

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_θ , $\theta \in \mathbb{R}^d$ 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}_θ is the distribution of the parametrized density P_θ , then, asymptotically, this amounts to minimizing the Kullback-Leibler divergence $KL(\mathbb{P}_r \parallel \mathbb{P}_\theta)$.

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 $g_\theta : Z \rightarrow X$ (typically a neural network of some kind) that directly generates samples following a certain distribution \mathbb{P}_θ . 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 \mathbf{x} , we define a prior on input noise variables $p_z(\mathbf{z})$, then represent a mapping to data space as $G(\mathbf{z}; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g . We also define a second multilayer perceptron $D(\mathbf{x}; \theta_d)$ that outputs a single scalar. $D(\mathbf{x})$ represents the probability that \mathbf{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 $\log(1 - D(G(\mathbf{z})))$. In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

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

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, $\log(1 - D(G(\mathbf{z})))$ saturates. Rather than training G to minimize $\log(1 - D(G(\mathbf{z})))$ we can train G to maximize $\log D(G(\mathbf{z}))$. 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 $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{data}(\mathbf{x})$
- Update the discriminator by ascending its stochastic gradient:

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

end

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

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

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 distri-

bution 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:

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

where, $\Pi(p_r, p_g)$ 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 $\Pi(p_r, p_g)$ to compute $\inf_{\gamma \sim \Pi(p_r, p_g)}$. 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 $f : \mathbb{R} \rightarrow \mathbb{R}$ 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 .

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

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 \mathcal{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-Lipschitz 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 $\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}$. Our new objective is

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

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

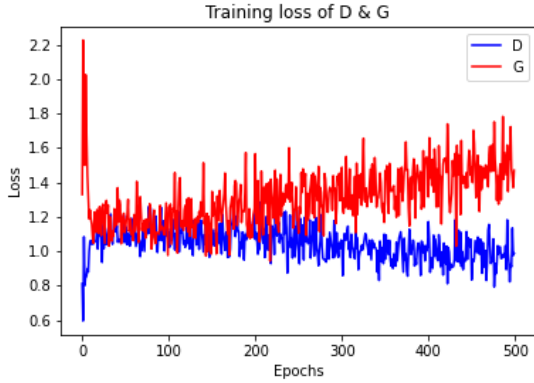
Most prior GAN implementations use batch normalization in both the generator and the discriminator to help stabilize training, but batch normalization changes the form of the discriminator’s problem from mapping a single input to a single output to mapping from an entire batch of inputs to a batch of 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.

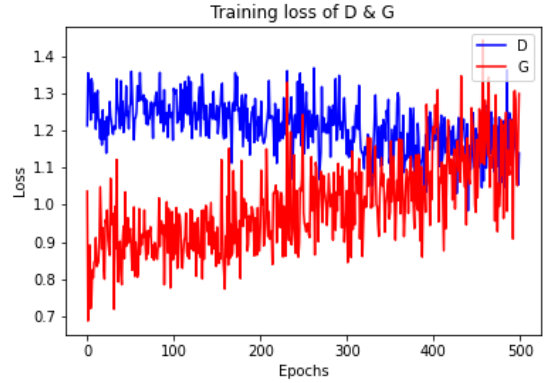
4. Results

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 $\text{beta1} = 0.5$. The generator nets used a mixture of ReLU activations and tanh activations, while the discriminator net used LeakyReLU activations.



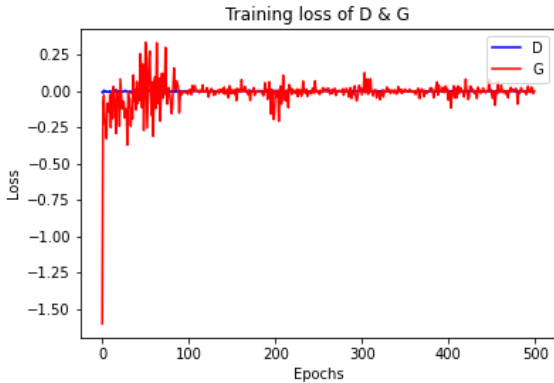
(a) Vanilla GAN



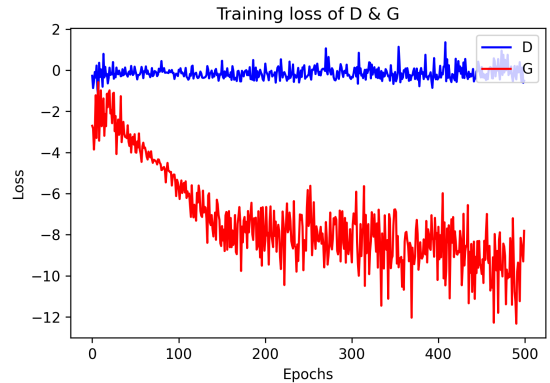
(b) DCGAN

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).

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

For the WGAN, we used the RMSProp optimizer, while for the WGAN-GP we used Adam with $\text{beta1} = 0.5$ and $\text{beta2} = 0.9$. 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	0377987974 7974833747 0961818254 0878979554 3137997849 5485196978 481278183 4373289577 1180808577 3759512953	0295721056 7145867419 350358318 6978740509 4280763798 4866378775 1521831151 7266114059 7060136591 9965775463	9427350346 8918070244 9003911357 0598130038 6771082304 7089718487 7711970531 9104756016 757256105 3095737251	2852595271 3716479036 2367267099 6902578457 7835848184 7524923875 5292570359 9321263594 0384703609 7804311720
	4101617688 4242371273 3722711219 4532888444 9620920915 7753477474 2081781189 1244273174 3595773783 3531707158	0514751490 7694109318 1288773160 0912513063 4586289620 9701125297 0531739917 1520809304 8305578904 1898134449	0930270103 1707476630 7617554061 1325194894 7470194824 7592846054 1666842393 0006473577 9182844417 0607318203	4908484211 0042000344 0522382609 1875347079 7302742090 4202173558 9280323501 7789076278 2167074062 608021997
	6625271224 0741637427 4922170809 2835931272 6894978819 0130716442 9738473847 8891180110 9649185918 4247177698	1394505799 1172923401 0187793013 1639441763 0109791490 7790195269 4507371947 1207081622 8301235080 464833632	1851190530 2165282026 1637317098 4207469508 9353328370 7800820729 6731771211 6317062070 4742185954 870336206	6281180899 9784914074 0718832002 2794164647 9029860166 9181762681 6881244357 8637331046 477019723 6178147097
#12	1662138017 4107101306 5441703761 9278971717 7828936166 157414684 6046741077 7944397696 3621131818 0674241419	5496304459 1587489108 4831848883 7010057121 9274120997 0170180710 1007726129 9064249113 8587212741 0901402848	4044011327 2404415464 6650130022 3910992235 1455602709 2153417999 3564111519 6794365238 0008114004 2951814204	6973635432 4681687371 2521267316 0523307158 6325933148 1528989352 0275131323 3544790973 4075776527 588771326
	4418319194 5961912451 1875981884 1996412195 7830069437 9740113741 7101540422 1026157891 028157891 5691461644	8659871714 0894372469 7410418502 935548820 0272123057 6147937303 4187296604 1104980864 0371280841 024479642	9879030678 6583294664 6102398181 8627197241 9451850324 8916003070 8625711147 2076093738 2762497390 1673099239	7871108924 1620189141 2791731829 7533400399 4208713989 2234401179 1857231042 3952351280 0175417180 7535428721
	039706245 4458590937 1911643640 7656083017 790363240 1367876733 5141392818 8840135956 6725184871 4844135587	5050483067 8067932344 1116782745 1009163063 1273762472 1207042074 0984238418 0919117519 7545938351 0970849707	0887813707 8272466094 7982623489 5294570927 9015045271 0713694640 9375061293 7991283455 6843025265 4336035960	1469273913 0058949016 1316627375 6411870438 4401690991 7656736914 7000971352 1576280166 5131216729 2941452059
#29				

Continued on next page

Table A.1 – continued from previous page

#	GAN	DCGAN	WGAN	WGAN-GP
#39	3853739181 4541085228 1430476211 6757996608 7919674697 2191632810 2174980721 7189182845 9113711018 1179131072	9014616469 0847477450 7549407385 2942075661 1364578379 3760003506 1700008110 6296277002 237600133 8713441580	7232502998 8011772617 1967228040 6375521969 4235947552 3004073088 0052552345 1611024499 5843533514 3064069447	9361014087 4523027203 5624612931 8117214514 0974398996 2539249811 0528564047 1842135460 5838822678 0223574974
	5183771728 1824982517 3942715194 7268898111 6961195121 1191840746 8811168433 1876589988 9811335174 1383842319	2166229192 4371415083 1043054208 4055775228 2978506611 0114310710 7313840432 2030218874 9940027654 9497027451	3921304987 0144487524 0950455444 5176920532 3918827692 2191402315 3474461198 7302530897 8458046644 1123178081	7455231396 1309274070 8078821840 4869108097 0961915302 875106621 1757763187 4004150384 8711484021 831633202
	1462759114 4668509821 4368721191 1019973920 311233114 8374993402 1674613287 2137268118 3984664914 2715166144	2948617807 1594846433 2567649401 7380830305 4201482774 2450128381 2770231902 4271695470 0461819002 3135927454	0344683900 9043699174 5719401694 6342792900 1490121474 8364179877 7636624138 5700920158 6509877088 2286177730	09374482538 5978496881 29118722600 6087839720 9197174236 9013104730 2828038425 9901988079 8390784037 8133473023
#69	6074114111 4361044363 9181704794 9644141483 7541439719 8648744469 6319731261 9146564802 3496760812 7830692311	7051196758 4316546733 8578067820 7309148143 7032750096 1636466408 1264715981 1748947817 1234805289 7900016322	3877823550 3295200704 6713090715 2027677033 1968478804 3669388688 0000010823 9164409497 0941079427 1739678403	8245731742 8610162220 9110840504 3464289356 1064204114 5062475544 1290073855 7212726136 9617239233 4167287695
	0409421716 9903168748 1104741877 9671102700 4127711193 9136431798 7870958799 0088131785 7167978627 6491976331	3476474412 9436541130 9768754823 4040481595 3658205771 1181143946 0488077759 4173035296 1209201860 7158542930	4897177503 6786221812 2102838673 7939827605 3422291296 6780258170 7091005567 9418218975 1516013584 7415603104	1104654844 9849475492 7815305571 7856014724 0704678812 2680700947 0540013468 1214135863 7699877658 5106303022
	1366689181 1447175912 6795129148 7731112794 1017849940 5396719973 4168981891 9801103194 2411928476 0817265381	6248419526 1146388874 9514764625 0675709082 2645232273 5858795331 8429328193 7594944717 2000630702 1192466701	1352767174 8273041484 4108861623 5025136740 2675190298 2998976060 4725522026 2537019534 0605807162 3581219188	3298404601 0348581620 1313426967 2898182813 6434952806 2281774116 4894145746 4321736008 9408282505 4772130291
#161	0817265381	1192466701	3581219188	4772130291

Continued on next page

Table A.1 – continued from previous page

#	GAN	DCGAN	WGAN	WGAN-GP
#214	4940710140 3576789581 1812912117 1798875473 6496181591 2147617036 3984517901 8696153777 1094117979 9577811153	4171730113 4053244850 4417867452 1093613916 6305813747 2140158650 1687372484 8606594363 5814369617 6457611387	6960812774 1892044736 5601199103 2005768750 8102170895 8682164250 5393055891 0401938470 7180438942 5779303827	5246758989 1906545949 7715773077 0621907009 1261402164 2249420782 2675414686 4145038680 3442459858 0944865824
	5014397111 7819907711 6238667271 4111647185 7984910387 3673819267 9111482729 1197791267 2894406779 1465767811	0280904647 5871147316 7445632127 7314912178 7592165126 1726675969 3283186700 2278906733 5726316492 4382706910	2101501392 6445747203 2873080176 6549525678 5797327619 3740241096 3099716717 8048357194 2008553763 2464009334	4045603636 6816517875 9061434428 0027504059 4986590147 3485676097 4541567470 6718668637 1548510985 8782748882
#284	4970117969 0619731019 7539741167 7941174140 1294294187 1957473650 8199613810 6578715528 7691768919 1446106717	3859052873 4814290754 9853413397 0724622812 8761153748 6414371934 6161739680 0550155011 7209217589 8981844005	8644716014 7732788646 9904736148 6857296984 1891123465 5900459202 2334923519 6449324769 4199190360 5971947690	6703536778 6678520091 1190729334 6783778125 6249477027 6739783533 8455243161 8729114871 1490118209 8164392238
#376	5947110644 4435139687 2424757117 4918761911 1711771667 9978918736 9771177011 1011177141 1151801611 8701760931	5393174913 1110890974 3121134658 7433727510 1966397141 1597364501 1548443969 0790776897 4531871835 5761406227	0383284925 0127864318 0194254305 9164050922 0647891754 1103872620 0504324233 1670746019 1264772853 5659455313	3302825708 8761389287 3737460010 6923423907 117172783 7369336615 8730676381 4011804389 7572476145 7460920869
#499	5947110644 4435139687 2424757117 4918761911 1711771667 9978918736 9771177011 1011177141 1151801611 8701760931	5393174913 1110890974 3121134658 7433727510 1966397141 1597364501 1548443969 0790776897 4531871835 5761406227	0383284925 0127864318 0194254305 9164050922 0647891754 1103872620 0504324233 1670746019 1264772853 5659455313	3302825708 8761389287 3737460010 6923423907 117172783 7369336615 8730676381 4011804389 7572476145 7460920869

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