g} \min_{\theta_d} \mathbb{E}_{z\sim p_z(z)}[D_{\theta_d}(G_{\theta_g}(z)] - \mathbb{E}_{x\sim\mathbb{P}_r}[D_{\theta_d}(x)] + \lambda\, \mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}} D_{\theta_d}(\hat{x})\|_2 - 1)^2]$$

For $\hat{x}$, [5] defines $\mathbb{P}_{\hat{x}}$ sampling uniformly along straight lines between pair of points sampled from $\mathbb{P}_r$ and $\mathbb{P}_g$. In implementation, for $\epsilon \sim U[0, 1]$,

$$\hat{x} \leftarrow \epsilon \cdot x + (1 - \epsilon) \cdot G_{\theta_g}(z)$$

for some $x \sim \mathbb{P}_r$ and $z \sim p_z(z)$.

Notice the reversal in order of max and min since penalty term is added to the loss function with respect to D which is negative of the function being maximized in equation (5).

4.4 Algorithm Template for GANs

Algorithm 1 Stochastic Gradient Descent Training of GANs– DCGAN/ WGAN/ WGAN-GP. For our experiments, batch size $m = 50$.

1: **for** number of generator training iterations **do**
2: **for** k steps **do**
3: Sample from real data i.e. $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$.
4: Sample a batch of noise samples i.e. $\{z^{(i)}\}_{i=1}^m \sim p_z(z)$.
5: Update discriminator weights by descending its loss function:

$$\theta_d \leftarrow \textbf{optimizer}\big[\nabla_{\theta_d}(\text{Loss function w.r.t. } D),\ \theta_d,\ \text{optimizer parameters}\big]$$

6: **end for**
7: Sample a batch of noise samples i.e. $\{z^{(i)}\}_{i=1}^m \sim p_z(z)$.
8: Update generator weights by descending its loss function:

$$\theta_g \leftarrow \textbf{optimizer}\big[\nabla_{\theta_g}(\text{Loss function w.r.t. } G),\ \theta_g,\ \text{optimizer parameters}\big]$$

9: **end for**

[1]Note: A differentiable function $g : \mathbb{R} \to \mathbb{R}$ is K-Lipschitz continuous (with $K = \sup |g'(x)|$) if and only if it has bounded first derivative.

	DCGAN	WGAN	WGAN-GP
No. of critic iterations, k	1	5	5
Loss function[2] w.r.t D	$\frac{1}{m}\sum_{i=1}^{m}\Big[-\log D_{\theta_d}\Big(x^{(i)}\Big) - \log\Big(1-D_{\theta_d}\Big(G\Big(z^{(i)}\Big)\Big)\Big)\Big]$	$\frac{1}{m}\sum_{i=1}^{m}\Big[D_{\theta_d}\Big(G\Big(z^{(i)}\Big)\Big) - D_{\theta_d}\Big(x^{(i)}\Big)\Big]$	$\frac{1}{m}\sum_{i=1}^{m}\Big[D_{\theta_d}\Big(G\Big(z^{(i)}\Big)\Big) - D_{\theta_d}\Big(x^{(i)}\Big) + \lambda\Big(\|\nabla_{\hat{x}} D_{\theta_d}(\hat{x}^{(i)})\|_2 - 1\Big)^2\Big]$
Loss function w.r.t G	$\frac{1}{m}\sum_{i=1}^{m} -\log D\Big(G_{\theta_g}\Big(z^{(i)}\Big)\Big)$	$\frac{1}{m}\sum_{i=1}^{m} -D\Big(G_{\theta_g}\Big(z^{(i)}\Big)\Big)$	$\frac{1}{m}\sum_{i=1}^{m} -D\Big(G_{\theta_g}\Big(z^{(i)}\Big)\Big)$
Optimizer	Adam	RMSprop	Adam
Optimizer parameters	$\alpha = 2\times10^{-4},\ \beta_1 = 0.5$	$\alpha = 5\times10^{-5}$	$\alpha = 1\times10^{-4},\ \beta_1 = 0.5,\ \beta_2 = 0.9$
Additional step	–	After discriminator update $\theta_d \leftarrow \mathbf{clip}(\theta_d,\ -c,\ c)$	Before discriminator update For $\epsilon \in U[0,1],\ i = 1,\ldots,m$ $\hat{x}^{(i)} \leftarrow \epsilon\cdot x^{(i)} + (1-\epsilon)\cdot G_{\theta_g}\big(z^{(i)}\big)$

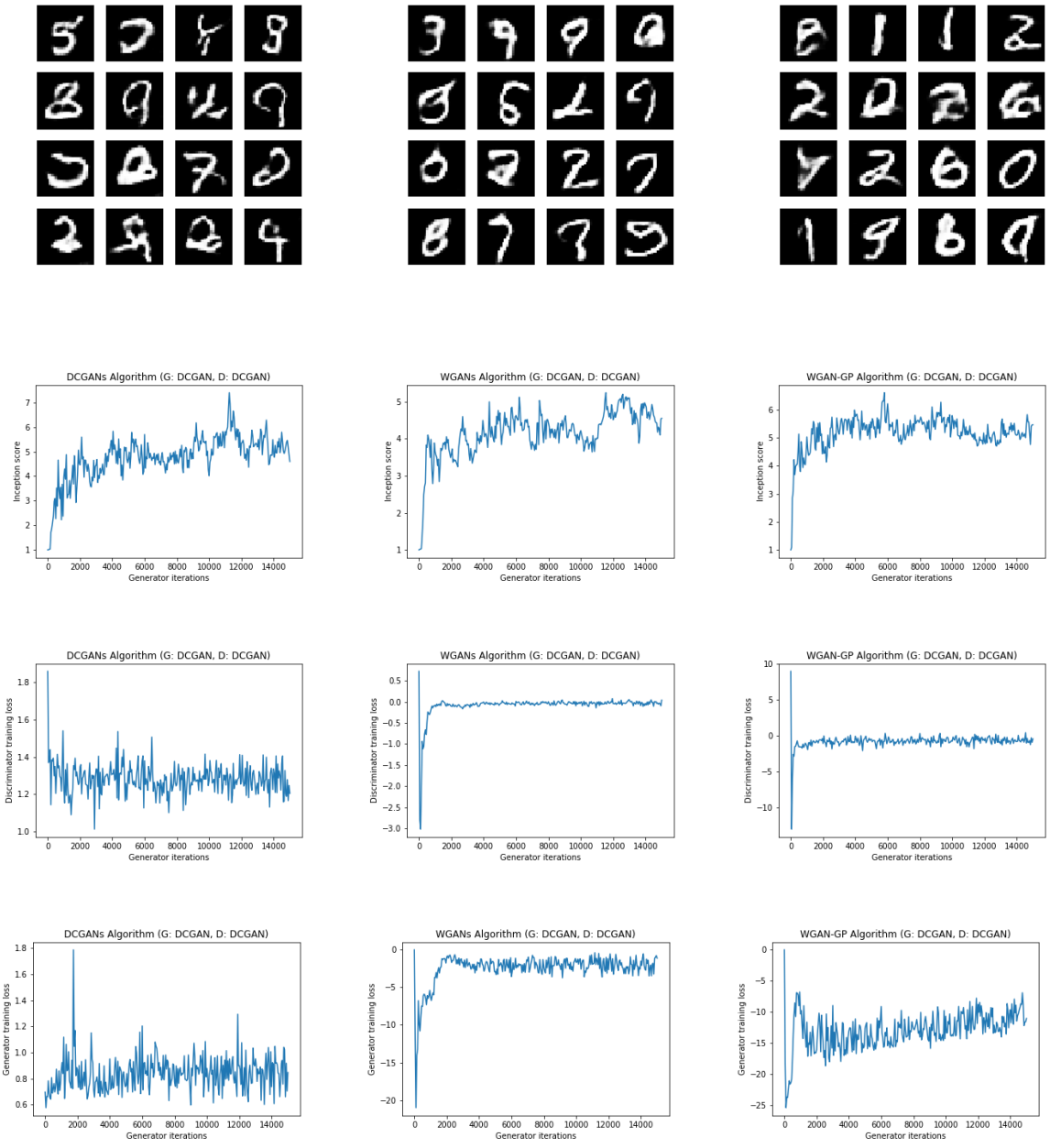
Table 1: Table for different parameters of Algorithm 1 for the three GAN formulations.

5 Results

In our experiments, for all the formulations mentioned above, the generator (G) and discriminator (D) network architectures[3] are DCGANs. One of the dataset used is MNIST Handwriting dataset (also called just 'MNIST dataset') which consists of $60,000$ training images of handwritten digits from 0 to 9. Another dataset used is 'MNIST Fashion dataset' consisting of a training set of 60,000 examples. For both the datasets, each example is a 28×28 grayscale image, associated with a label from 10 classes.

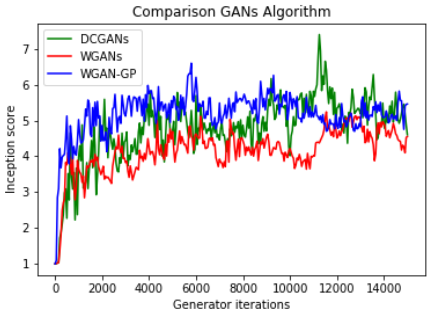
For comparing the performance of the three formulations/algorithms on MNIST dataset, we calculated the inception score given by [6].

[2]Note: Here, for some loss function, $f(\mathbf{x})$ it implies that the corresponding optimization problem is $\min_{\mathbf{x}} f(\mathbf{x})$.
[3]Refer Appendix I for the details.

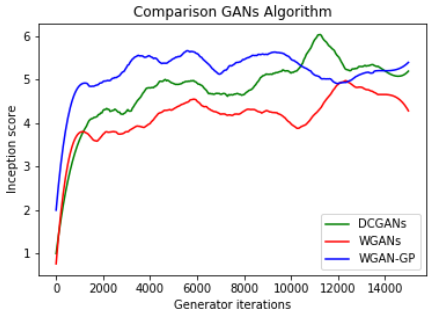


(a) DCGANs (b) WGANs (c) WGAN-GP

Figure 4: Results after running the algorithm on MNIST dataset for 15,000 epochs.



(a) Inception score (unfiltered)



(b) Inception score (filtered)

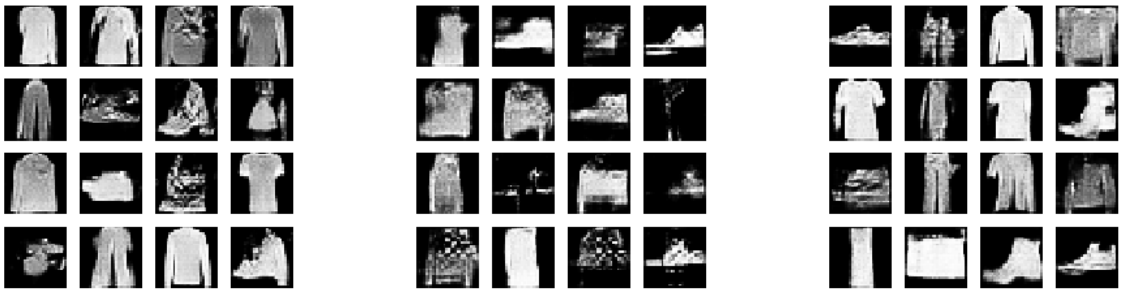
Figure 5: Comparison between the three formulations on MNIST dataset.

For MNIST dataset,[4]

- Results animation (clickable links): [DCGANs], [WGANs], [WGAN-GP]
- Colab Notebook (clickable links): [DCGANs], [WGANs], [WGAN-GP]

	Inception Score (Mean)	Inception Score (Maximum)
DCGANs	4.7723	7.40064
WGANs	4.1548	5.24146
WGAN-GP	5.1830	6.60160

Table 2: Inception scores for the three formulations on MNIST dataset.



(a) DCGANs (b) WGANs (c) WGAN-GP

Figure 6: Results after running the algorithm on MNIST Fashion dataset for 15,000 epochs.

[3]For filtering in fig 5(b), function **savgol_filter(score, 53, 3)** form python library 'signal' is used with parameters specified in brackets. This is a Savitzky Golay filter with window length = 53 and fits a polynomial of order 3.

[4]For more results on images generated between epochs, refer Appendix II.

For MNIST Fashion dataset,[5]

- Results animation (clickable links): [DCGANs], [WGANs], [WGAN-GP]

- Colab Notebook (clickable links): [DCGANs], [WGANs], [WGAN-GP]

6 Observations/Discussions

From the results above, we would like to point out some of our observations,

- The mean inception score for WGAN-GP is higher compared to WGANs and DCGANs. This indicates that the WGAN-GP formulation generates images of quality better than the other two formulations for the same number of generator training iterations (which is $15,000$ in our case).

- Comparing WGANs and WGAN-GP, we already established that WGAN-GP has higher inception score and furthermore, the generator training loss for WGAN-GP is significantly lower than WGANs. This implies that the gradient penalty term contributes for the improved training of WGANs.

- For all the three GAN formulations, we observe that the quality of generated images enhances as the training iterations progress. But when we compare WGANs, WGAN-GP with DCGANs, the discriminator and generator training loss for DCGANs show no correlation with the image quality. On the other hand, for WGANs, WGAN-GP we see that the training is more stable and the discriminator loss achieves near zero value which shows good correlation with the quality of generated images.

  Reason- The Wasserstein-1 distances induces a norm (to calculate the distance between distributions) which is more smooth and provides significant gradient value everywhere which is useful for training GANs.

Now we talk about the scope of future work. For our experiments, we used only one class of deep network architectures i.e. DCGAN but even better quality of images can be achieved if one creatively setup the different architectures for both generator and discriminator, for example, ResNet. We also see the scope of theoretical research in trying to mathematically characterize the quality of the optimal solutions from these algorithms which can lead to even better formulations in the future. There is also an opportunity to extend the work on the theme of WGAN-GP and to come up with new regularizers which helps to better impose the Lipschitz continuity assumption on the discriminator (D) function.

[5]For training results and images generated between epochs, refer Appendix III.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[3] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks, 2017.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[5] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5769–5779, USA, 2017. Curran Associates Inc.

[6] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

A Appendix I: Generator & Discriminator Network Architecture

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 12544)             1254400
_________________________________________________________________
batch_normalization (BatchNo (None, 12544)             50176
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 12544)             0
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 7, 7, 128)         819200
_________________________________________________________________
batch_normalization_1 (Batch (None, 7, 7, 128)         512
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 14, 14, 64)        204800
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 28, 28, 1)         1600
=================================================================
Total params: 2,330,944
Trainable params: 2,305,472
Non-trainable params: 25,472
_________________________________________________________________
```

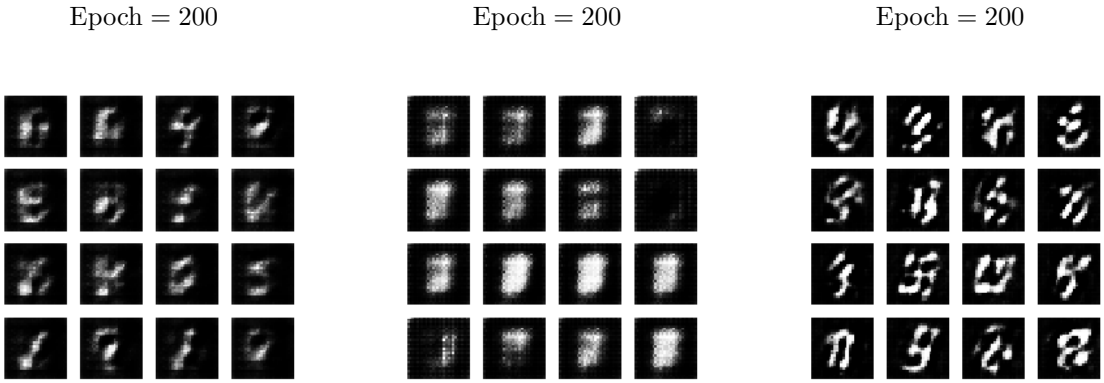
(a) Generator network

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 64)        1664
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 128)         204928
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 6273
=================================================================
Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0
_________________________________________________________________
```

(b) Discriminator network

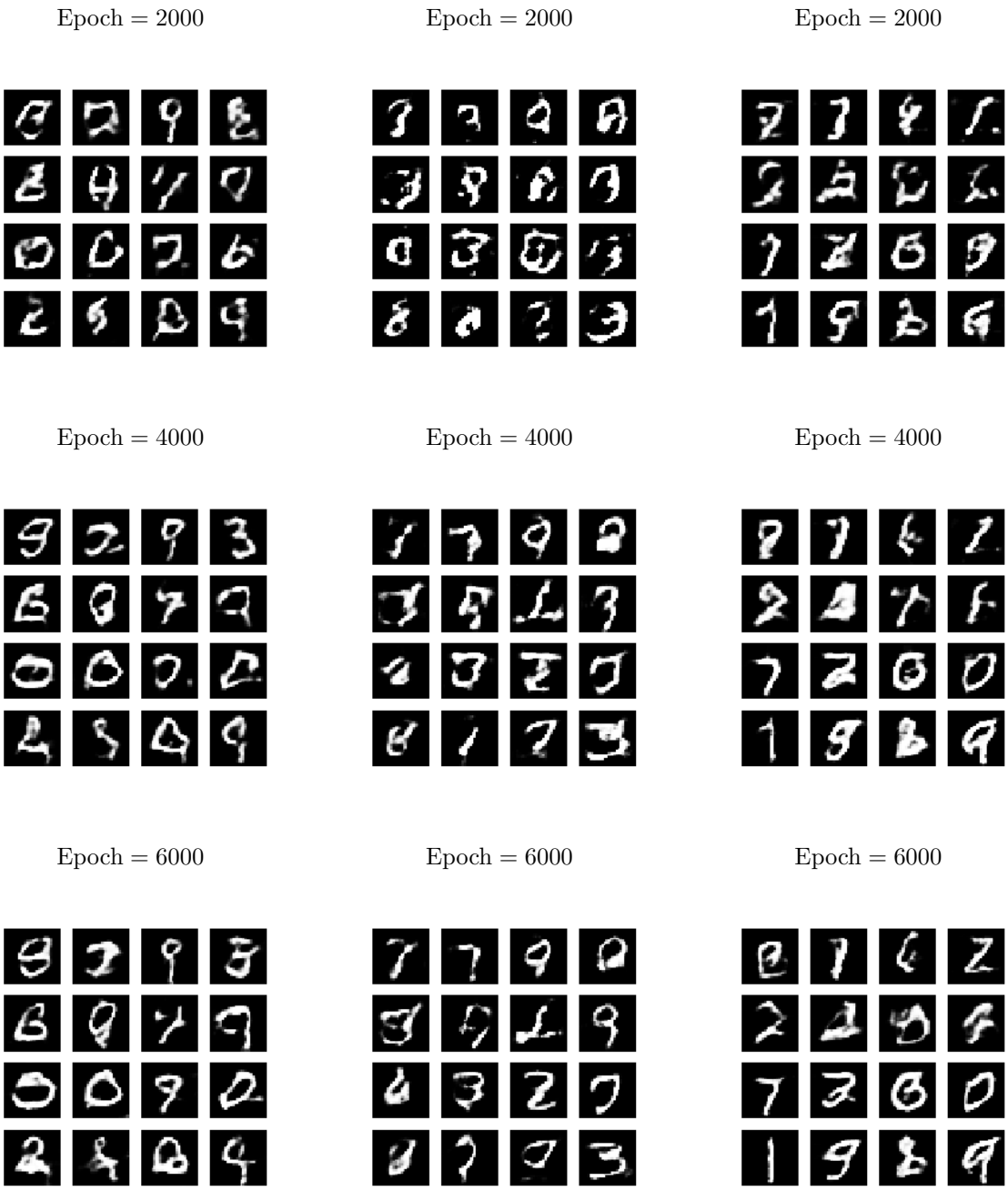
Figure 7: DCGAN architecture used for the three formulations (Source: Tensorflow).

B Appendix II : More Results on MNIST dataset



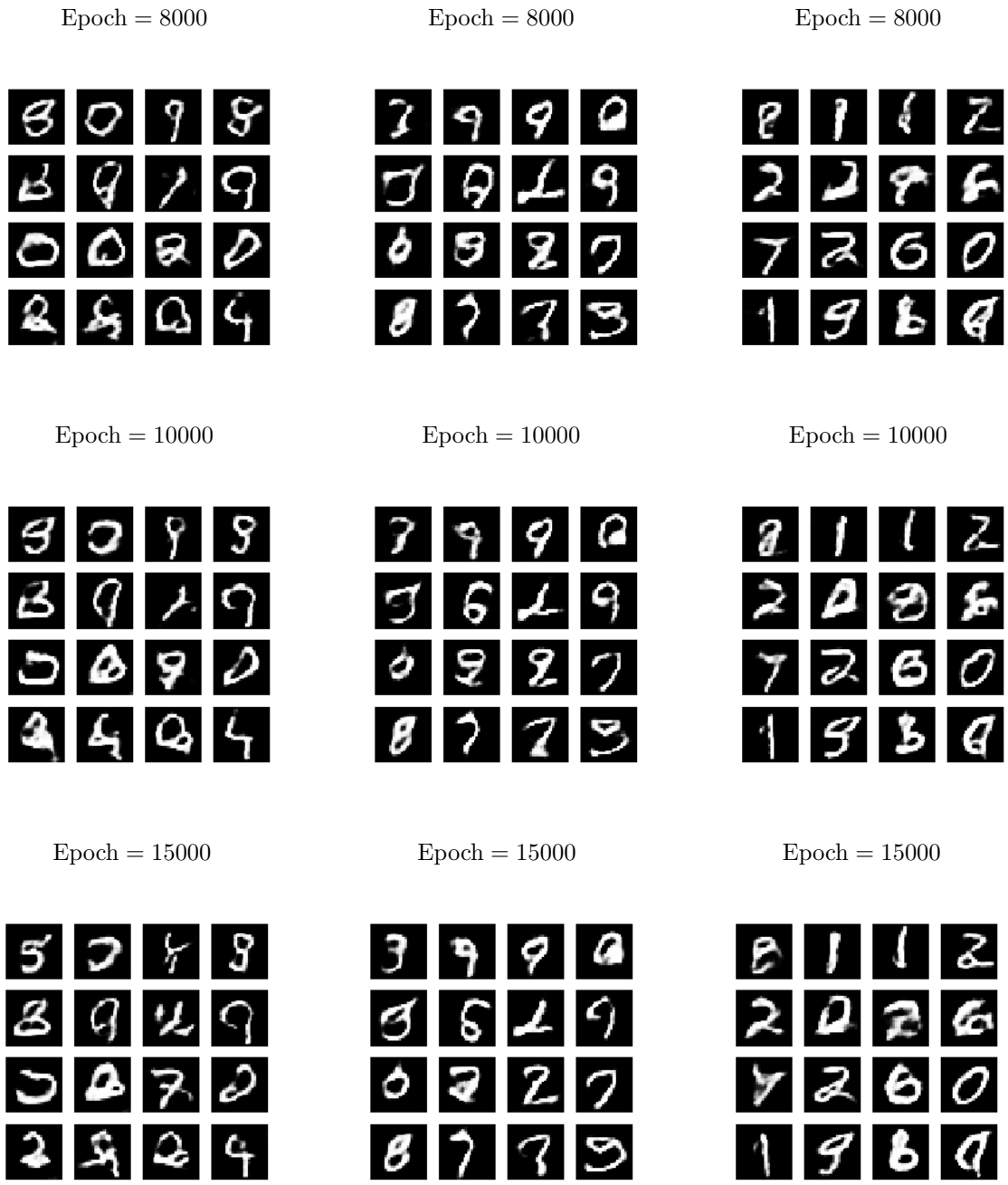
(a) DCGANs (b) WGANs (c) WGAN-GP

Figure 8: Generated images between the epochs.



(a) DCGANs (b) WGANs (c) WGAN-GP

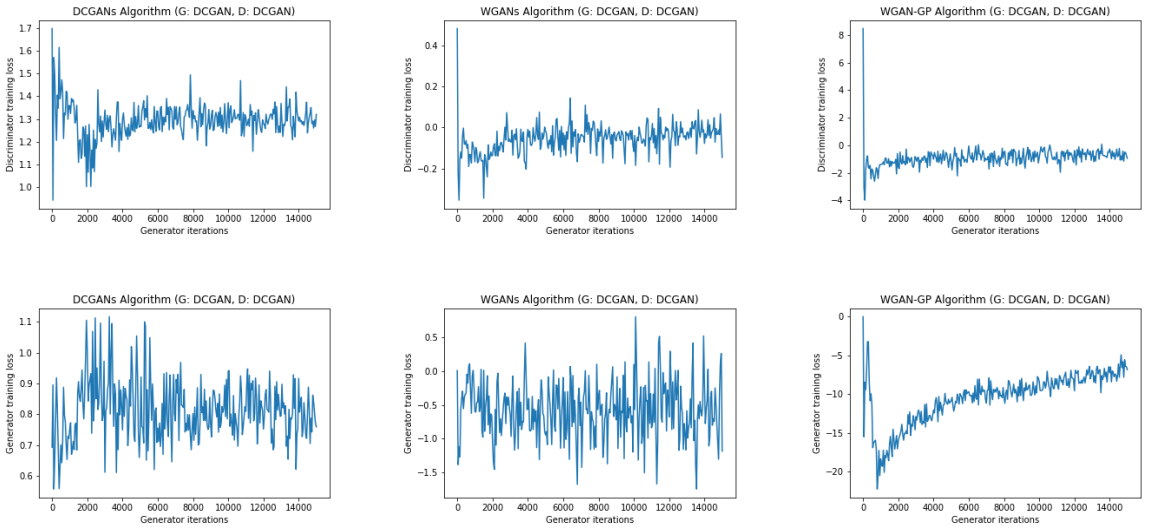
Figure 9: Generated images between the epochs.



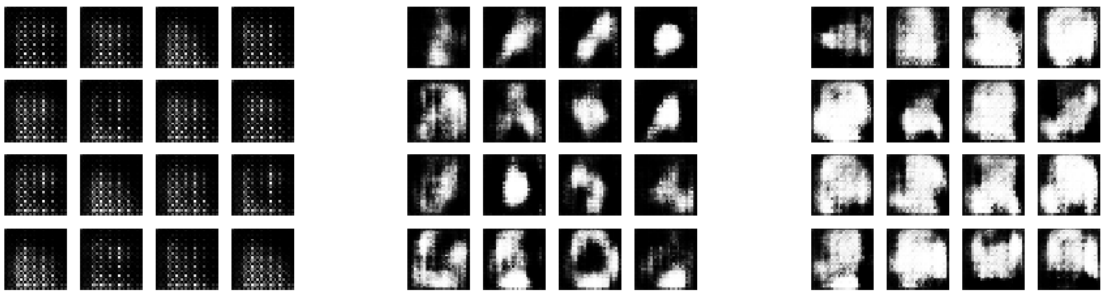
(a) DCGANs (b) WGANs (c) WGAN-GP

Figure 10: Generated images between the epochs.

C Appendix III : More Results on MNIST Fashion dataset

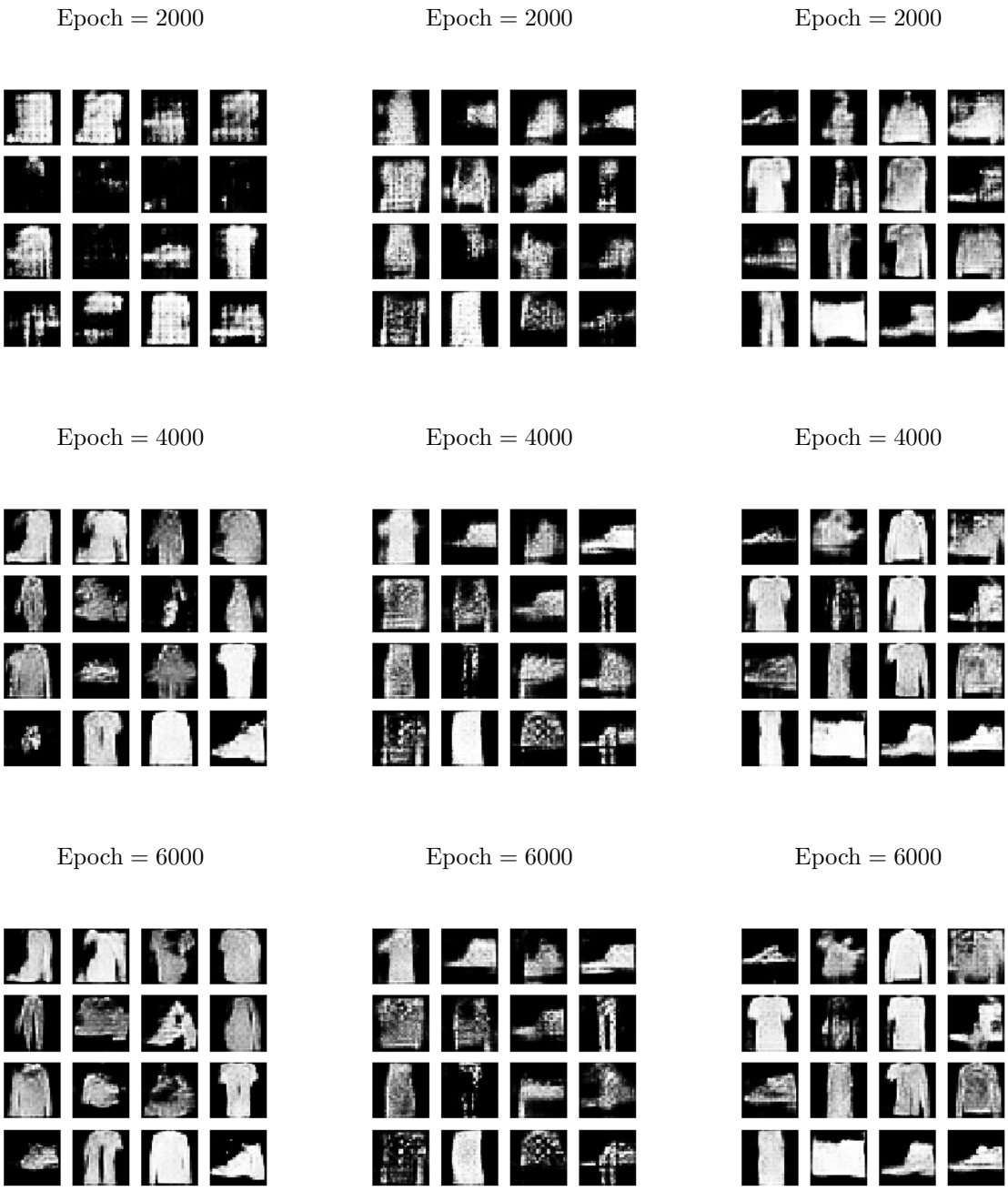


Epoch = 200 Epoch = 200 Epoch = 200



(a) DCGANs (b) WGANs (c) WGAN-GP

Figure 11: Training results and Generated images between the epochs.



(a) DCGANs (b) WGANs (c) WGAN-GP

Figure 12: Generated images between the epochs.

(a) DCGANs (b) WGANs (c) WGAN-GP

Figure 13: Generated images between the epochs.