Università di Roma "La Sapienza"
Dipartimento di Ingegneria Informatica Automatica e Gestionale
"Antonio Ruberti"
Deep Learning for Computer Vision

Stable Training for Generative Adversarial Networks

Amr Aly 1848399
Ahmed El Sheikh 1873337

Supervisor:
Prof. Fiora Pirri

March 2020

Contents

1 Introduction **1**

2 Background **3**

3 Models **4**

3.1 GANs 4

3.2 Deep Convolutional GANs 5

3.3 Wasserstein GANs 6

3.3.1 Wasserstein Distance as GAN Loss Function 6

3.3.2 Lipschitz Continuity 7

3.3.3 Wasserstein Loss Function 7

3.4 Improved Wasserstein GANs 8

4 Results **9**

4.1 Loss function evolution 9

4.2 Generated Images 10

5 Conclusion **11**

A Generated Images **12**

Bibliography **16**

1. Introduction

The problem this report is concerned with is that of unsupervised learning. Mainly, what does it mean to learn a probability distribution? The classical answer to this is to learn a probability density. This is often done by defining a parametric family of densities $P_\theta,\ \theta \in \mathbb{R}^d$ and finding the one that maximized the likelihood on our data: if we have real data examples $\{x^{(i)}\}_{i=1}^m$, we would solve the problem

$$\max_{\theta \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \log\ P_\theta(x^i)$$

If the real data distribution $\mathbb{P}_r$ admits a density and $\mathbb{P}_\theta$ is the distribution of the parametrized density P_θ, then, asymptotically, this amounts to minimizing the Kullback-Leibler divergence $KL(\mathbb{P}_r \| \mathbb{P}_\theta)$.

For this to make sense, we need the model density P_θ to exist. This is not the case in the rather common situation where we are dealing with distributions supported by low dimensional manifolds. It is then unlikely that the model manifold and the true distribution's support have a non-negligible intersection, and this means that the KL distance is not defined (or simply infinite).

The typical solution is to add a noise term to the model distribution. This is why virtually all generative models described in the classical machine learning literature include a noise component. In the simplest case, one assumes a Gaussian noise with relatively high bandwidth in order to cover all the examples. It is well known, for instance, that in the case of image generation models, this noise degrades the quality of the samples and makes them blurry.

Rather than estimating the density of $\mathbb{P}_r$ which may not exist, we can define a random variable Z with a fixed distribution $p(z)$ and pass it through a parametric function $g_\theta : Z \to X$ (typically a neural network of some kind) that directly generates samples following a certain distribution $\mathbb{P}_\theta$. By varying θ, we can change this distribution and make it close to the real data distribution $\mathbb{P}_r$. This is useful in two ways. First of all, unlike densities, this approach can represent distributions confined to a low dimensional manifold. Second, the ability to easily generate samples is often more useful than knowing the numerical value of the density.

Variational Auto-Encoders (VAEs) [1] and Generative Adversarial Networks (GANs) [2] are well known examples of this approach. Because VAEs focus on the approximate likelihood of the examples, they share the limitation of the standard models and need to fiddle with additional noise terms. GANs offer much more flexibility in the definition of the objective function, including Jensen-Shannon distance. On the other hand, training GANs is well known for being delicate and unstable.

In this report, we study two approaches to improve the stability for GANs training. First, we study different models/architectures which we expect to improve training in general situations. Second, we study various ways to measure how close the model distribution and the real distribution are, or equivalently, various ways to define a distance or divergence $\rho(\mathbb{P}_\theta, \mathbb{P}_r)$. The most fundamental difference between such distances is their impact on the convergence of sequences of probability distributions.

2. Background

Generative Adversarial Networks (GANs) [2] are a powerful class of generative models that cast generative modeling as a game between two networks: a generator network G produces synthetic data given some noise source, and a discriminator network D discriminates between the generator's output and true data. The goal for G is to maximize the probability of D making a mistake. GANs can produce very visually appealing samples, but are often hard to train. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $1/2$ everywhere.

DCGANs: In recent years, convolutional networks (CNNs) have seen huge adoption in computer vision applications. In [3], the authors introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets shows convincing evidence that deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, the learned features can be used for novel tasks – demonstrating their applicability as general image representations.

WGANs: [4] argues that the divergences which GANs typically minimize are potentially not continuous with respect to the generator's parameters, leading to training difficulty. They propose instead using the Earth-Mover (also called Wasserstein-1) distance $W(q, p)$, which is informally defined as the minimum cost of transporting mass in order to transform the distribution q into the distribution p (where the cost is mass times transport distance). Under mild assumptions, $W(q, p)$ is continuous everywhere and differentiable almost everywhere.

The WGAN value function results in a critic (discriminator) function whose gradient with respect to its input is better behaved than its GAN counterpart, making optimization of the generator easier. Empirically, it was also observed that the WGAN value function appears to correlate with sample quality, which is not the case for GANs.

WGAN-GP: The authors of [5] found that although Wasserstein GANs makes progress toward stable training of GANs, but sometimes can still generate only poor samples or fail to converge. This is due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to undesired behavior. They propose an alternative to clipping weights: penalizing the norm of gradient of the critic with respect to its input. This method performs better than standard WGANs and enables stable training of a wide variety of GAN architectures with almost no hyperparameter tuning.

3. Models

3.1 GANs

The adversarial modeling framework is most straightforward to apply when the models are both multilayer perceptrons. To learn the generator's distribution p_g over data $\boldsymbol{x}$, we define a prior on input noise variables $p_{\boldsymbol{z}}(\boldsymbol{z})$, then represent a mapping to data space as $G(\boldsymbol{z};\ \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. We also define a second multilayer perceptron $D(\boldsymbol{x};\ \theta_d)$ that outputs a single scalar. $D(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $\log(1 - D(G(\boldsymbol{z})))$. In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

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

In practice, we must implement the game using an iterative, numerical approach. Optimizing D to completion in the inner loop of training is computationally prohibitive, and on finite datasets would result in overfitting. Instead, we alternate between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly enough. The procedure is formally presented in Algorithm 1.

In practice, equation 1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 - D(G(\boldsymbol{z})))$ saturates. Rather than training G to minimize $\log(1 - D(G(\boldsymbol{z})))$ we can train G to maximize $\log D(G(\boldsymbol{z}))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

Algorithm 1: Minibatch stochastic gradient descent training of generative adversarial nets. We used $k = 1$.

for *number of training iterations* **do**

  for *k steps* **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{data}(\boldsymbol{x})$
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(\boldsymbol{x}^{(i)}) + \log \left(1 - D(G(\boldsymbol{z}^{(i)}))\right) \right]$$

  end

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D(G(\boldsymbol{z}^{(i)}))\right)$$

end

3.2 Deep Convolutional GANs

Historical attempts to scale up GANs using CNNs to model images have been unsuccessful. Authors of [3] also encountered difficulties attempting to scale GANs using CNN architectures commonly used in the supervised literature. However, after extensive model exploration they identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models.

Core to the approach is adopting and modifying three recently demonstrated changes to CNN architectures.

The first is the all convolutional net [6] which replaces deterministic spatial pooling functions (such as maxpooling) with strided convolutions, allowing the network to learn its own spatial downsampling. This approach is used in the generator, allowing it to learn its own spatial upsampling, and discriminator.

Second is the trend towards eliminating fully connected layers on top of convolutional features. The strongest example of this is global average pooling which has been utilized in state of the art image classification models. We found global average pooling increased model stability but hurt convergence speed. A middle ground of directly connecting the highest convolutional features to the input and output respectively of the generator and discriminator worked well. The first layer of the GAN, which takes a uniform noise distribution Z as input, could be called fully connected as it is just a matrix multiplication, but the result is reshaped into a 4-dimensional tensor and used as the start of the convolution stack. For the discriminator, the last convolution layer is flattened and then fed into a single sigmoid output.

Third is Batch Normalization which stabilizes learning by normalizing the input to each unit to have zero mean and unit variance. This helps deal with training problems that arise due to poor initialization and helps gradient flow in deeper models. This proved critical to get deep generators to begin learning, preventing the generator from collapsing all samples to a single point which is a common failure mode observed in GANs. Directly applying batchnorm to all layers however, resulted in sample oscillation and model instability. This was avoided by not applying batchnorm to the generator output layer and the discriminator input layer.

The ReLU activation is used in the generator with the exception of the output layer which uses the Tanh function. We observed that using a bounded activation allowed the model to learn more quickly to saturate and cover the color space of the training distribution. Within the discriminator we found the leaky rectified activation to work well, especially for higher resolution modeling. This is in contrast to the original GAN paper, which used the maxout activation.

3.3 Wasserstein GANs

3.3.1 Wasserstein Distance as GAN Loss Function

Wasserstein distance is a measure of the distance between two probability distributions. It is also called Earth Mover's distance, short for EM distance, because informally it can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution. The cost is quantified by: the amount of dirt moved $\times$ the moving distance.

When dealing with the continuous probability domain, the Wasserstein distance formula becomes:

$$W(p_r, p_g) = \inf_{\gamma \sim \prod(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}\Big[\|x - y\|\Big]$$

where, $\prod(p_r, p_g)$ is the set of all possible joint probability distributions between p_r and p_g, and the inf (infimum, also known as 'greatest lower bound') indicates that we are only interested in the smallest cost.

However, it is intractable to exhaust all the possible joint distributions in $\prod(p_r, p_g)$ to compute $\inf_{\gamma\sim\prod(p_r,p_g)}$. Thus the authors of [4] proposed a smart transformation of the formula based on the Kantorovich-Rubinstein duality to:

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)]$$

where sup (supremum) is the opposite of inf (infimum); we want to measure the least upper bound or, in even simpler words, the maximum value.

3.3.2 Lipschitz Continuity

The function f in the new form of Wasserstein metric is demanded to satisfy $\|f\|_L \leq K$, meaning it should be K-Lipschitz continuous.

A real-valued function $f : \mathbb{R} \to \mathbb{R}$ is called K-Lipschitz continuous if there exists a real constant $K \geq 0$ such that, for all $x_1, x_2 \in \mathbb{R}$,

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$$

Here K is known as a Lipschitz constant for function f. Functions that are everywhere continuously differentiable are Lipschitz continuous, because the derivative, estimated as $\frac{|f(x_1)-f(x_2)|}{|x_1-x_2|}$, has bounds.

3.3.3 Wasserstein Loss Function

Suppose this function f comes from a family of K-Lipschitz continuous functions, $\{f_w\}_{w\in\mathcal{W}}$, parameterized by w. In the modified Wasserstein-GAN, the "discriminator" model is used to learn w to find a good f_w and the loss function is configured as measuring the Wasserstein distance between p_r and p_g.

$$L = W(p_r, p_g) = \max_{w\in\mathcal{W}} \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[f_w(g_\theta(\boldsymbol{z}))]$$

Thus the "discriminator" is not a direct critic of telling the fake samples apart from the real ones anymore. Instead, it is trained to learn a K-Lipschitz continuous function to help compute Wasserstein distance. As the loss function decreases in the training, the Wasserstein distance gets smaller and the generator model's output grows closer to the real data distribution.

One big problem is to maintain the K-Lipschitz continuity of f_w during the training in order to make everything work out. The paper presents a simple but very practical trick: After every gradient update, clamp the weights w to a small window, such as $[-0.01, 0.01]$, resulting in a compact parameter space W and thus f_w obtains its lower and upper bounds to preserve the Lipschitz continuity.

Sadly, Wasserstein GAN is not perfect. Even the authors mentioned that "*Weight clipping is a clearly terrible way to enforce a Lipschitz constraint*". WGAN still suffers from unstable training, slow convergence after weight clipping (when clipping window is too large), and vanishing gradients (when clipping window is too small).

3.4 Improved Wasserstein GANs

As discussed before, because of the weight clipping WGAN still suffers from unstable training, slow convergence after weight clipping, and vanishing gradients. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used (such as in RNNs).

Another problem is that implementing a K-Lipshitz constraint via weight clipping biases the critic towards much simpler functions. The optimal WGAN critic has unit gradient norm almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$; under a weight-clipping constraint, we observe that the neural network architectures which try to attain their maximum gradient norm k end up learning extremely simple functions.

The authors propose an alternative way to enforce the Lipschitz constraint [5]. A differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere, so we consider directly constraining the gradient norm of the critic's output with respect to its input. To circumvent tractability issues, we enforce a soft version of the constraint with a penalty on the gradient norm for random samples $\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}$. Our new objective is

$$L = \underbrace{\mathbb{E}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \; \mathbb{E}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Gradient penalty}}$$

with λ a penalty coefficient, $\hat{\boldsymbol{x}} = \epsilon\tilde{\boldsymbol{x}} + (1-\epsilon)\boldsymbol{x}$ and ϵ uniformly sampled between 0 and 1.

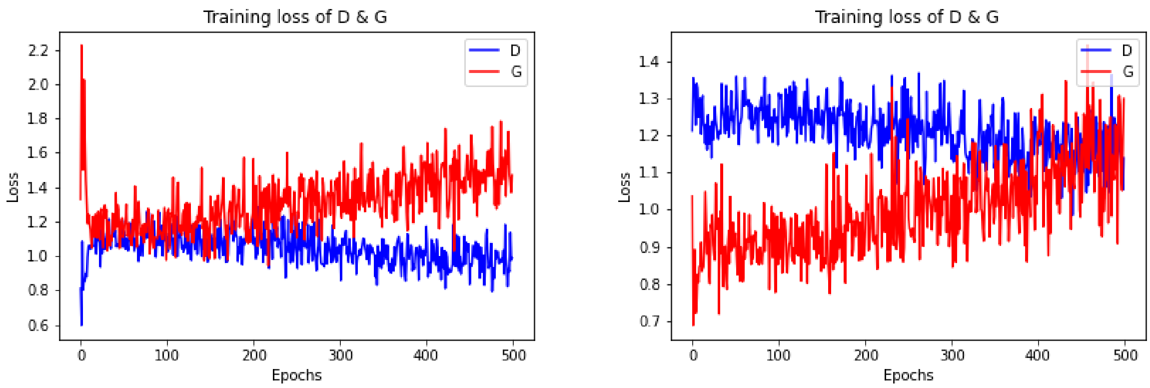
Most prior GAN implementations use batch normalization in both the generator and the discriminator to help stabilize training, but batch normalization changes the form of the discriminator's problem from mapping a single input to a single output to mapping from an entire batch of inputs to a batch of outputs. The penalized training objective is no longer valid in this setting, since we penalize the norm of the critic's gradient with respect to each input independently, and not the entire batch. To resolve this, we simply omit batch normalization in the critic in our models, finding that they perform well without it.

We encourage the norm of the gradient to go towards 1 (two-sided penalty) instead of just staying below 1 (one-sided penalty). Empirically this seems not to constrain the critic too much, likely because the optimal WGAN critic anyway has gradients with norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$ and in large portions of the region in between.

4. Results

4.1 Loss function evolution

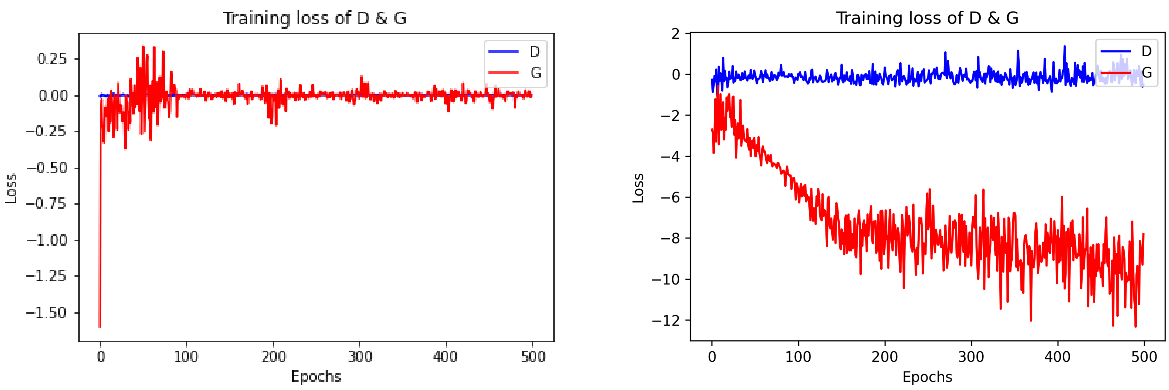
Using the loss function defined in 3.1, we trained the vanilla GAN and the DCGAN on the MNIST handwritten digits dataset. Each model was trained for 500 total epochs using the Adam optimizer with beta1 = 0.5. The generator nets used a mixture of ReLU activations and tanh activations, while the discriminator net used LeakyReLU activations.



(a) Vanilla GAN **(b)** DCGAN

Figure 4.1

As expected the discriminator loss converges around 1 (with $p(R) \approx p(F) \approx 1/2$[1], not shown in figure).



(a) WGAN **(b)** WGAN–GP

Figure 4.2

On the other hand, for the Wasserstein loss, we see that the discriminator loss indeed converges to zero (since it represents the earth mover's distance between two distributions).

[1]R = Image being real, F = Image being generated

For the WGAN, we used the RMSProp optimizer, while for the WGAN–GP we used Adam with beta1 = 0.5 and beta2 = 0.9. We also removed the batch normalization layer from the WGAN–GP generator net.

4.2 Generated Images

By the end of the training we see that all models produce realistic images, with DCGAN and WGAN–GP having the most realistic and sharpest images. This is because of the simplicity of the dataset. Were we to try a much more complicated dataset, the results would have definitely been different. Nevertheless, we can still extract some meaningful insights by looking at the evolution of the sampled images of each model (See Appendix A).

It is worth noting that the following analysis is relative to the total number of epochs we trained which was 500. In general, all these models need to be trained for thousands of epochs.

We can see the vanilla GAN slowly converging to reach better images at around 200+ epochs. Meanwhile, thanks to using convolutional layers, the DCGAN is able to extract latent visual features from the images and it starts producing good results after only 120 epochs.

When comparing the WGAN and its variant WGAN–GP, we can immediately notice the effect of hard vs soft Lipschitz constraint. For the WGAN we can see from the third epoch the formation of digit-like characters, which is due to the enforcement of the constraint through weight clipping and batch normalization. For the WGAN–GP, the case is different. Since the constraint is enforced by applying a penalty term on the loss function, we start to see the effect only after ten epochs. On the long run, however, we can see the WGAN–GP having a clear advantage over the WGAN in terms of better images generated.

5. Conclusion

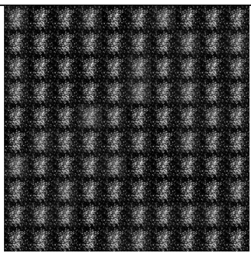
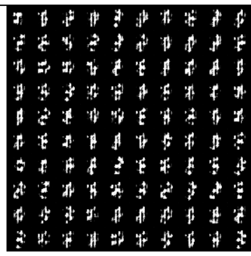
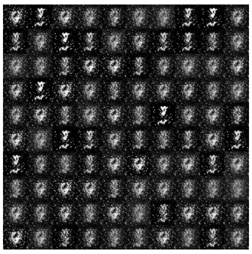
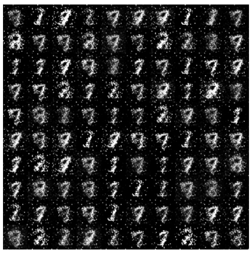
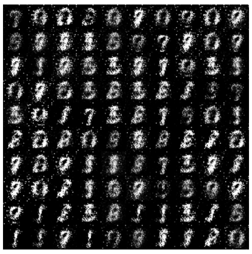
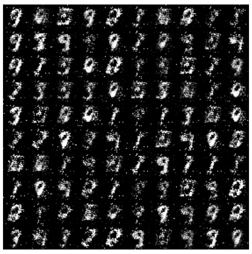
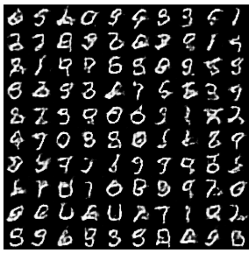
We presented two approaches to improve the stability of GAN training and producing quality output. In the first approach, we exploited a different network architecture which could better extract latent features specially in visual data. Having this advantage proved worthwhile in terms of stability and speed of convergence and quality of produced images.

In the second approach, we revised the definition of a distance between two probabilistic distributions (namely, the real data distribution and the generated data distribution). Instead of using Jensen-Shannon distance in vanilla GANs, we use the Wasserstein metric, which provides smooth measures where JSD fails. This in turn is helpful for stable training using gradient descent.

However, for the Wasserstein metric to work, a Lipschitz constraint has to be met. The WGAN uses hard rules to enforce the constraint. Namely, weight norm clipping and batch normalization, both of which proved to introduce some instabilities. That is why our last model suggest a new way of enforcing the Lipschitz constraint. By applying a penalty of the gradient norm, the learning procedure can control the norm of the weights to make sure the constraint is eventually met.

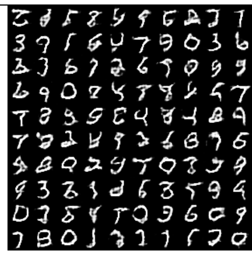
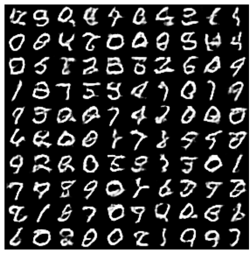
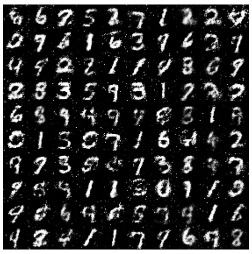
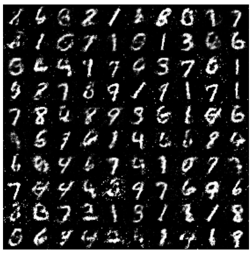
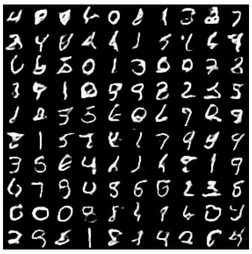
A. Generated Images

Table A.1: Generated images with epochs for different models

#	GAN	DCGAN	WGAN	WGAN–GP
#1				
#2				
#3				
#4				
#5				

Continued on next page

Table A.1 – continued from previous page

#	GAN	DCGAN	WGAN	WGAN–GP
#7				
#9				
#12				
#16				
#22				
#29				

Continued on next page

Table A.1 – continued from previous page

#	GAN	DCGAN	WGAN	WGAN–GP
#39				
#52				
#69				
#91				
#121				
#161				

Continued on next page

Table A.1 – continued from previous page

#	GAN	DCGAN	WGAN	WGAN–GP
#214				
#284				
#376				
#499				

Bibliography

[1] Kingma, Diederik P., and Max Welling. ”Auto-encoding variational bayes.” arXiv preprint arXiv:1312.6114 (2013). 1

[2] Goodfellow, Ian, et al. ”Generative adversarial nets.” Advances in neural information processing systems. 2014. 1, 3

[3] Radford, Alec, Luke Metz, and Soumith Chintala. ”Unsupervised representation learning with deep convolutional generative adversarial networks.” arXiv preprint arXiv:1511.06434 (2015). 3, 5

[4] Arjovsky, Martin, Soumith Chintala, and Léon Bottou. ”Wasserstein gan.” arXiv preprint arXiv:1701.07875 (2017). 3, 6

[5] Gulrajani, Ishaan, et al. ”Improved training of wasserstein gans.” Advances in neural information processing systems. 2017. 3, 8

[6] Springenberg, Jost Tobias, et al. ”Striving for simplicity: The all convolutional net.” arXiv preprint arXiv:1412.6806 (2014). 5