Architecture and algorithm development for generative adversarial networks

Yazici, Yasin

2020

Yazici, Y. (2020). Architecture and algorithm development for generative adversarial networks. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/142989>

<https://doi.org/10.32657/10356/142989>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 13:30:16 SGT

ARCHITECTURE AND ALGORITHM DEVELOPMENT FOR GENERATIVE ADVERSARIAL NETWORKS

YASIN YAZICI

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING

2020

Architecture and Algorithm Development for Generative Adversarial Networks

Yasin Yazıcı

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2020

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

22 June 2020

.....

Date

.....

Yasin Yazıcı

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

22 June 2020

.....

Date

.....

Assoc Prof Yap Kim Hui

Authorship Attribution Statement

This thesis contains materials from 3 papers published in the following peer-reviewed conferences in which I am listed as the first and corresponding author.

Chapter 4 is published as [Y. Yazici, C.S. Foo, S. Winkler, K.H. Yap, G. Piliouras, V. Chandrasekhar. The Unusual Effectiveness of Averaging in GAN Training. In Seventh International Conference on Learning Representations \(ICLR 2019\)](#)

The contributions of the co-authors are as follows:

- I came up with the idea and suggested an explanation to the phenomenon.
- I designed and conducted all the experiments.
- I prepared the manuscript draft. The manuscript was revised by C.S. Foo, S. Winkler, K.H. Yap, G. Piliouras, and V. Chandrasekhar.
- C.S. Foo, S. Winkler, K.H. Yap, G. Piliouras, and V. Chandrasekhar provided discussion to refine the idea and guidance for the experimentation.
- C.S. Foo and G. Piliouras helped me with the theoretical analysis.

Chapter 5 is published as [Y. Yazici, C.S. Foo, S. Winkler, K.H. Yap, V. Chandrasekhar. An Empirical Analysis of Overfitting and Mode Drop in GAN Training. In 27th IEEE International Conference on Image Processing \(ICIP 2020\)](#)

The contributions of the co-authors are as follows:

- I came up with the idea.
- I designed and conducted all the experiments.
- I prepared the manuscript draft. The manuscript was revised by C.S. Foo, S. Winkler, K.H. Yap, and V. Chandrasekhar.
- C.S. Foo, S. Winkler, K.H. Yap, and V. Chandrasekhar provided discussion to refine the idea and guidance for the experimentation.

Chapter 6 is published as [Y. Yazici, S. Winkler, K.H. Yap. Autoregressive Generative Adversarial Networks. In Sixth International Conference on Learning Representations \(Workshop Track\), \(ICLR-W 2018\)](#)

The contributions of the co-authors are as follows:

- I came up with the idea.
- I designed and conducted all the experiments.
- I prepared the manuscript draft. The manuscript was revised by S. Winkler and K.H. Yap.
- S. Winkler and K.H. Yap provided discussion to refine the idea and guidance for the experimentation.

22 June 2020

.....

Date

.....

Yasin Yazıcı

Acknowledgements

First and foremost, I would like to express my greatest gratitude to my supervisors, Prof. Kim-Hui Yap, Prof. Stefan Winkler, and Dr. Vijay Ramaseshan Chandrasekhar for their guidance, encouragement, and support. Without them this thesis would not be possible. Also, I wish to express my deepest gratitude to my mentor Chuan-Sheng Foo, who closely followed my works, gave detailed feedback, and guided me along the way.

I wish to express my deepest gratitude to Agency for Science, Technology and Research (A*STAR) for providing SINGA scholarship, and to Nanyang Technological University for giving me a chance to pursue my PhD degree. I am grateful to the Rapid-Rich Object Search (ROSE) Lab, Institute for Infocomm Research (I2R), and Advanced Digital Sciences Center (ADSC) for providing me a stimulating research environment and computational resources. I am indebted to National Supercomputing Centre, Singapore for providing computational resources for my research. I appreciate all the support I received from my friends in NTU, ROSE, I2R, and ADSC. It made the journey much more satisfying and rewarding.

I would like to thank Dr. Yasin Mamatjan and Prof. Kai-Hsiang Chuang for supporting and encouraging me to start a PhD degree.

Last but not least, I would like to thank my family for their constant support and encouragement.

“If I had one hour to save the world, I would spend 55 minutes defining the problem and only five minutes finding the solution.”

—Einstein, Albert

To my dear family

Contents

Acknowledgements	ix
Abstract	xvii
List of Figures	xix
List of Tables	xxvii
Symbols and Acronyms	xxix
1 Introduction	1
1.1 Introduction to GANs	1
1.1.1 Training Procedure and Visualizations	3
1.2 Applications of GANs	6
1.3 Challenges	8
1.3.1 Instability/ non-Convergence	8
1.3.2 Overfitting/Generalization	9
1.3.3 Mode Drop/Collapse	9
1.3.4 Multi Distribution Modeling	9
1.3.5 Improving Data Quality	10
1.4 Proposed Methods	10
1.5 Thesis Contributions	12
1.6 Outline of the Thesis	14
2 Literature Review	15
2.1 Related Works	15
2.1.1 Multi-generator/discriminator GANs	15
2.1.2 Conditional GANs	16
2.1.3 Mixture of Distributions with GANs	16
2.1.4 Stability of GANs Training	17
2.1.5 Overfitting /Generalization	18
2.1.6 Mode Drop (or Collapse)	19
2.1.7 Convergence	20
2.1.8 Improving Data Quality	20

2.2	Related Datasets	21
2.3	Evaluation Metrics	25
2.3.1	Inception Score and Fréchet Inception Distance	25
2.3.2	Classification Performance	26
2.3.3	Evaluation of Overfitting	27
3	VennGAN: Discovering Commonalities and Particularities of Multiple Distributions	29
3.1	Introduction	29
3.2	Method	32
3.2.1	Conceptual Relation to Venn Diagrams	33
3.2.2	VennGAN Modeling	35
3.2.2.1	Generator Side	35
3.2.2.2	Discriminator Side	35
3.2.2.3	Additional constraint	36
3.2.3	Network Architecture	38
3.3	Experiments	40
3.3.1	Implementation Details	41
3.3.1.1	Objective Details	41
3.3.1.2	Optimization and Hyperparameters	41
3.3.2	Quantification of Results	42
3.3.3	Illustrative Examples	43
3.3.4	Main Experiments	45
3.3.4.1	Conditional Generator versus Independent Generators	50
3.3.4.2	Effect of the Classifier	50
3.3.4.3	CelebA Results	50
3.3.4.4	Overall Discussion	52
3.4	Application: Incorporating External Knowledge	53
3.5	Summary	55
4	The Unusual Effectiveness of Averaging in GAN Training	57
4.1	Introduction	57
4.2	Method	58
4.3	Why Does Averaging Work?	59
4.3.1	Global Analysis for Simple Bilinear Games	60
4.3.2	Local Stability Analysis for GANs	62
4.4	Experiments	65
4.4.1	Mixture of Gaussians	68
4.4.2	Real-world Datasets	70
4.4.3	CIFAR-10	72
4.4.4	CelebA	74
4.4.5	STL-10 and ImageNet	80
4.5	Ablation Studies	81

4.5.1	Parameterized Online Averaging	82
4.5.2	Quantitative Effects of Different β for EMA	83
4.6	Application to Image In-painting	84
4.7	Summary	85
5	An Empirical Analysis of Overfitting, Mode Drop and Convergence in GANs Training	87
5.1	Introduction	87
5.2	Method	88
5.2.1	Why to Remove Stochasticity?	90
5.2.2	Network Architectures	91
5.3	Experiment Details	92
5.3.1	Quantitative Metrics	93
5.4	Results	94
5.4.1	Effect of the Batch Size	94
5.4.2	Noisy Latent Code	100
5.4.3	Importance of Capacity	104
5.5	Summary	104
6	Autoregressive Generative Adversarial Networks	107
6.1	Introduction	107
6.2	Relation to Other Methods	109
6.3	Method	110
6.3.1	Preliminary	110
6.3.2	Autoregressive GAN (ARGAN)	111
6.3.3	Patch-GAN + S-ARGAN (PARGAN)	114
6.3.4	Network Architectures	116
6.4	Experiments	118
6.4.1	Qualitative Results	120
6.4.2	Quantitative Results	120
6.5	Summary	122
7	Conclusion and Future Work	123
	List of Author's Publications	127
	Bibliography	129

Abstract

Generative Adversarial Networks (GANs) are recently invented generative models which can produce highly realistic samples such as images. In GANs framework, a generator and a discriminator play a game against each other. The discriminator aims to find differences between the data distribution and the generator's distribution, while the generator aims to trick the discriminator. Both players update their own parameters, in an iterative manner, to satisfy their own objective. By playing this game, the generator learns to produce samples that are similar to the ones in the data distribution.

Despite the success of GANs, there are various problems with them such as non-convergence (instability) of the objective function, mode drop (not modeling certain parts of the distribution), lack of efficient multi-distribution modeling, and generation quality. In this thesis, we have proposed novel algorithms, architectures and provided theoretical/empirical analysis to mitigate these problems.

In the first work of the thesis, we address the question of how to model common and unique aspects of multiple distributions. Finding these aspects can be used for a subsequent task e.g. what can be found in distribution A but not in distribution B? We propose a new GANs design called VennGAN which models multiple distributions effectively and discovers their commonalities and particularities. Each data distribution is modeled with a mixture of multiple generator distributions. The generators are partially shared between the modeling of different data distributions: shared ones capture the commonality of the distributions, while non-shared ones capture unique aspects of them. We thoroughly evaluate our proposed VennGAN and show its effectiveness on various datasets (MNIST, Fashion-MNIST, CIFAR-10, CelebA) and settings with compelling results. Additionally, we illustrate how this method can be used to incorporate external (prior) knowledge to distribution modeling.

In the second work, we theoretically and empirically examine two different techniques for parameter averaging in GANs training in order to mitigate non-convergence

of the objective function. Moving Average (MA) computes the time-average of the network parameters, whereas Exponential Moving Average (EMA) computes an exponentially discounted sum of the network parameters. Whilst MA is known to lead to convergence in bilinear settings, to the best of our knowledge, we provide the first theoretical analysis in support of EMA. We show that EMA shrinks the cycles around the equilibrium for simple bilinear games and also enhances the stability of general GANs training. We establish experimentally that both techniques are strikingly effective in the general (non-convex-concave) GANs setting as well. Both improve commonly used Inception Score and Frechet Inception Distance on different architectures and for different GANs objectives. We provide comprehensive experimental results across a range of datasets, mixture of Gaussians, CIFAR-10, STL-10, CelebA and ImageNet, to demonstrate its effectiveness. Also, we have applied EMA into image in-painting problem which results in smoother, more realistic images with less artifacts.

In the third work, we have studied important questions in GANs literature such as overfitting, mode drop and non-convergence from empirical perspective on high dimensional distributions. Mode drop is phenomenon of leaving out certain modes of the data distribution. Overfitting has important implications for privacy concerns as the generator can faithfully memorize the training samples which might be a private data. While mitigating mode drop and non-convergence leads to good distribution modeling. We have shown that, given enough capacity, when stochasticity is removed from the training (optimization), GANs can overfit, show almost no mode drop and reduce non-convergence rates significantly. These results shed a light on important characteristics of GANs training and dismiss certain beliefs in the current literature such as GANs do not memorize the training set, mode drop and non-convergence are mainly due to theoretical characteristic of GANs objective rather than an optimization issue.

In the fourth work, we aim to improve the quality of image generation with GANs by refining the discriminator architecture and its objective. We have proposed a novel GAN architecture & objective, called Autoregressive GANs (ARGAN), and successfully trained it on CIFAR-10, STL-10, and CelebA datasets. Different from original GANs setting, our method models the latent distribution of real samples to learn feature co-occurrences implicitly. Through efficient factorization, we come

up with two version of our model, namely Spatial-wise ARGAN and Channel-wise ARGAN. Further, we have combined our proposed Spatial-wise ARGAN with Patch-GAN to further improve the stability of training and the quality of image generation. Our results show that the proposed model outperforms DCGAN, EBGAN, and WGAN on the same architecture.

List of Figures

1.1	The Generator of GANs framework. It is parameterized as a multi-layer neural network.	4
1.2	The Discriminator of GANs framework. It is parameterized as a multi-layer neural network.	4
1.3	Improving the generator. The parameters of the generator updated to increase the output of the discriminator for the generated samples.	5
1.4	BigGAN as an application of GANs. It can generate realistic images belong to specific categories. The figure is taken from Brock et al. [1]	7
1.5	CycleGAN as an application of GANs. It can map images from one domain to another with similar semantics and without paired samples. (left) Monet paintings to landscape photos, (center) zebra to horse, (right) summer to winter photos, (bottom) photos to different painting styles. The figure is taken from Zhu et al. [2]	7
2.1	Random samples from MNIST dataset.	22
2.2	Random samples from Fashion-MNIST dataset.	22
2.3	Random samples from SVHN dataset.	23
2.4	Random samples from CIFAR-10 dataset.	23
2.5	Random samples from STL-10 dataset.	23
2.6	Random samples from CelebA dataset.	24
2.7	Random samples from FFHQ dataset.	24
2.8	Random samples from ImageNet dataset.	25
3.1	Motivating illustration. (top) samples from two real data distributions, S_1 and S_2 . (bottom) generated samples via our proposed model where each set operation models the correct portion of the data i.e. commonalities ($S_1 \cap S_2$) and particularities ($S_1 \setminus S_2$ and $S_2 \setminus S_1$) of the data distributions. The commonalities and particularities are captured at an abstract (semantic) level.	30
3.2	Three different configurations of Venn diagrams with either 2 or 3 sets. We annotate each region with r_i and each diagram with d_i as we refer to them later in this chapter.	31

3.3	An illustration of Venn GAN architecture for d_2 type diagram. For simplicity, we have used a conditional generator, however K independent generators can be used in the same way. Each condition of the generator represents corresponding region in Fig. 3.2; other Venn diagrams can be constructed in a similar way. Each discriminator receives union of its corresponding set's regions. $+$ sign is union over its incoming regions, and the union is fed into a discriminator as fake class. Each discriminator receives different true data distribution and with GAN objective it compares them to the mixture of the regions.	37
3.4	Samples from the data distributions for the illustrative example. Each distributions equally mixes 4 out of 7 Gaussians.	43
3.5	Generated regions for the illustrative example. Annotation of the regions w.r.t. d_2 of Fig. 3.2: r_1 is blue, r_2 is orange, r_3 is green, r_4 is brown, r_5 is purple, r_6 is red and r_7 is pink.	44
3.6	MNIST, Fashion-MNIST, CIFAR-10 results for case A.	46
3.7	MNIST, Fashion-MNIST, CIFAR-10 results for case B.	47
3.8	MNIST, Fashion-MNIST, CIFAR-10 results for case C.	48
3.9	CelebA case A results: $S_1 \setminus S_2$ is only male figures, $S_2 \setminus S_1$ is only female figures, $S_1 \cap S_2$ is intersection.	51
3.10	CelebA case B results: $S_1 \setminus S_2$ is only female figures, $S_1 \cap S_2$ is only male figures.	52
4.1	MA versus EMA for various values of β	61
4.2	Effect of averaging parameters for Mixture of Gaussians. From top to bottom: without average, EMA, MA, OMD, CO, Zero-GP. . . .	71
4.3	Generation for CIFAR-10 dataset. (left) Without averaging, (right) with EMA	73
4.4	CIFAR-10 FID and IS scores during training. Setting: original GAN objective, conventional architecture, $n_{dis} = 1$	73
4.5	CIFAR-10 FID and IS scores during training. Setting: original GAN objective/ conventional architecture/ $n_{dis} = 5$	74
4.6	CIFAR-10 FID and IS scores during training. Setting: WGAN-GP objective/ conventional architecture/ $n_{dis} = 1$	74
4.7	CIFAR-10 FID and IS scores during training. Setting: WGAN-GP objective/ conventional architecture/ $n_{dis} = 5$	74
4.8	CIFAR-10 FID and IS scores during training. Setting: original GAN objective/ ResNet architecture/ $n_{dis} = 5$	75
4.9	CIFAR-10 FID and IS scores during training. Setting: WGAN-GP objective/ ResNet architecture/ $n_{dis} = 1$	75
4.10	Comparison of EMA, OMD, and CO on CIFAR-10.	76
4.11	Generation for CelebA dataset for various β values at 100k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA	77

4.12	Generation for CelebA dataset for various β values at 150k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA	77
4.13	Generation for CelebA dataset for various β values at 200k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA	78
4.14	Generation for CelebA dataset for various β values at 250k iterations. From top to bottom: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA.	78
4.15	Generation for CelebA dataset for various β values at 300k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA	79
4.16	Generation for CelebA dataset for various β values at 350k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA	79
4.17	Generation for CelebA dataset for the same 2 noise samples from 50k to 200k with 10k intervals. (top 2 rows) without averaging, (bottom two rows) with EMA $\beta = 0.9999$	80
4.18	Generation for STL-10 dataset after 500k iteration. (Left): w/o averaging, (Right): with EMA	81
4.19	Generation for ImageNet dataset after 500k iteration. (Left): w/o averaging, (Right): with EMA	81
4.20	ImageNet FID and IS score during training. Setting: original GAN objective/ conventional architecture/ $n_{dis} = 1$	82
4.21	Image in-painting with Yang et al. [3] on CelebA 128×128 (Top): Incomplete images, (Middle): In-painted images without EMA, (Bottom): In-painted images with EMA. Averaged images tends to be smoother, more realistic and has less artifact.	84
5.1	Analysis of overfitting on SVHN. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.	95
5.2	Analysis of overfitting on CIFAR-10. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.	96
5.3	Analysis of overfitting on FFHQ. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.	96
5.4	Analysis of mode drop on SVHN. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.	97

5.5	Analysis of mode drop on CIFAR-10. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.	97
5.6	Analysis of mode drop on FFHQ. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.	98
5.7	Quantitative analysis of overfitting (top), mode drop (middle) and convergence (bottom) on SVHN (left), CIFAR10 (middle) and FFHQ (right) with pixel-wise metric for various mini-batch sizes.	99
5.8	Illustration of scaled linear interpolation in code(z) space of the generator. Upper row corresponds to SVHN (left), CIFAR10 (middle) and FFHQ (right) with mini-batch size of 3200, while the lower row is the same except the mini-batch size of 50.	100
5.9	Overfitting analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on SVHN. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.	101
5.10	Overfitting analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on CIFAR-10. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.	101
5.11	Overfitting analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on FFHQ. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.	102
5.12	Mode drop analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on SVHN. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.	102
5.13	Mode drop analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on CIFAR-10. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.	102
5.14	Mode drop analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on FFHQ. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.	103

5.15	Quantitative analysis of overfitting (top), mode drop (middle) and convergence (bottom) on CIFAR-10 with pixel-wise metric for mini-batch of $3.2k$ and two different setting for channel size (width).	105
6.1	A simplified comparison of GAN models including the proposed Autoregressive GAN.	109
6.2	Proposed Autoregressive GAN (ARGAN) architecture.	111
6.3	Two different types of factorization for our proposed model. The blue tensor block represents activation map after a convolution operation with size $H \times W \times C$. (a) Spatial Factorization: $2 \times 2 \times 4$ feature tensor factorized into $1 \times 1 \times 4$ tensors which corresponds to factorization along spatial dimensions, (b) Channel-wise Factorization: $2 \times 2 \times 4$ feature tensor factorized into $2 \times 2 \times 1$ tensors which corresponds to factorization along depth (or channel) dimensions.	114
6.4	Left: A feature tensor factorized in spatial fashion. Right (top): An autoregressive model on factorized features. Right (bottom): Patch-GAN: a single discriminator is used for each spatial tensor to ensure local realism.	115
6.5	Generation for CIFAR-10 dataset. (Left): S-ARGAN. (Middle): C-ARGAN. (Right): PARGAN	120
6.6	Generation for STL-10 dataset with 48x48 resolution. (Left): S-ARGAN. (Middle): C-ARGAN. (Right): PARGAN	121
6.7	Generation for STL-10 dataset with 96x96 resolution and PARGAN modeling.	121
6.8	Generation for CelebA dataset with 64x64 resolution and S-ARGAN modeling.	121

List of Tables

3.1	Generator architecture for 28x28 resolution (MNIST and Fashion-MNIST)	39
3.2	Discriminator architecture for 28x28 resolution (MNIST and Fashion-MNIST)	39
3.3	Generator architecture for 32x32 resolution (CIFAR-10)	39
3.4	Discriminator architecture for 32x32 resolution (CIFAR-10)	40
3.5	ResNet generator architecture for 64x64 resolution (CelebA)	40
3.6	Discriminator architecture for 64x64 resolution (CelebA)	41
3.7	Configurations of artificial datasets. Refer to Table 3.8 to check correspondence of labels for Fashion-MNIST and CIFAR-10.	45
3.8	Correspondence of labels for Fashion-MNIST and CIFAR-10.	45
3.9	Quantitative results on 3 datasets and 3 cases. Accuracy of each region and their average is reported. IG stands for Independent Generators and Avg is average value over the regions. The regions can be tracked from Fig. 3.2. n/a is placed into the regions where it does not exist in the type of Venn diagram. Average score for random baseline is 0.5, 0.25 and 0.61 for settings A, B and C respectively. We have bolded the best average results per dataset - case setting.	49
3.10	Comparison of VennGAN and baseline conditional generator in terms of FID measured for each distribution. FID_i indicates FID scores between i-th generator distribution, p_{g_i} and i-th data distribution, p_{data_i} . Lower scores are better.	54
4.1	Settings for CIFAR-10, STL-10, CelebA, ImageNet	65
4.2	Conventional Generator Architecture for 32x32 resolution	66
4.3	Conventional Discriminator Architecture for 32x32 resolution	67
4.4	Settings for Mixture of Gaussians	69
4.5	Settings for Mixture of Gaussians for <i>OMD</i>	69
4.6	Settings for Mixture of Gaussians for <i>CO</i>	70
4.7	Settings for Mixture of Gaussians for <i>Zero-GP</i>	70
4.8	Wasserstein-1 Distance for non-averaged generator, EMA, MA, OMD, CO and Zero-GP	70
4.9	Inception and FID scores on CIFAR-10, STL-10 and ImageNet. (100k), (250k) etc. refer to number of iterations of the generator. Experiments are repeated 3 times.	72

4.10	IS and FID on CIFAR-10 (the same setting of the first row of Table 4.9) with various α values for PMA.	83
4.11	IS and FID on CIFAR-10 (the same setting of the first row of Table 4.9) with various β values for EMA.	83
5.1	The generator architecture	91
5.2	The discriminator architecture	92
5.3	The table of overfitting, mode drop and convergence with varying batch size on SVHN, CIFAR10 and FFHQ. All the results are evaluated with pixel-wise metric. The percentages (10% 5% 1%) shows the worst case scenarios as explained in Section 5.3.1.	98
5.4	The table of overfitting, mode drop and convergence on SVHN, CIFAR10 and FFHQ w/o and w/- noisy input. All the results are evaluated with pixel-wise metric. The percentages (10% 5% 1%) shows the worst case scenarios as explained in Section 5.3.1.	103
6.1	The generator architecture for CIFAR-10	116
6.2	The discriminator (E , R , and d) architecture for CIFAR-10	117
6.3	The generator architecture for STL-10	117
6.4	The discriminator (E , R , and d) architecture for STL-10	117
6.5	The generator architecture for CelebA	118
6.6	The discriminator (E , R , and d) architecture for CelebA	118
6.7	Inception scores for our proposed models and state-of-the-art baselines on CIFAR-10 and STL-10.	122

Symbols and Acronyms

Symbols

$\mathcal{R}^n$	the n -dimensional Euclidean space
$\ \cdot\ $	the 2-norm of a vector or matrix in Euclidean space
$\mathbf{x}$	the data samples
$\mathbf{z}$	the latent code (samples from the prior)
$D(\cdot)$	the discriminator
$G(\cdot)$	the generator
$C(\cdot)$	the classifier
$E(\cdot)$	the encoder
$R(\cdot)$	the autoregressive model
$\phi^{(t)}$	the discriminator parameters at t -th time point
$\theta^{(t)}$	the generator parameters at t -th time point
$\phi_c^{(t)}$	the classifier parameters at t -th time point
$\mathcal{L}(D, G)$	the value function (loss) of GANs
$p_{data}(\mathbf{x})$	the true data distribution
$p_g(\mathbf{x})$	the generator's distribution
$p_{g_i}(\mathbf{x})$	the i -th generator's distribution
$p_z(\mathbf{z})$	the prior distribution
S_i	i -th set
r_i	i -th region
$\mathcal{N}(\mathbf{0}, \mathbf{I})$	the normal distribution with zero mean and isotropic unit variance
$\mathcal{O}$	the complexity
$\mathbf{O}$	the mixture matrix
K	the number of mixtures
$\cup$	set union operator
$\cap$	set intersection operator

$\setminus$	set minus operator
$\subset$	subset of
$\odot$	the Hadamard (component-wise) product
$\langle \cdot, \cdot \rangle$	the inner product of two vectors
$\circ$	the composition of functions
$\mathbf{I}$	the identity matrix
$\mathbf{0}$	the zero matrix
∇f	the gradient vector
$V(\theta, \phi)$	the gradient vector field
$J_V(\theta, \phi)$	the Jacobian of the gradient vector field

Acronyms

GANs	Generative Adversarial Networks
WGAN	Wasserstein Generative Adversarial Networks
CGAN	Conditional Generative Adversarial Networks
DCGAN	Deep Convolutional Generative Adversarial Networks
AC-GAN	Auxiliary Classifier Generative Adversarial Networks
EBGAN	Energy Based Generative Adversarial Networks
ARGAN	Autoregressive Generative Adversarial Networks
s-ARGAN	Spatial-wise Autoregressive Generative Adversarial Networks
c-ARGAN	Channel-wise Autoregressive Generative Adversarial Networks
PARGAN	Patch-Autoregressive Generative Adversarial Networks
GMM	Gaussian Mixture Model
RNN	Recurrent Neural Networks
LSTM	Long Short-term Memory
1-NN	1-Nearest Neighbour
IS	Inception Score
FID	Fréchet Inception Distance
EMA	Exponential Moving Average
MA	Moving Average
PMA	Parameterized Moving Average
SVHN	The Street View House Numbers
FFHQ	Flickr-Faces-HQ
OMD	Optimistic Adam

CO	Consensus Optimization
GP	Gradient Penalty
Zero-GP	Zero Centered Gradient Penalty
GD	Gradient Descent
SGD	Stochastic Gradient Descent
<i>i.i.d.</i>	independent and identically distributed

Chapter 1

Introduction

In this chapter, we provide a gentle introduction to Generative Adversarial Networks (GANs). Afterwards, we briefly introduce some of the applications of GANs in order to motivate GANs framework and its challenges. Later, we discuss some of the challenges of GANs that we aim to deal with. Then, we briefly discuss how our proposed models analyze and alleviate some of the challenges. Following that, we discuss our contributions in this thesis. Finally, we provide an outline for the rest of the thesis.

1.1 Introduction to GANs

Generative Adversarial Networks (GANs) [4] are recently invented generative models which can produce high dimensional samples from data distribution. In a nutshell, GANs are an adversarial game between a generator (G) and a discriminator (D); the discriminator denies the similarity between generated samples and authentic data samples, while the generator tries to trick the discriminator by generating data-like samples. This game can be represented as a zero-sum objective:

$$\min_{\theta} \max_{\phi} \mathcal{L}(D_{\phi}, G_{\theta}) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D_{\phi}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D_{\phi}(G_{\theta}(\mathbf{z})))] \quad (1.1)$$

where D is the discriminator, G is the generator, ϕ is the discriminator parameters, θ is the generator parameters, $p_{data}(\mathbf{x})$ is the data distribution, and $p_z(\mathbf{z})$ is the latent distribution. It utilizes a discriminator to assess a pseudo-divergence between the true data distribution, $p_{data}(\mathbf{x})$, and the generator's distribution, $p_g(\mathbf{x})$ ¹. The discriminator maximizes the divergence, while the generator minimizes it. In other words, the discriminator aims to find differences between the data distribution and the generator's distribution, while the generator aims to trick the discriminator. Both players update their own parameters, in an iterative manner, to satisfy their own objective. In this way, the generator learns to mimic the data distribution implicitly. Goodfellow et al. [4] shows that, under certain assumptions, for a fixed optimal D ², minimizing Eq. 1.1 for G in the function space would lead to $p_g(\mathbf{x}) = p_{data}(\mathbf{x})$.

Eq. 1.1 can be simplified by considering it from each player's perspective. For a fixed G_{θ}^* ³, Eq. 1.1 turns into:

$$\max_{\phi} \mathcal{L}(D_{\phi}, G_{\theta}^*) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D_{\phi}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D_{\phi}(G_{\theta}^*(\mathbf{z})))] \quad (1.2)$$

This objective simply corresponds to minimizing a binary cross entropy between two classes: real and fake (or generated). Hence, the discriminator is simply a binary classifier.

On the other end, for a fixed D_{ϕ}^* , Eq. 1.1 turns into:

---

¹This distribution is not explicitly defined in Eq. 1.1, but it is the distribution of $G_{\theta}(\mathbf{z})$.

²For clarity we omit parameters of the discriminator and the generator which are ϕ and θ respectively.

³that is fixing its parameters

$$\min_{\theta} \mathcal{L}(D_{\phi}^*, G_{\theta}) = \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D_{\phi}^*(G_{\theta}(\mathbf{z})))] \quad (1.3)$$

Notice that the first term in Eq. 1.1 disappears from the above objective as it is constant w.r.t. G . Minimizing the above objective corresponds to increasing the output of D closer to 1 which is the ideal output for the real samples. Intuitively, minimizing this objective brings generated samples closer to the real ones.

Eq. 1.1 is a nested objective where D optimizes inner loop, while G optimizes the outer one. Ideally, inner loop should be optimized till convergence before optimizing the outer one, however this procedure is costly and does not work well in practice. Instead, each player is updated for a few times before the other player updates itself [4]. In other words, for each optimization of D w.r.t. Eq. 1.2, G is optimized for k times w.r.t. Eq. 1.3. With this procedure, after some time, G manages to produce data-like samples.

1.1.1 Training Procedure and Visualizations

In GANs framework, the generator is parameterized by a neural network (see Fig. 1.1). It takes a simple code $\mathbf{z}$ as input and pass it through the function G . Output of G , denoted as $G(\mathbf{z})$, which lies in the same space of $\mathbf{x}$, aims to produce data-like samples. As there is no predefined set of $(\mathbf{x}, \mathbf{z})$ pairs to train the generator with maximum likelihood, GANs utilize another approach. It uses an auxiliary discriminator, D , to assess difference between generated samples and training samples (see Fig. 1.2). D is simply a classifier which classifies real images into the category of 1 and fake ones into 0 which simply corresponds to maximizing Eq. 1.1. By doing so, D implicitly learns what makes real data different than the generated ones.

After the discriminator maximizes Eq. 1.1, it can be used to give feedback to G to make generated samples closer to the real ones. This is simply done by changing the parameters of G to make D 's output closer to 1 which is the real class (see

FIGURE 1.1: The Generator of GANs framework. It is parameterized as a multi-layer neural network.

FIGURE 1.2: The Discriminator of GANs framework. It is parameterized as a multi-layer neural network.

Fig. 1.3). In other words, G minimizes Eq. 1.1. Notice that D does not impose restriction on which z should corresponds to which x , but instead, whether x are data-like. This gives a degree of freedom to G to make the mapping.

FIGURE 1.3: Improving the generator. The parameters of the generator updated to increase the output of the discriminator for the generated samples.

With this procedure, GANs achieve a good approximation to the target distribution and generates data-like samples. During the inference stage, a new $\mathbf{z}$ is sampled from the latent distribution and passed through the generator to create a novel data-like sample. As we can sample from the prior distribution freely, the generator can produce potentially infinite amount of samples. In general, the latent code, $\mathbf{z}$, is a low dimensional, relative to the data, and simple distribution such as isotropic Gaussian or uniform distribution. Latent code is chosen as low dimensional to

enforce abstractness on this space, similar to latent representations of a classifier. Also, having analytically defined distribution makes it easy to sample from it both during the training and inference (test stage).

1.2 Applications of GANs

GANs have gained popularity recently as they can be applied to various applications. Due to enormous interest, they have been improved substantially over the past few years [1, 5–9]. The main aim of this section to show applications of GANs in order to motivate the GANs objective and solving its challenges. We do not mean to cover all the applications but to show small but prominent use cases.

Image Generation: This is the most basic setting of GANs. A well trained generator can produce high quality, high resolution images as in Karras et al. [8, 10]. Starting from the prominent work of Goodfellow et al. [4], quality of image generation is improved substantially over years [5, 7, 8, 10]. Generated images can be used in a downstream task such as additional data to unsupervised learning etc.

Conditional Image Generation: Basic image generation mentioned above does not define what type of images to generate. It might be desirable to generate images with specific attributes. Conditional image generation solves this problem by introducing a condition into the model which defines the class, group or attributes of the image. This idea is mentioned by Goodfellow et al. [4], and actualized by Mirza and Osindero [11]. Following works improved the idea further [1, 12, 13]. Fig. 1.4 illustrates BigGAN [1] as an example of conditional image generation with GANs. Indeed, conditional GANs (CGANs) are the most used version of GANs and below applications (*Text-to-Image Transformation*, *Image-to-Image Transformation*, and *Super Resolution*) are some instances of it.

Text-to-Image Transformation: This is an interesting application of GANs, where the generator takes a text description and converts it into an image, which is consistent with the description [14–18].

FIGURE 1.4: BigGAN as an application of GANs. It can generate realistic images belong to specific categories. The figure is taken from Brock et al. [1]

Image-to-Image Transformation: Perhaps, this application (also called as domain transfer) is the most popular use case of GANs for computer vision community. It takes an image as input and transforms it into another image with similar semantics but different low level statistics such as style, texture [2, 19–26]. Fig. 1.5 illustrates application of CycleGAN [2] as an example of image-to-image transformation with GANs.

FIGURE 1.5: CycleGAN as an application of GANs. It can map images from one domain to another with similar semantics and without paired samples. (left) Monet paintings to landscape photos, (center) zebra to horse, (right) summer to winter photos, (bottom) photos to different painting styles. The figure is taken from Zhu et al. [2]

Super Resolution: This application, indeed, a type of image-to-image transformation where low resolution input image in mapped into a higher resolution one [27–30]. As the problem is ill-defined, there can be various mapping, the details should be hallucinated, which makes it a good application for GANs.

Semi-supervised Learning: In most cases of machine learning, unlabeled data is abundant while the labeled data is scarce due to the costly process of labeling. The promise of semi-supervised learning is to reduce dependence on labeled data by utilizing unlabeled data during the training. Salimans et al. [31] introduce GANs into semi-supervised learning, while the following works improve this technique by including regularizers etc. [32–34]. The main intuition is sharing the large chunk of the discriminator of GANs with a classifier that is used for supervised learning. This leads to reduced overfitting over the classifier with little labeled data.

Anomaly Detection: This application detects abnormal samples from large number of normal samples. As it can be framed as learning a density model on the normal samples and detecting low density (abnormal) samples, it is a natural application for GANs. Schlegl et al. [35] achieve this by using usual GANs architecture, while Zenati et al. [36] utilize a BiGAN [37, 38] to make the inference faster. For details, Mattia et al. [39] provide a survey for anomaly detection with GANs.

Some Other Applications: Other than the applications above, GANs have been used for person re-identification [40], hashing [41], and retrieval [42] etc.

1.3 Challenges

Despite the success of GANs, there are various problems with them such as improving high dimensional sample generation, non-convergence of the objective function, overfitting/ generalization, mode drop (not modeling certain parts of the distribution), and lack of efficient multi-distribution modeling. Below we briefly discuss the problems. A detailed explanation and literature review can be found in Chapter 2.

1.3.1 Instability/ non-Convergence

This is one of the notorious behaviour of GANs; instead of converging to a fixed point in space, GANs tend to avoid convergence [9, 43]. There have been various

approaches to mitigate instability/ non-convergence problem with varying degree of success [31, 44–48]. In Chapter 4, we theoretically and empirically analyse the affect of parameter averaging on converge/stability. Also in Chapter 5, we empirically relate this behaviour with the stochasticity in the training.

1.3.2 Overfitting/Generalization

One of the interesting questions about GANs is that whether they overfit and if not why? Various papers claim that GANs do not overfit by searching the nearest-neighbors of generated images in training dataset [1, 4, 8]. Different from these methods, Webster et al. [49], Metz et al. [50] analyze overfitting in GANs and other generative models through searching in the code space of the generator. While the latter approaches achieve closer generated samples to the ones in the training set, there are significant differences between the images, especially the details. In Chapter 5, we empirically show that, under specific conditions, GANs can overfit to the empirical distribution.

1.3.3 Mode Drop/Collapse

This is a notorious behaviour of GANs; instead of covering the entire support of the data distribution, GANs tend to cover only sub-part of it. There are various interpretations of why this phenomenon happens. Some of the interpretations are the underlying divergence of GAN objective [51, 52], low capacity of discriminator [51], and lack of multimodality in prior distribution [53]. In Chapter 5 we link the mode drop to stochasticity in the training.

1.3.4 Multi Distribution Modeling

GANs are designed to learn a single distribution, though multiple distributions can be modeled by treating them separately. However, this naive implementation does

not consider relationships between the distributions. An interesting question is how we can model multiple distributions efficiently and discover their common and unique aspects? This question has not been answered in the current literature. We take this problem into consideration and propose a successful approach in Chapter 3.

1.3.5 Improving Data Quality

Starting from the prominent work of Goodfellow et al. [4], GANs have been improved considerably over short period of time [5, 6, 31, 54, 55]. However, GANs' performance does not scale well to large datasets with high sample variation such as ImageNet (by the time we write our work.). Generated images tend to be sharp but incoherent and with considerable artifacts. In Chapter 6, by pinpointing a problem with GANs, we have proposed an autoregressive modeling in the latent space of the discriminator in order to improve image generation quality. Our hypothesis is that by modeling the feature co-occurrences explicitly, the generator can capture the data distribution better. Also, our work in Chapter 4 significantly boosts image quality where images are smoother, more realistic and include less artifacts.

1.4 Proposed Methods

In this thesis, we present four works where each aims to analyse and mitigate important issues of GANs. Below, we briefly present the problems, motivations, approaches and the results of these works.

In our first work (Chapter 3), we address the question of how to model common and unique aspects of multiple distributions. Finding these aspects can be used for a subsequent task e.g. what can be found in distribution A but not in distribution B?

We propose a new GANs design called VennGAN which models multiple distributions effectively and discovers their commonalities and particularities. Each data distribution is modeled with a mixture of multiple generator distributions. The generators are partially shared between the modeling of different data distributions: shared ones capture the commonality of the distributions, while non-shared ones capture unique aspects of them. We thoroughly evaluate our proposed VennGAN and show its effectiveness on various datasets (MNIST, Fashion-MNIST, CIFAR-10, CelebA) and settings with compelling results. Additionally, we illustrate how this method can be used to incorporate external (prior) knowledge to distribution modeling.

In the second work (Chapter 4), we examine two different techniques for parameter averaging in GANs training in order to mitigate non-convergence of the objective function. Moving Average (MA) computes the time-average of the network parameters, whereas Exponential Moving Average (EMA) computes an exponentially discounted sum of the network parameters. Whilst MA is known to lead to convergence in bilinear settings, to the best of our knowledge, we provide the first theoretical analysis in support of EMA. We show that EMA shrinks the cycles around the equilibrium for simple bilinear games and also enhances the stability of general GANs training. We establish experimentally that both techniques are strikingly effective in the general (non-convex-concave) GANs setting as well. Both improve commonly used Inception Score and Frechet Inception Distance on different architectures and for different GANs objectives. We provide comprehensive experimental results across a range of datasets, mixture of Gaussians, CIFAR-10, STL-10, CelebA and ImageNet, to demonstrate its effectiveness. Also, we have applied EMA into image in-painting problem which results in smoother, more realistic images with less artifacts.

In the third work (Chapter 5), we study important questions in GANs literature such as overfitting, mode drop and non-convergence from empirical perspective on high dimensional distributions. Overfitting has important implications for privacy concerns as the generator can faithfully memorize the training samples which might

be a private data. While mitigating mode drop and non-convergence leads to better distribution modeling. We have shown that, given enough capacity, when stochasticity is removed from the training (optimization), GANs can overfit, show almost no mode drop and reduce non-convergence rates significantly. These results shed a light on important characteristics of GANs training and dismiss certain intuition in the current literature such as GANs do not memorize the training set, mode drop and non-convergence are mainly due to theoretical characteristic of GANs objective rather than an optimization issue.

In the fourth work (Chapter 6), we aim to improve the quality of image generation with GANs by refining the discriminator architecture and its objective. We have proposed a novel GAN architecture & objective, called ARGAN, and successfully trained it on CIFAR-10, STL-10, and CelebA datasets. Different from original GANs setting, it models the latent distribution of real samples to learn feature co-occurrences implicitly. We propose two types of ARGAN model, namely Spatial-wise ARGAN and Channel-wise ARGAN. Also, we have combined our proposed Spatial-wise ARGAN with Patch-GAN [19] to further improve the stability of training and the quality of image generation. Our results show that the proposed model outperforms DCGAN [5], EBGAN [56], and WGAN [55] on the same architecture.

1.5 Thesis Contributions

Our main contributions are as follows:

- (i) We propose a solution to an interesting problem setting where there exist multiple distributions in various configurations (see Figure 3.2). Our method (Chapter 3) can capture commonalities and particularities of various distributions with high success rate. We thoroughly evaluate the proposed method on various datasets,

namely MNIST, Fashion MNIST, CIFAR-10, CelebA, with compelling results. Additionally, we show that our method can be used to incorporate external (prior) knowledge to distribution modeling.

(ii) We show theoretically that although EMA does not converge to equilibrium, even in bilinear games, it helps to stabilize cyclic behavior by shrinking its amplitude (Chapter 4). In non-bilinear settings it preserves the stability of locally stable fixed points. We empirically demonstrate that both averaging techniques consistently improve results for several different datasets, network architectures, and GAN objectives. Also, we have applied EMA into image in-painting problem which results in smoother, more realistic images with less artifacts. Additionally, we compare it with several other methods that try to alleviate the cycling or non-convergence problem and demonstrate its effectiveness.

(iii) We show that GANs can overfit to training set when stochasticity is removed from the training (Chapter 5). When it overfits, by definition, it does not mode drop which goes against the common belief about relation between mode drop/-collapse to GAN objective. Also our results show that significant part of the oscillation/non-convergence in GAN training can be attributed to the stochasticity in the optimization.

(iv) We propose a novel method which refines discriminator architecture and its objective in order to improve data generation quality (Chapter 6). Different from original GANs setting, our method models the latent distribution of real samples to learn feature co-occurrences implicitly. Through efficient factorization techniques, we have reduced our method's computation overhead. Also, we have combined our proposed method with Patch-GAN to further improve the stability of training and the quality of image generation. Our results show that the proposed model outperforms DCGAN, EBGAN, and WGAN on the same architecture.

1.6 Outline of the Thesis

The rest of the chapters are as follows. Chapter 2 reviews prominent and related works on GANs, introduces datasets and evaluation techniques we have used. Chapter 3 introduces our proposed VennGAN which models multiple distributions efficiently and discovers their common and unique aspects. Chapter 4 theoretically and empirically analyses two different techniques for parameter averaging in GANs training in order to mitigate non-convergence of the objective function. Chapter 5 studies important questions in GANs literature such as overfitting, mode drop and non-convergence from empirical perspective on high dimensional distributions. Chapter 6 introduces a novel method that models feature co-occurrences in data samples in order to improve data generation quality. Finally, Chapter 7 concludes the thesis with discussion and the future work.

Chapter 2

Literature Review

This chapter is organized under three sections. In the first section, we provide a literature review on the works relevant to ours. In the second section, we introduce the datasets used in this thesis. In the last section, we describe the evaluation techniques used in this thesis to assess quality of the results.

2.1 Related Works

In this section, we review the prominent and related works in the literature. In each subsection, we first provide relevant works on a specific topic/ challenge, then we compare and relate it with our proposed methods/ analysis.

2.1.1 Multi-generator/discriminator GANs

There have been some attempts to use multiple generators/ discriminators in order to solve various issues with GANs. Arora et al. [51], Hoang et al. [57], Ghosh et al. [58] model a single distribution with multiple generators to capture different modes of the distribution. In order to guide the generators into different modes, they utilize a classifier which separates each generator from one another. Durugkar

et al. [59], Neyshabur et al. [60], Juefei-Xu et al. [61] utilize multiple discriminators to address mode collapse and optimization stability. Similarly, Doan et al. [62] use multiple discriminators with learned importance to ease training of GANs. Tolstikhin et al. [63] use a meta-learning algorithm analogous to *AdaBoost* to improve coverage of modes with multiple generators. In Chapter 3, we use multiple generators/ discriminators. However the aim/ motivation of our work is different which is learning commonalities and particularities between the distributions.

2.1.2 Conditional GANs

This type of GANs uses a condition, alongside of noise, to generate data [11]. The conditions are desired to correlate with generated data. It has been used for Image-to-Image transformation [19, 64, 65], text-to-image [14], super resolution [27]. The way GANs are conditioned is still an active research field. The most common way to include conditions into the generator is to provide it as input [11, 12, 14]. Recently, Miyato and Koyama [13] use conditional BatchNorm [66, 67] to include conditions into the generator through parametrization. In Chapter 3, we use both conditional generator and multiple generators, though our aim is different from all the mentioned works.

2.1.3 Mixture of Distributions with GANs

Some of the earlier works consider multiple generators as mixture of distributions to model a single distribution [51, 57, 58]. Yu and Zhou [68] use mixture of GANs to discover clusters in a distribution which is traditionally modeled with Gaussian Mixture Models. Our approach (Chapter 3) is different from the mixture of distributions with GANs, as they are not aiming to learn our aforementioned objective. Conceptually, our modeling is similar to class conditional mixture of distributions with shared components [69, 70]; though our modeling is with GANs

which makes it possible to model high dimensional data distributions e.g. natural images. Concurrent work of Kaneko et al. [71] is perhaps the most similar work to ours. However, their motivation, method and experiments are different from ours. They are motivated by ambiguous class labels due to noisy labels and propose a model to discover class-distinct and class-mutual parts. Their method utilizes a modified version of AC-GAN [12] and redesigns the input of the generator to achieve the objective. In contrast, our work scales GANs objective into n distributions and models each distribution as a mixture of generator distributions.

2.1.4 Stability of GANs Training

There have been attempts to improve stability and alleviate cycling issues arising in GANs. Salimans et al. [31] use historical averaging of both generator and discriminator parameters as a regularization term in the objective function: each model deviating from its time average is penalized to improve stability. Ge et al. [44] update the discriminator by using a mixture of historical generators. Heusel et al. [45] use two time scale update rules to converge to a local equilibrium. Mescheder et al. [48] identify the presence of eigenvalues of the Jacobian of the gradient vector with zero real part and large imaginary part as a reason for non-convergence. They address this issue by including squared gradients of the players to the objective. Extra-gradient (optimistic) methods help with convergence [46, 47]. Some of the above stated methods such as Salimans et al. [31], Mescheder et al. [48] influence the min-max game by introducing regularization terms into the objective function. Regularizing the adversarial objective can potentially change the optimal solution of the original objective. Our method (Chapter 4) on the other hand does not change the game dynamics, by keeping an average over iterates *outside* of the training loop.

Uniform Average and Exponential Moving Average over model parameters have been used for a long time for maximum likelihood objectives [72–76]. However we would like to emphasize the difference between convex/non-convex optimization

and saddle point optimization. GANs are an instance of the latter approach. There is no formal connection between the two approaches, so we cannot extrapolate our understanding of averaging at maximum likelihood objectives to saddle point objectives.

In case of averaging for min-max objective, Gidel et al. [77] use uniform averaging similar to our work (Chapter 4). However, we show its shortcomings and how exponential decay averaging performs better. While this strategy has been used in Karras et al. [8] recently, their work lacks any insights or a detailed analysis of this approach. We aim to provide both theoretical tools and detailed experimentation to explain its superior performance.

2.1.5 Overfitting /Generalization

An important question in GANs literature is whether they overfit/ memorize the training set? To answer this question two major approaches have been used. The first approach searches the nearest-neighbors of randomly generated images in training set [1, 4, 8]. The second approach searches for overfitting in the code space of the generator [49, 50]. In general, the former approach fails to find strong memorization patterns. Even though the latter approach retrieves closer generated samples to the ones in the training set, there are significant differences between the images, especially the details. Bousquet et al. [78] suggest that memorization may not be happening as the generator does not directly learn from the training samples but from the feedback of the discriminator. These observations suggest that GANs do not overfit the training set¹.

In Chapter 5, we empirically show that when stochasticity is removed from the training, GANs can overfit to the empirical distribution. This is, to the best of our knowledge, first time to show that GANs can memorize the training set with minimal difference. Recently, Chavdarova et al. [79] suggest methods to reduce

---

¹Brock et al. [1] claims that the discriminator overfits by showing the discrepancy between the logits of training set and validation set. In Chapter 5, we are looking to overfitting/memorization from the generator side which is different than the overfitting of the discriminator.

variance in gradient calculations without using larger mini-batches. Our work, in essence, also utilizes reducing stochasticity by means of larger batch size etc. However the aim of our work is different and more generic that is showing how overfitting, mode drop and convergence is linked with stochasticity of the training. A related work of Nagarajan et al. [80] provides a theoretical analysis of GAN memorization. Their memorization is defined as generator distribution matches empirical data distribution over all samples from the prior, while ours aims to match only on limited samples from the prior. Also they do not aim to answer the questions about mode drop and non-convergence which we do.

2.1.6 Mode Drop (or Collapse)

Mode drop is a common problem in GANs. GANs tend to cover only some parts of the data distribution, while leaving certain modes out. Some people link this behaviour to underlying divergence of GAN objective (non-saturating version) which is somewhat related to reverse-KL divergence [51, 52]. Theoretical work of Arora et al. [51] suggests that low capacity of discriminator is related with mode drop. Arora et al. [81] analyse this with birthday paradox and conclude that GANs are prone to leaving modes out. Xiao et al. [53] relate the mode drop to mismatch between multimodality of data distribution and unimodality of prior distribution of the generator. As a fix, they suggest use of Gaussian mixture model as prior, similar to Ben-Yosef and Weinshall [82], instead of Gaussian or uniform distribution. Motivated by disconnected manifolds, Khayatkhoei et al. [83] also blame the multimodality of the data distribution and suggest mixture distribution of multiple generators. Thanh-Tung et al. [84] relate mode drop with catastrophic forgetting as a result of continuously changing generative distribution. Contrast to the previous works, in Chapter 5, we empirically show that, under specific conditions, GANs show minimal mode drop. Our observations show that mode drop is not necessarily due to the objective but rather the optimization more specifically the stochasticity in the training.

2.1.7 Convergence

Convergence of GAN objective and its other variants has been analyzed by Mescheder et al. [9], Thanh-Tung et al. [43] recently. It shows that un-regularized GAN objective does not converge to a saddle point but cycles around it. However in practice even with these regularization schemes there is an empirical non-convergence due to ignored characteristics of the training. For example, toy example in Mescheder et al. [9] (Dirac-GAN) does not consider multiple samples or transition from Gradient Descent (GD) to Stochastic Gradient Descent (SGD). Their analysis based on learning a dirac, or degenerate Gaussian, distribution conceals important issues of GAN such as mode drop. In usual GAN setting, the generated images reach some level of quality and keep changing even though they do not get better or it gets worse e.g. mode collapse [85]. In Chapter 5, we empirically analyze the convergence of generated samples which is directly related with convergence of the value function of GANs. Our convergence analysis shed a light on non-convergence of the samples by showing its relation to stochasticity in optimization. This analysis is complementary to the theoretical analysis of GAN objective in Chapter 4. Recently, Chavdarova et al. [79] suggest methods to reduce variance in gradient calculations without using larger mini-batches. Our work in Chapter 5, in essence, also utilizes reducing stochasticity by means of larger batch size etc. However the aim of our work is different and more generic that is showing how overfitting, mode drop and convergence is linked with stochasticity of the training.

2.1.8 Improving Data Quality

Starting from the prominent work of Goodfellow et al. [4], GANs have been improved considerably. Denton et al. [54] improve the image quality by generating low resolution images first and conditioning the high resolution generation on top of the low resolution images. This divide-and-conquer mechanism improves the sample quality and stabilizes the training. Radford et al. [5] come up with architectural changes and tricks to scale GANs training into realistic face, indoor,

and outdoor scenes with significantly lower artifacts and without using stage-wise training. Salimans et al. [31] introduce tips and tricks to improve the generation quality. Arjovsky et al. [55] change underlying divergence of GANs objective from Jensen–Shannon divergence to Wasserstein-1 divergence in order to improve informativeness in the gradients even if the support of the distributions are not matching. Its improved version Gulrajani et al. [6] incorporates a gradient term into the objective to satisfy lipschitzness criteria of WGAN which results in improved scores for the generated samples. Though, Fedus et al. [86] report that the same gradient penalty also improves vanilla GANs setting. In Chapter 6, by pinpointing a problem with GANs, we have proposed an autoregressive modeling in the latent space of the discriminator in order to improve image generation quality. Our hypothesis is that by modeling the feature co-occurrences explicitly, the generator can capture the data distribution better.

To the best of our knowledge, the closest work to our method (Chapter 6) is *Denoising Feature Matching* [87]. Even though they use density estimation in latent space, there are major differences which make learning dynamic of our model totally different from theirs. (i) Their method is complementary to GAN objective while our method can be learned standalone. (ii) More importantly their discriminator are trained as in original GAN objective which means that features learned from the data distribution are based on classifier’s feedback not on density model’s. This crucial difference makes both works different from one another. (iii) In our model feature co-occurrences is modeled explicitly. (iv) Motivation for both works are totally different.

2.2 Related Datasets

In order to assess the quality of our methods, we have used various image datasets in this thesis. The datasets are MNIST [88], Fashion-MNIST [89], SVHN [90], CIFAR-10 [91], STL-10 [92], CelebA [93], FFHQ [10], and ImageNet [94]. Below we include a brief description of these datasets.

MNIST [88] is a handwritten digit dataset with 60k training and 10k test samples (see Fig. 2.1). It is gray scale with 28×28 resolution and contains 10 classes. In our experiments, we have used the 60k training samples only.

FIGURE 2.1: Random samples from MNIST dataset.

Fashion-MNIST [89] is a dataset of 10 fashion items: t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot (see Fig. 2.2). It is released as replacement for MNIST to make the task more challenging. The characteristics of it are similar to MNIST that is 60k training samples, 10k test samples, gray scale, 28×28 resolution, and contains 10 classes. In our experiments, we have used the 60k training samples only.

FIGURE 2.2: Random samples from Fashion-MNIST dataset.

The Street View House Numbers (SVHN) [90] is a real-world image dataset collected from the streets (see Fig. 2.3). In terms of digit content it is similar to MNIST, though with harder settings: color image, 32×32 resolution, and in wild. In our experiments, we have used cropped format as described in <http://ufldl.stanford.edu/housenumbers/>.

CIFAR-10 [91] is a real-world image dataset of 10 objects: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck (see Fig. 2.4). It contains 60k

FIGURE 2.3: Random samples from SVHN dataset.

color images with 32×32 resolution and $6k$ images per class. In our experiments, we have only used the training section of this dataset which corresponds to $50k$ images.

FIGURE 2.4: Random samples from CIFAR-10 dataset.

STL-10 [92], similar to CIFAR-10, is a real-world image dataset of 10 objects: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck (see Fig. 2.5). The main difference from CIFAR-10 is higher resolution (96×96), fewer labeled samples per class, and more challenging background. In our experiments, we have discarded the labels and combined the training images with $100k$ unlabeled images to use for unconditional image generation task.

FIGURE 2.5: Random samples from STL-10 dataset.

CelebA [93] is a widely used celebrity face dataset for image generation tasks (see Fig. 2.6). It contains more than 200k celebrity images with 40 attributes. In our experiments, we have used officially cropped and aligned version of the dataset.

FIGURE 2.6: Random samples from CelebA dataset.

Flickr-Faces-HQ Dataset (FFHQ) [10], similar to CelebA, is a face dataset used for image generation (see Fig. 2.7). It contains 70k high quality images with 1024×1024 resolution. The images are highly diverse with various age groups, ethnicity and image background etc.

FIGURE 2.7: Random samples from FFHQ dataset.

ImageNet [94] is a large scale natural image dataset with more than millions of images over thousand classes (see Fig. 2.8). On average it contains thousand examples per class. The complexity of the dataset makes it very challenging for image generation task. In our experiments, we have used down-sampled (32×32) version of ImageNet as in Van Den Oord et al. [95] which can be found in this link: <http://www.image-net.org/small/download.php>. This dataset contains only images with no labels which suits well for unconditional image generation.

FIGURE 2.8: Random samples from ImageNet dataset.

2.3 Evaluation Metrics

Evaluation of generative models is still an active research topic. The main aim of the evaluation metric is to measure similarity between the generated distribution and the real one. Ideally, as generated distribution gets closer to the real one, the metric should monotonically decrease. A detailed empirical analysis of evaluation metrics for generative models can be found in Borji [96] and Shmelkov et al. [97].

2.3.1 Inception Score and Fréchet Inception Distance

Initially, Goodfellow et al. [4] utilize a Gaussian Parzen window based log likelihood. However, Theis et al. [98] show that this is not a good technique when data is high dimensional. Two most commonly used evaluation metrics in the literature are Inception Score (IS) [31] and Fréchet Inception Distance (FID) [45]. IS is defined as:

$$\exp(\mathbb{E}_{\mathbf{x}} \mathcal{D}(p(y|\mathbf{x}) || p(y))) \quad (2.1)$$

where $p(y|\mathbf{x})$ is conditional label distribution attained by using a classifier, $p(y)$ is marginal distribution of the labels, $\mathcal{D}$ is Kullback–Leibler divergence, and $\exp$ is exponentiation term. $p(y|\mathbf{x})$ is used to assess how good the generated samples are classified, while $p(y)$ is used to assess how diverse they are. If generated images are not similar to natural images or include artifacts etc., then it yields lower scores of

$p(y|\mathbf{x})$. This score is also sensitive to blurriness of images [12]. A good generative model should result in low entropy in $p(y|\mathbf{x})$ and high entropy in $p(y)$ which leads to higher IS values. Inception network from Szegedy et al. [99] is used as the classifier.

Even though, IS shows high correlation with human judgment [31], there are several drawbacks. IS does not take inter-class diversity into account [100]. Also, it does not use statistics of the real data samples and compare it to the statistics of generated samples [45]. In order to alleviate this issues, Heusel et al. [45] suggest Fréchet Inception Distance (FID). FID compares first two moments (mean and covariance) of real data distribution and generator distribution:

$$d^2((\mathbf{m}_g, \mathbf{C}_g), (\mathbf{m}_r, \mathbf{C}_r)) = \|\mathbf{m}_g - \mathbf{m}_r\|_2^2 + \text{Tr}(\mathbf{C}_g + \mathbf{C}_r - 2(\mathbf{C}_g \mathbf{C}_r)^{1/2}) \quad (2.2)$$

where $\mathbf{m}_g$, $\mathbf{m}_r$, $\mathbf{C}_g$, $\mathbf{C}_r$, and Tr are mean of generated samples, mean of real samples, covariance of generated samples, covariance of real samples and *Trace* operator. To deal with high dimensional samples, mean and covariance are calculated at the latent space of a feature extractor. Similar to IS, FID uses Inception network [99] as its feature extractor. Unlike IS, lower FID scores are better.

In this thesis, we use both IS and FID for evaluation. For Inception score evaluation, we follow the guideline from Salimans et al. [31]. Inception score is evaluated on 50k generated samples, while FID score is evaluated on 10k generated images. For FID scores, statistics of data are calculated at the same scale of generation e.g. 32x32 data statistics for 32x32 generation.

2.3.2 Classification Performance

Another metric to evaluate quality of the samples is classification performance. Similar to the first term in IS, classification performance utilizes a classifier to

assess at what degree the generated samples can be classified that is considered a sign of quality [5, 19]. Note that this technique does not measure the diversity. In Chapter 3, we also utilize this technique to measure whether conditionally generated samples belongs to the correct group.

2.3.3 Evaluation of Overfitting

Other than evaluation of generated samples for quality and diversity, some works evaluate traces of overfitting to show that the model does not memorize the training samples. To achieve it, Brock et al. [1], Goodfellow et al. [4] use nearest neighbor analysis on pixel space. As memorization may not be detected with nearest neighbor analysis on pixel space, Brock et al. [1], Karras et al. [8] search in the latent space of a pre-trained network such as Inception network [99]. In Chapter 5, we also use nearest neighbor on both pixel space and latent space to analyze overfitting and mode drop.

Chapter 3

VennGAN: Discovering Commonalities and Particularities of Multiple Distributions

3.1 Introduction

Generative Adversarial Networks (GANs) [4] learn a function that can be used to draw samples efficiently from an approximated probability distribution. Informally, it achieves this by pitting a discriminator and a generator against each other as an adversarial game. Due to enormous interest, GANs have been improved substantially over the past few years [5–9].

GANs are designed to learn a single distribution, though multiple distributions can be modeled by treating each distribution separately. However, this naive implementation does not consider relationship between the distributions. An interesting question is how we can model multiple distributions efficiently and discover their relations, namely common and unique aspects? When discovering these aspects we are more interested in at an abstract level. For example, in Fig. 3.1 (top) there are two data distributions, S_1 and S_2 , with overlapping modes i.e. digits $\{3, 4, 5, 6\}$.

We can explain this situation by utilizing Venn diagrams. Fig. 3.2 depicts some cases of different settings between either 2 or 3 sets, where each set represents a distribution. In d_1 , there are two sets with an intersection. In d_2 , each set has its own unique part and intersections with the other sets, whereas in d_3 , some sets are a *superset* of others. Each case can be useful in different scenarios, e.g. d_3 can be used in a case where a distribution is a subset of another distribution, such as a specific dog breed and its *superset* is many different dog-breeds. As another

example, d_1 can be used to find unique and common aspects of images collected from two different fashion suppliers. A user can generate new clothes that are similar to the supplier S_1 but different from the supplier S_2 by generating the unique aspect ($S_1 \setminus S_2$) of the clothing. This might help the user understand what are unique elements of each supplier are.

FIGURE 3.2: Three different configurations of Venn diagrams with either 2 or 3 sets. We annotate each region with r_i and each diagram with d_i as we refer to them later in this chapter.

In this chapter, we propose VennGAN, which models multiple distributions efficiently and discovers their commonalities and particularities. In a nutshell, each data distribution is modeled with a mixture of K generator distributions. The generators are partially shared between the modeling of different true data distributions; shared ones capture the commonality of the distributions, while non-shared ones capture unique aspects of them. Our contributions are the following:

- Proposing a solution to an interesting problem setting where there exist multiple distributions in various configurations (see Figure 3.2). Our method can capture commonalities and particularities of various distributions with high success rate.
- Thoroughly evaluating the method on various datasets, namely MNIST, Fashion MNIST, CIFAR-10, CelebA, with compelling results.
- Proposing an application of VennGAN as incorporating external (prior) knowledge to distribution modeling.

3.2 Method

GAN is a two-player zero-sum game between a discriminator (D) and a generator (G):

$$\min_G \max_D \mathcal{L}(D, G) \quad (3.1)$$

$$\mathcal{L}(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g(\mathbf{x})} [\log(1 - D(\mathbf{x}))] \quad (3.2)$$

It utilizes a discriminator to assess a pseudo-divergence between the true data distribution, $p_{data}(\mathbf{x})$, and the generator's distribution, $p_g(\mathbf{x})$. The discriminator maximizes the divergence, while the generator minimizes it. In this way, the generator learns to mimic the data distribution implicitly. Goodfellow et al. [4] shows that, under certain assumptions, for a fixed optimal D , minimizing Eq. 3.2 for G would lead to $p_g(\mathbf{x}) = p_{data}(\mathbf{x})$.

The loss function, Eq. 3.1, can be scaled to n distributions as follows:

$$\min_{G_1, \dots, G_n} \max_{D_1, \dots, D_n} \mathcal{L}(D_1, D_2, \dots, D_n, G_1, G_2, \dots, G_n) \quad (3.3)$$

$$\begin{aligned} \mathcal{L}(D_1, D_2, \dots, D_n, G_1, G_2, \dots, G_n) = \\ \frac{1}{n} \sum_{i=1}^n \mathbb{E}_{\mathbf{x} \sim p_{data_i}(\mathbf{x})} [\log D_i(\mathbf{x})] + \\ \frac{1}{n} \sum_{i=1}^n \mathbb{E}_{\mathbf{x} \sim p_{g_i}(\mathbf{x})} [\log(1 - D_i(\mathbf{x}))] \end{aligned} \quad (3.4)$$

where $p_{data_i}(\mathbf{x})$ is i -th true data distribution and $p_{g_i}(\mathbf{x})$ is i -th generator's distribution. Note that D_i and G_j ¹ in above equation interact with one another only when $i = j$. Also, D_i is independent from D_j and G_i is independent from G_j when $i \neq j$. This makes learning one distribution independent from the others. Due to

---

¹Eq. 3.4 does not explicitly show G_j but p_{g_j} , which is the distribution of the j -th generator, G_j

the independence property, by following the proof from [4], we can show that at equilibrium $\mathbf{p}_{data} = \mathbf{p}_g$.

However this objective does not consider possible overlaps/ interactions between the data distributions. Incorporating this can lead to interesting discoveries, e.g. commonalities and particularities of the distributions. In order to achieve this, we have reformulated the way we construct each generator distribution. p_{g_i} is no longer equal to the distribution of the i -th generator, but a mixture of K generator distributions, denoted by p_{r_i} :

$$\begin{bmatrix} p_{g_1} \\ p_{g_2} \\ \vdots \\ p_{g_n} \end{bmatrix} = \begin{bmatrix} O_{11} & O_{12} & \cdots & O_{1K} \\ O_{21} & O_{22} & \cdots & O_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ O_{n1} & O_{n2} & \cdots & O_{nK} \end{bmatrix} \begin{bmatrix} p_{r_1} \\ p_{r_2} \\ \vdots \\ p_{r_K} \end{bmatrix} \quad (3.5)$$

or in a compact form:

$$\mathbf{p}_g = \mathbf{O}\mathbf{p}_r \quad (3.6)$$

where $\mathbf{O} \in \mathbb{R}^{n \times K}$ (compact form of O_{ij} above) is a mixture matrix whose rows sum up to 1 to make p_{g_i} valid. Note that this reformulation does not change the objective (Eqs. 3.3 and 3.4), but how we model p_{g_i} which makes it easier to analyse its behavior. In this way, each data distribution is modeled as a mixture of distributions. As p_{r_i} are shared for all data distributions, some of them cover common parts and others unique ones depending on the values of the mixture matrix, $\mathbf{O}$. Each p_{r_i} learns only a sub-part of the distributions and combines them at different amounts to make the data distributions.

3.2.1 Conceptual Relation to Venn Diagrams

The proposed method in the previous section can be explained by using Venn diagrams where each set represents a distribution. We deal with a situation where

multiple distributions exist. Each distribution might have a unique part and commonalities with other distributions, e.g. d_2 of Figure 3.2. In another case, one distribution's support might cover the others', e.g. d_3 of Figure 3.2. Our proposed method models each region of a Venn diagrams as a probability distribution $p_{r_i}(\mathbf{x})$. Each set should capture the distribution of its corresponding data distribution, e.g. $p_{S_i} = p_{g_i} = p_{data_i}$. Each set can be represented by union of its regions, e.g. d_2 of Figure 3.2, $S_1 = r_1 \cup r_4 \cup r_6 \cup r_7$. Similarly, each region can be represented with set operations e.g. d_3 of Figure 3.2, $r_1 = S_1 \setminus (S_2 \cup S_3)$. Set configurations can be in different forms, e.g. d_3 of Figure 3.2 is $S_3 \subset S_2 \subset S_1$.

The value of $\mathbf{O}$ specifies which venn diagram the method corresponds to. d_3 type diagram can be represented by:

$$\mathbf{O} = \begin{bmatrix} \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.7)$$

Similarly d_2 type diagram can be represented by:

$$\mathbf{O} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \end{bmatrix} \quad (3.8)$$

In both cases we assume that each region contributes equally. From a Venn diagram perspective, p_{g_i} can be represented by:

$$p_{g_i}(\mathbf{x}) = p_{S_i}(\mathbf{x}) = \frac{1}{|S_i|} \sum_{r_j \in S_i} p_{r_j} \quad (3.9)$$

where $|S_i|$ is number of regions in set S_i . This representation is equivalent to Eq. 3.5.

3.2.2 VennGAN Modeling

After explaining the concept of the VennGAN, in this section, we delve into the design details. We, first, explain how generator and discriminator should be modeled. Then, we provide some intuition on how our modeling discovers regions in the Venn diagrams. Finally, we discuss how imposing additional constraints on the objective improves the modeling.

3.2.2.1 Generator Side

We can use two approaches to model the generators $(G_1, G_2, \dots, G_K)$. The first is the use of K independent generators for each region. Each generator is modeled by $r_i = G_i(z; \theta_i)$, where G is a generative network, z is input noise and θ_i are the parameters of the i -th network. The second approach is a single generator with K conditions. Each region is modeled with a function $r_i = G(z, c = i; \theta)$, where c is a condition whose i -th index used to generate region r_i and θ are the network parameters. The former approach can be expensive when there are many regions to model. However, it has its own advantages as we will show in Section 3.3. Conditional generator is more efficient as the number of regions grows exponentially with distributions, e.g. n distributions contain up to $2^n - 1$ regions. Also, sharing weights with other generators regularizes the model and makes training easier. Besides, using this type of generator has certain effects on modeling, namely different conditions with the same noise produce semantically related samples, as detailed in the CelebA experiments. We use both types and discuss their advantages and disadvantages in more detail in Section 3.3.

3.2.2.2 Discriminator Side

There should be n discriminators for n -distribution game. As we change the generator distribution into a mixture of distributions, each discriminator takes input

from all incoming generators, which has non-zero mixture weight. Fig. 3.3 illustrates how a d_2 type diagrams looks like in terms on connections. Other types can be constructed in a similar way by following the connection pattern from the weight matrix $\mathbf{O}$. In practice, when sampling from the generators to feed into D_i , the number of samples from each generator should be proportional to the i -th row of $\mathbf{O}$. The $+$ sign in the diagram corresponds to union operation over the incoming regions; in practice it is a concatenation at batch dimension of the input tensor.

As each set should represent a true data distribution, $p_{S_i}(x) = p_{data_i}(x)$, union of regions that belongs to S_i should match to i -th data distribution, as in Eq. 3.9. In order to satisfy this, each discriminator D_i compares a specific true data distribution, $p_{data_i}(x)$, with union of regions, $p_{S_i}(x)$, which belongs to the corresponding set, e.g. $S_1 = r_1 \cup r_4 \cup r_6 \cup r_7$.

As certain regions are fed into more than one discriminator, those regions would be forced to represent common parts of the distributions. For example, r_7 of d_2 type diagram will suffer a loss if its modeling does not satisfy the 3-way intersection of the distributions. In other words, it will receive a negative feedback from the discriminator(s) which it could not satisfy. Similar analogies can be made to r_4 , r_5 , r_6 which are 2-way intersections, whereas individual regions like r_1 , r_2 , r_3 are only used by a single discriminator, thereby they are inclined to model the unique part of its corresponding distribution. Sharing the regions between different discriminators which receive different true data distributions is the core dynamic of learning commonalities between the data distributions.

3.2.2.3 Additional constraint

Even though, the above modeling can discover commonalities and particularities of the distributions(see Section 3.3), in practice we observe that there is some amount of leakage between the regions. This leakage might be happening due to not well understood properties of the optimization or lack of constraint in the above objective. For example, in d_2 type diagram, r_1 can model some portion of r_4 , r_6 ,

FIGURE 3.3: An illustration of Venn GAN architecture for d_2 type diagram. For simplicity, we have used a conditional generator, however K independent generators can be used in the same way. Each condition of the generator represents corresponding region in Fig. 3.2; other Venn diagrams can be constructed in a similar way. Each discriminator receives union of its corresponding set's regions. $+$ sign is union over its incoming regions, and the union is fed into a discriminator as fake class. Each discriminator receives different true data distribution and with GAN objective it compares them to the mixture of the regions.

r_7 without suffering a loss from the objective. In order to alleviate this issue, we include an additional objective, which aims to separate regions of the generator from one another:

$$\begin{aligned} \max_{C, G_1, G_2, \dots, G_n} \mathbb{E}_{z \sim p_z(z)} \log C(y_i | \mathbf{r}_i; \phi_c) \\ = \mathbb{E}_{z \sim p_z(z)} \log C(y_i | G_i(z; \theta_i); \phi_c) \end{aligned} \quad (3.10)$$

where y_i is the category for $\mathbf{r}_i$ and C is a classifier which outputs probability distribution over the regions. With this objective, the classifier tries to separate the regions and the generator tries to satisfy the classifier by increasing differences between regions. A similar loss function has been used by Hoang et al. [64] previously for a different task. By combining Eq. 3.3 and Eq. 3.10, we come up with a new

objective:

$$\min_{C, G_1, G_2, \dots, G_n} \max_{D_1, D_2, \dots, D_n} \mathcal{L}(D_1, D_2, \dots, D_n, G_1, G_2, \dots, G_n) - \lambda \mathbb{E}_{z \sim p_z(z)} \log C(y_i | G_i(z)) \quad (3.11)$$

where λ is a hyper-parameter balancing the two terms. With this objective, we satisfy all the criteria to learn commonalities and particularities of the distributions without the leakage issue.

3.2.3 Network Architecture

In the previous section, we have discussed the connection map of our proposed models but not about the network architectures. Discriminator and generator architectures are similar to DCGAN [5] for MNIST, Fashion-MNIST and CIFAR-10, while CelebA uses ResNet type architecture as in Table 3.1, Table 3.2, Table 3.3, Table 3.4, Table 3.5, and Table 3.6. Prior distribution for the generator(s) is a 128-dimensional isotropic Gaussian distribution. *Cond* is conditioning which is linear scaling and addition for each feature channel. It is not used when multiple generators are utilized. *LReLU* is LeakyReLU with $\alpha = 0.2$. *Conv* $n \times n$ is convolution operation with $n \times n$ kernel size. *st* is the stride of the convolution operation, while *pad* is the amount of padding applied. If not mentioned, stride and padding of the convolution is 1. In the discriminator we have used spectral normalization [7].

The classifier architecture is the same as the discriminator except for the last layer, whose output dimensions equal to the number of regions. Exponential Moving Average (EMA) has been used over generator(s) parameters outside of the training loop [8, 85]. Conditioning of G is similar to that of [13, 66, 67] except that there is no normalization term but scaling and addition.

TABLE 3.1: Generator architecture for 28x28 resolution (MNIST and Fashion-MNIST)

Layers	Act.	Output Shape
Latent vector	-	128 x 1 x 1
Conv 4 x 4, pad=3	Cond - LReLU	128 x 4 x 4
Conv 4 x 4, pad=3	Cond - LReLU	128 x 7 x 7
Upsample	-	128 x 14 x 14
Conv 3 x 3, pad=1	Cond - LReLU	64 x 14 x 14
Upsample	-	64 x 28 x 28
Conv 3 x 3, pad=1	Cond - LReLU	32 x 28 x 28
Conv 3 x 3, pad=1	Tanh	1 x 28 x 28

TABLE 3.2: Discriminator architecture for 28x28 resolution (MNIST and Fashion-MNIST)

Layers	Act.	Output Shape
Input image	-	3 x 28 x 28
Conv 4 x 4, st=3	LReLU	64 x 14 x 14
Conv 4 x 4, st=3	LReLU	128 x 7 x 7
Conv 4 x 4, st=3	LReLU	256 x 3 x 3
Conv 3 x 3, st=1, pad=0	Squeeze	1

TABLE 3.3: Generator architecture for 32x32 resolution (CIFAR-10)

Layers	Act.	Output Shape
Latent vector	-	128 x 1 x 1
Conv 4 x 4, pad=3	Cond - LReLU	512 x 4 x 4
Upsample	-	512 x 8 x 8
Conv 3 x 3	Cond - LReLU	256 x 8 x 8
Upsample	-	256 x 16 x 16
Conv 3 x 3	Cond - LReLU	128 x 16 x 16
Upsample	-	128 x 32 x 32
Conv 3 x 3	Cond - LReLU	64 x 32 x 32
Conv 3 x 3	Tanh	3 x 32 x 32

TABLE 3.4: Discriminator architecture for 32x32 resolution (CIFAR-10)

Layers	Act.	Output Shape
Input image	-	3 x 32 x 32
Conv 3 x 3	LReLU	64 x 32 x 32
Conv 3 x 3	LReLU	128 x 32 x 32
Downsample	-	128 x 16 x 16
Conv 3 x 3	LReLU	128 x 16 x 16
Conv 3 x 3	LReLU	256 x 16 x 16
Downsample	-	256 x 8 x 8
Conv 3 x 3	LReLU	256 x 8 x 8
Conv 3 x 3	LReLU	512 x 8 x 8
Downsample	-	512 x 4 x 4
Conv 4 x 4, st=1, pad=0	Squeeze	1

TABLE 3.5: ResNet generator architecture for 64x64 resolution (CelebA)

Layers	Act.	Output Shape
Latent vector	-	128 x 1 x 1
Conv 4 x 4, pad=3	Cond	512 x 4 x 4
ResBlock	-	512 x 4 x 4
Upsample	Cond	512 x 8 x 8
ResBlock	-	512 x 8 x 8
Upsample	Cond	512 x 16 x 16
ResBlock	-	256 x 16 x 16
Upsample	Cond	256 x 32 x 32
ResBlock	-	128 x 32 x 32
Upsample	Cond	128 x 64 x 64
ResBlock	LReLU - Cond	64 x 64 x 64
Conv 3 x 3	Tanh	3 x 64 x 64

3.3 Experiments

In this section, we first discuss objective/ optimization details and hyper parameters used in our experiments. Then, we use an illustrative examples with our VennGAN to spot any subtle behaviour. Following that we present our main experiments on MNIST, Fashion-MNIST [89], CIFAR-10 [91] and CelebA [93] both qualitatively and quantitatively to show effectiveness of our method.

TABLE 3.6: Discriminator architecture for 64x64 resolution (CelebA)

Layers	Act.	Output Shape
Input image	-	3 x 64 x 64
Conv 3 x 3	-	64 x 64 x 64
ResBlock	-	64 x 64 x 64
Downsample	-	64 x 32 x 32
ResBlock	-	128 x 32 x 32
Downsample	-	128 x 16 x 16
ResBlock	-	256 x 16 x 16
Downsample	-	256 x 8 x 8
ResBlock	-	512 x 8 x 8
Downsample	-	512 x 4 x 4
ResBlock	LReLU	512 x 4 x 4
Conv 4 x 4, st=1, pad=0	Squeeze	1

3.3.1 Implementation Details

3.3.1.1 Objective Details

Zero gradient penalty [9] has been applied on true data distributions for each discriminator with weight 1.0 in every case but illustrative examples which use 0.1. Higher values suggested by Mescheder et al. [9] lead to worse results. Such gradient penalties improve the quality of the samples and stability of the training [101]. In our experiments, we found that the gradient penalty improves the quality of sample generation, especially in CelebA.

3.3.1.2 Optimization and Hyperparameters

We have used ADAM [102] optimizer for all networks with a learning rate of 0.0002, $\beta_1 = 0.0$ and $\beta_2 = 0.9$. The optimization of discriminator and generator follows the alternating update rule with a single generator update per discriminator update. We run the experiments till convergence which corresponds to 100k iterations for CelebA, 50k for CIFAR-10, 20k for MNIST and Fashion-MNIST. For each experiments, we have used the largest batch size that can fit into a single GPU. For

each region, we use a batch size of 16, except for the illustrative example which uses 64. The batch size of real data depends on the number of regions fed to each discriminator. For example, union over k regions would correspond to a batch size of $16 \times k$. λ is selected as 0.1 by searching over $[0.1, 10.0]$ with linear intervals whichever maximizes the quantitative score (see Section 3.3.2). For the conditional generator, we use the same noise for different conditions during the training. This makes the semantic comparison of the different regions easier and more meaningful as explained in Section 3.3.4.

3.3.2 Quantification of Results

In this section, we come up with a quantitative metric that measures the rate of correct generation (accuracy) for each region. In order to achieve this, we train a separate classifier (independent from the classifier used during the training of our model) on MNIST, Fashion-MNIST and CIFAR-10 by using their training data split. The classifiers are used to assess if the generated images from each region belong to the correct class(es). The accuracy of the classifier on each region is calculated by using 10k generated samples per region. Then, average accuracy is calculated by averaging scores over all regions. To the best to our knowledge, there is no other method achieving our objective e.g. discovering common and unique parts of the distributions. Hence, we compare average accuracy scores from our model to a random baseline that is the probability of a random sample appearing in the correct region.

For MNIST, Fashion-MNIST and CIFAR-10, we have trained 3 separate classifiers to assess quality of the method. For each dataset, the architecture is the same with the discriminator used for that dataset except the last layer which outputs 10 logits value instead of 1. We have used ADAM [102] optimizer with learning rate of 0.0002, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. Each model has been trained for 50k iterations with a batch size of 64. The accuracies of the classifier on test sets for MNIST, Fashion-MNIST and CIFAR-10 are 99.12, 91.20 and 84.20 respectively. During VennGAN

training, we measure the model at every $2k$ iterations and report the best average results. We have used our trained classifiers instead of Inception network as the latter is trained on high resolution RGB images which produces sub-optimal results for low resolution gray-scale MNIST and Fashion-MNIST datasets.

3.3.3 Illustrative Examples

We use mixture of Gaussians illustrative example to show that the proposed method works as anticipated. The nature of the dataset and its low dimensionality make it easier to spot subtle behaviours of our method e.g. whether the supports of the regions overlap etc. For this experiment we generate 3 different data distributions, where each data distribution equally mixes 4 out of 7 Gaussians, as shown in Fig. 3.4. The Gaussian means are located at $(0, 1)$, $(-1, 0)$, $(0, 0)$, $(1, 0)$, $(-1, -1)$, $(0, -1)$, and $(1, -1)$. Each Gaussian is two dimensional, isotropic, and has $\sigma = 0.2$.

FIGURE 3.4: Samples from the data distributions for the illustrative example. Each distributions equally mixes 4 out of 7 Gaussians.

In order to model these distributions, we have used d_2 type (refer to Fig. 3.2). The experiments are conducted with independent generators; we use 7 generators and 3 discriminators. The network architecture for generators is 4 fully connected layer followed by LeakyReLU except the last one which is linear. Each discriminator is constructed from 4 fully connected layers followed by LeakyReLU except the last one which is linear. In both networks, each layer has 256 units while last layer of generator has 2 and last layer of the discriminator has 1. Prior distribution for

the generators is a 128-dimensional isotropic Gaussian distribution. We have used ADAM [102] optimizer with learning rate of 0.0002, $\beta_1 = 0.0$ and $\beta_2 = 0.9$. The optimization of discriminator and generator follows alternating update rule with single discriminator update per generator update. The model has been trained for $5k$ iterations. For each region (generator), we use a batch size of 64. λ of R_1 regularizer [9] is 0.1.

For this experiment, we show the results in Fig. 3.5. By comparing it with Fig. 3.4 and with the color annotations given in the caption of Fig. 3.5, we can see that all the regions are generated at the correct position, e.g. the pink samples generated by r_7 , which is the common mode of all the distributions. No single region is excluded from the modeling, and standard deviation of each Gaussian roughly equal to the ones in Fig. 3.4. We have conducted this experiment multiple times with no notable difference which shows the method consistently converges to the correct solution with all the modes. We have not done a quantitative analysis for this experiment as it is only an illustrative case.

FIGURE 3.5: Generated regions for the illustrative example. Annotation of the regions w.r.t. d_2 of Fig. 3.2: r_1 is blue, r_2 is orange, r_3 is green, r_4 is brown, r_5 is purple, r_6 is red and r_7 is pink.

3.3.4 Main Experiments

In order to investigate the working dynamic of our proposed method and show its effectiveness, we have constructed multiple real-world datasets by utilizing MNIST, Fashion-MNIST, CIFAR-10 and CelebA. With these datasets, we constructed 2- and 3-distributions by using the class information from the datasets as per Table 3.7. For all cases, each distribution contains 2000 samples from the classes it includes. We never use the same sample twice for different distributions, which could lead to trivial solutions. For cases A, B and C we have used Venn diagram type of d_1 , d_2 and d_3 , respectively by utilizing our prior knowledge. We first illustrate the qualitative results and interpret them. Then we show and discuss the quantitative results for the same experiments. Following that we discuss the effect of independent or conditional generators. Later, we also discuss the effect of the classifier in the objective.

TABLE 3.7: Configurations of artificial datasets. Refer to Table 3.8 to check correspondence of labels for Fashion-MNIST and CIFAR-10.

Case	Distributions	Sets
A	2	$S_1 = \{0, 1, 2, 3, 4, 5, 6\}$, $S_2 = \{3, 4, 5, 6, 7, 8, 9\}$
B	3	$S_1 = \{0, 3, 5, 6\}$, $S_2 = \{1, 4, 5, 6\}$, $S_3 = \{2, 3, 4, 6\}$
C	3	$S_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $S_2 = \{3, 4, 5, 6, 7, 8, 9\}$, $S_3 = \{6, 7, 8, 9\}$

TABLE 3.8: Correspondence of labels for Fashion-MNIST and CIFAR-10.

	0	1	2	3	4	5	6	7	8	9
Fashion-MNIST	T-shirt/top	Trousers	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
Cifar-10	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck

Figs. 3.6, 3.7, and 3.8 show the results for cases A, B, and C respectively. In each case, the caption provides information about which region is visualized with accompanying set operations. The correctness of the generations can be traced by Table 3.7.

In case A of MNIST (see Fig. 3.6), $S_1 \setminus S_2 = \{0, 1, 2\}$, $S_2 \setminus S_1 = \{7, 8, 9\}$ and $S_2 \cap S_1 = \{3, 4, 5, 6\}$ are correctly modeled. Similarly, for Fashion-MNIST, $S_1 \setminus S_2 =$

$\{T\text{-shirt}, \text{Trousers}, \text{Pullover}\}$, $S_2 \setminus S_1 = \{Sneaker, Bag, Ankle\ boot}\}$ and $S_2 \cap S_1 = \{Dress, Coat, Sandal, Shirt}\}$ correctly generates right objects in the right regions. In both cases, we do not see any sign of dropping (missing) objects in any region. CIFAR-10 image quality is not as good as the others, so it is not easy to make a judgment from qualitative results. However, from the recognizable classes we see that *Automobile*, *Horse*, *Ship*, and *Truck* appear in the right region.

FIGURE 3.6: MNIST, Fashion-MNIST, CIFAR-10 results for case A.

In case B (Fig. 3.7) of MNIST and Fashion-MNIST, almost all the objects appear in the right region with good image quality. Similar to the previous case, we do

not observe any sign of missing objects from any regions. Also we observe a good amount of coverage of the modes e.g. various styles of digit 1. For CIFAR-10, the results are decent for *Airplane*, *Automobile*, *Deer*, however for other regions the quality is not satisfactory to make a complete commentary. Also there seems to be some amount of leaking between the regions.

FIGURE 3.7: MNIST, Fashion-MNIST, CIFAR-10 results for case B.

In case C (Fig. 3.8), we see near perfect performance in case of MNIST and Fashion-MNIST; objects are placed in the right regions and image quality is good enough to recognize the objects. Again, similar to the previous cases there is no sign of missing objects and there is high intra-class variation in generated images. Image quality of CIFAR-10 is again not that good, but the objects seem to be placed in the right regions e.g. $S_1 \setminus (S_2 \cap S_3)$ only includes *Airplane*, *Automobile*, and *Bird*, while $S_1 \cup S_2 \cup S_3$ only includes *Frog*, *Horse*, *Ship*, and *Truck*.

After presenting the qualitative results, Table 3.9 lists quantitative results for the experiments above. As expected, the average score for each setting is highly correlated with the qualitative interpretations above; MNIST and Fashion-MNIST scores are high while CIFAR-10 scores are relatively low. Also all the scores are significantly above the random baseline, which demonstrates the effectiveness of our model. The random baselines for cases A, B and C are 0.5, 0.25, and 0.61 respectively.

FIGURE 3.8: MNIST, Fashion-MNIST, CIFAR-10 results for case C.

When comparing the different settings (via average scores), we see that some settings are harder to model than the others e.g. case A is easier than case C which is easier than case B. Our interpretation is case A only contains 2 sets with 3 regions which makes it relatively easy to model. Case B on the other hand contains 3 sets with 7 regions which makes it harder than the others. These results hold across all the datasets.

Other than the average scores, we can analyze how well individual regions are modeled and how they compare to one another. Intuitively, simpler regions are

TABLE 3.9: Quantitative results on 3 datasets and 3 cases. Accuracy of each region and their average is reported. IG stands for Independent Generators and Avg is average value over the regions. The regions can be tracked from Fig. 3.2. n/a is placed into the regions where it does not exist in the type of Venn diagram. Average score for random baseline is 0.5, 0.25 and 0.61 for settings A, B and C respectively. We have bolded the best average results per dataset - case setting.

Dataset	Case	Classifier	IG	r_1	r_2	r_3	r_4	r_5	r_6	r_7	Avg
MNIST	A	Yes	Yes	99.76	99.11	81.72	n/a	n/a	n/a	n/a	93.53
MNIST	A	Yes	No	99.69	98.86	83.40	n/a	n/a	n/a	n/a	93.98
F-MNIST	A	Yes	Yes	91.37	87.75	80.17	n/a	n/a	n/a	n/a	86.43
F-MNIST	A	Yes	No	90.15	86.48	80.92	n/a	n/a	n/a	n/a	85.85
CIFAR-10	A	Yes	Yes	78.03	75.19	58.07	n/a	n/a	n/a	n/a	70.42
CIFAR-10	A	Yes	No	72.23	71.65	52.78	n/a	n/a	n/a	n/a	65.55
MNIST	B	Yes	Yes	99.33	100.0	95.67	98.44	98.22	99.64	99.58	98.70
MNIST	B	Yes	No	99.32	100.0	96.05	98.75	98.14	99.56	99.36	98.74
F-MNIST	B	Yes	Yes	73.03	97.36	70.43	68.33	91.02	92.09	18.59	72.97
F-MNIST	B	Yes	No	71.86	98.07	68.45	71.04	93.17	91.54	18.02	73.16
CIFAR-10	B	Yes	Yes	83.57	58.71	10.63	53.14	2.81	51.93	28.28	41.30
CIFAR-10	B	Yes	No	88.3	52.84	11.43	51.99	2.98	52.78	35.29	42.23
MNIST	C	Yes	Yes	99.5	n/a	n/a	n/a	n/a	93.85	94.19	95.85
MNIST	C	Yes	No	99.12	n/a	n/a	n/a	n/a	93.08	93.64	95.28
F-MNIST	C	Yes	Yes	94.88	n/a	n/a	n/a	n/a	85.25	67.49	82.54
F-MNIST	C	Yes	No	94.41	n/a	n/a	n/a	n/a	83.17	67.5	81.69
CIFAR-10	C	Yes	Yes	85.63	n/a	n/a	n/a	n/a	70.57	63.83	73.34
CIFAR-10	C	Yes	No	77.39	n/a	n/a	n/a	n/a	66.82	61.85	68.69
MNIST	A	No	No	99.54	98.88	81.45	n/a	n/a	n/a	n/a	93.29
F-MNIST	A	No	No	90.48	86.59	80.12	n/a	n/a	n/a	n/a	85.73
CIFAR-10	A	No	Yes	76.4	73.32	60.93	n/a	n/a	n/a	n/a	70.22
MNIST	B	No	No	98.72	99.99	95.28	99.08	97.40	99.29	99.29	98.43
F-MNIST	B	No	No	67.89	97.81	63.79	68.18	88.98	91.91	15.68	70.61
CIFAR-10	B	No	Yes	85.17	51.64	9.27	51.44	2.89	46.48	22.97	38.55
MNIST	C	No	No	98.49	n/a	n/a	n/a	n/a	92.88	93.71	95.03
F-MNIST	C	No	No	92.57	n/a	n/a	n/a	n/a	84.04	67.24	81.28
CIFAR-10	C	No	Yes	86.14	n/a	n/a	n/a	n/a	71.85	61.77	73.25

modeled better, e.g. case B of MNIST r_2 (digit 1) obtains a perfect score, or case B of Fashion-MNIST r_2 (Trousers) shows better performance than the other regions. Likewise, r_1 (Airplane), r_2 (Automobile), r_4 (Cat) and r_6 (Dog) of case B CIFAR-10 are modeled better than the other regions. Similar observations can be found in cases A and C.

3.3.4.1 Conditional Generator versus Independent Generators

Table 3.9 also shows the effect of conditional generator and independent generators. In case of MNIST and Fashion-MNIST, conditional generators produce comparable results to independent ones, while independent generators tend to be better for CIFAR-10. We postulate that in case of simple datasets, single conditional generator has sufficient capacity to match the quality of multiple generators. Besides, sharing most of the weights with different regions regularizes the training, as there are many common features between regions. However when it comes to CIFAR-10, sharing weights might be a burden for the representation of different regions rather than a regularization. This observation is similar to the capacity issue we encountered in Chapter 5.4.3 i.e. the architecture used may not have enough capacity to model CIFAR-10.

3.3.4.2 Effect of the Classifier

The bottom section of Table 3.9 shows 9 different settings without classifier term in the objective. In order to reduce the number of setting, we use conditional generators for MNIST and Fashion-MNIST and independent generators for CIFAR-10 due to reasons explained in Section 3.3.4.1. At all the settings there are slight but consistent improvements. Also for CIFAR-10, improvements are more significant than for the other datasets. The results encourage the use of the classifier especially for complex datasets.

3.3.4.3 CelebA Results

After showing qualitative and quantitative results for MNIST, Fashion-MNIST, and CIFAR-10, we conduct similar experiments for CelebA. For this dataset, we consider two cases. In case A, the first distribution contains only male faces while the second one contains females. In case B, the first distribution contains faces from both genders while the second one contains only males. We use d_1 and

d_3 types Venn diagrams to model cases A and B respectively. Our aim is to see whether semantic commonalities and differences of the distributions can be captured successfully. In case A, there should be no overlap in genders but we are interested in what type of commonalities our method can find. We have used conditional generator for these experiments to see the semantic relations between the regions more clearly.

In case A (Fig. 3.9), $S_1 \setminus S_2$ depicts stereotype masculine faces with short hair and masculine faces, whereas $S_2 \setminus S_1$ exhibits predominantly feminine features like long hair etc. On the other hand, $S_1 \cap S_2$ features faces which are neither predominantly male nor female. As the images in different regions are generated with the same noise, pose and background of an image at different regions remain similar, while the facial attributes change.

FIGURE 3.9: CelebA case A results: $S_1 \setminus S_2$ is only male figures, $S_2 \setminus S_1$ is only female figures, $S_1 \cap S_2$ is intersection.

Similarly, case B (Fig. 3.10) shows that the model can capture commonalities of the distributions well, $S_1 \cap S_2$, correctly capture the male faces, while the difference, $S_1 \setminus S_2$, are female faces as it should be. Again, due to the same noise, generations between different regions can be compared semantically. From $S_1 \cap S_2$ to $S_1 \setminus S_2$, the images do not change much other than the gender related features, which makes this model a type of image-to-image transformer. Both experiments show that VennGAN can capture high level semantic commonality between high dimensional complex distributions. In both cases A and B, we do not observe any mode collapse/drop behaviour.

FIGURE 3.10: CelebA case B results: $S_1 \setminus S_2$ is only female figures, $S_1 \cap S_2$ is only male figures.

3.3.4.4 Overall Discussion

In this chapter, we have used prior knowledge to choose the Venn type or $\mathbf{O}$ matrix. When we know that the distributions have intersections and unique parts, d_1 or d_2 type has been used; if a distribution is subset of another one, then we have utilized d_3 . If we have prior knowledge about the type of the distributions, then this method can be utilized easily. In case we have no prior knowledge about it, we can try to estimate it by either making assumptions or using domain knowledge. Ideally, it should be learned, which we leave it for the future work. In an ideal situation we should optimize $\mathbf{O}$ end-to-end with the model parameters. The main challenge is to find a meaningful reward signal for the training of $\mathbf{O}$, where the loss should negatively correlate with *leaks* between the regions. We think this is an important future research direction.

3.4 Application: Incorporating External Knowledge

In the previous section we have showed the effectiveness of our VennGAN modeling. In this section we will use VennGAN as an application to incorporate external knowledge to distribution modeling. The premise is that the individual distributions can be modeled better if some prior about their commonalities and particularities known beforehand. Note that we do not have labels associated with data samples, but a simple prior about the the distribution of the samples. The main aim here is to improve modeling of each distribution better whether it is quality of the samples or coverage of the supports.

Our VennGAN method offers a way to improve distribution modeling, for which we provide comparisons with existing approaches. To make our point we have designed an experimental setup where there are n distributions we would like to model. As baseline for this modeling, we use a conditional GAN [11] that generates each distribution; one per condition. We choose the baseline as a conditional GAN instead of independent generators to make the comparison more fair. For each distribution, we use a separate discriminator (again similar to our method to make it fair). The only difference of this baseline from the VennGAN setting is that it does not consider the relationship between the distributions. More formally, the baseline corresponds to Eq. 3.5 with $\mathbf{O} = \mathbf{I} \in \mathbb{R}^{n \times n}$, where $\mathbf{I}$ is an identity matrix and n is the number of distributions.

We train both models for 100k iterations on MNIST, Fashion-MNIST and CIFAR-10 datasets. The setting for the distributions in this section is the same as in the previous one (see Table 3.7). For hyper-parameters and network architecture, we follow the same setting as in the main experiments. To evaluate the performance of each distribution, we use Fréchet Inception Distance (FID) [45] which measures both quality and coverage of samples (i.e., recall, or mode collapse). For FID, we evaluate statistics of real and fake samples by using the penultimate layer of the

TABLE 3.10: Comparison of VennGAN and baseline conditional generator in terms of FID measured for each distribution. FID_i indicates FID scores between i -th generator distribution, p_{g_i} and i -th data distribution, p_{data_i} . Lower scores are better.

Dataset	Case	VennGAN	FID_1	FID_2	FID_3	Avg
MNIST	A	Yes	166.19	179.11	n/a	172.65
MNIST	A	No	187.19	204.33	n/a	195.76
F-MNIST	A	Yes	141.15	122.76	n/a	131.96
F-MNIST	A	No	187.19	204.33	n/a	195.76
CIFAR-10	A	Yes	1948.64	1502.09	n/a	1725.37
CIFAR-10	A	No	1701.51	1824.53	n/a	1763.02
MNIST	B	Yes	110.81	102.68	121.09	111.52
MNIST	B	No	122.31	123.25	97.09	114.22
F-MNIST	B	Yes	110.39	108.48	90.37	103.08
F-MNIST	B	No	120.97	123.30	94.74	113.00
CIFAR-10	B	Yes	1666.2	1766.61	1449.28	1627.36
CIFAR-10	B	No	2161.85	1745.59	1483.62	1797.02
MNIST	C	Yes	96.66	91.88	103.3	97.28
MNIST	C	No	266.46	118.91	115.12	166.83
F-MNIST	C	Yes	120.01	106.50	109.89	112.13
F-MNIST	C	No	275.56	118.79	116.56	170.30
CIFAR-10	C	Yes	1322.37	1483.28	2092.64	1632.76
CIFAR-10	C	No	2208.2	2015.89	1583.38	1935.82

classifiers that are used to evaluate accuracy scores. FID scores are computed with 10k generated samples from each distribution.

We tabulate all our results in Table 3.10, which show that VennGAN consistently improves the modeling of each distribution. In every single setting, the average FID score is lower for VennGAN, even though some individual distribution comparisons are not better. For individual distribution comparisons, VennGAN wins 20 times out of 24. In some settings (e.g. FID_1 of C), the improvement is by a large margin. The results show that VennGAN-type modeling is superior than modeling each distribution separately without utilizing prior knowledge.

3.5 Summary

In conclusion, we have proposed a novel multi-distribution GAN method which can discover particularities and commonalities of the distributions. Our method models each data distribution as a mixture of generator distributions. As the generators are partially shared between the modeling of different true data distributions, shared ones capture the commonality of the distributions, while non-shared ones capture unique aspects of them. We have successfully trained it on various datasets to show its effectiveness. Additionally, we show how our method can be used to incorporate external (prior) knowledge to distribution modeling in order to improve performance of each distribution. We believe this method has good potential for new applications and better data modeling.

Chapter 4

The Unusual Effectiveness of Averaging in GAN Training

4.1 Introduction

Generative Adversarial Networks (GANs) [4] are two-player zero-sum games. They are known to be notoriously hard to train, unstable, and often don't converge. There has been a lot of recent work to improve the stability of GANs by addressing various underlying issues. Searching for a more stable function family [5] and utilizing better objectives [6, 55] have been explored for training and instability issues. For tackling non-convergence, regularizing objective functions [31, 48], two time-scale techniques [45] and extra-gradient (optimistic) methods [46, 47] have been studied.

One reason for non-convergence is cycling around an optimal solution [103, 104], or even slow outward spiraling [105]. There might be multiple reasons for this cycling behavior: (i) eigenvalues of the Jacobian of the gradient vector field have zero real part and a large imaginary part [48]. (ii) Gradients of the discriminator may not move the optimization in the right direction and may need to be updated many times to do so [55]. (iii) Sampling noise causes the discriminator to move in different

directions at each iteration, as a result of which the generator fluctuates. (iv) Even if the gradients of the discriminator point in the right direction, the learning rate of the generator might be too large. When this overshooting happens, the gradients of the discriminator lead in the wrong direction due to over-confidence [9].

In this work, we explore in detail simple strategies for tackling the cycling behavior without influencing the adversarial game. Our strategies average generator parameters over time *outside* the training loop. Averaging generator and discriminator parameters is known to be an optimal solution for convex-concave min-max games [106]. However, no such guarantees are known (even for bilinear games) if we apply exponential discounting.

Our contributions are the following: (i) We show theoretically that although Exponential Moving Average (EMA) does not converge to equilibrium, even in bilinear games, it nevertheless helps to stabilize cyclic behavior by shrinking its amplitude. In non-bilinear settings it preserves the stability of locally stable fixed points. (ii) We demonstrate that both averaging techniques consistently improve results for several different datasets, network architectures, and GAN objectives. (i) We compare it with several other methods that try to alleviate the cycling or non-convergence problem and demonstrate its unusual effectiveness.

4.2 Method

In case of convex-concave min-max games, it is known that the average of generator/discriminator parameters is an optimal solution [106], but there is no such guarantee for a non-convex/concave setting, which is the case for real-world application of GANs. Moving Average (MA) over parameters θ is an efficient implementation of uniform averaging without saving iterates at each time point:

$$\theta_{MA}^{(t)} = \frac{t-1}{t} \theta_{MA}^{(t-1)} + \frac{1}{t} \theta^{(t)} \quad (4.1)$$

When the generator iterates reach a stationary distribution, Eq.4.1 approximates its mean. However there are two practical difficulties with this equation. First we do not know at which time point the iterates reach a stationary distribution. Second, and more importantly, iterates may not stay in the same distribution after a while. Indeed, both of these challenges are commonly observed in GANs training. Averaging over samples coming from different distributions can produce worse results, which we have also observed in our own experiments.

Because of the issues stated above, we use Exponential Moving Average (EMA):

$$\theta_{EMA}^{(t)} = \beta \theta_{EMA}^{(t-1)} + (1 - \beta) \theta^{(t)} \quad (4.2)$$

where $\theta_{EMA}^{(0)} = \theta^{(0)}$. EMA computes an exponentially discounted sum of the parameters. It has an effective time window over iterates, where early iterates fade at a rate depending on the β value. As β approaches 1, averaging effectively computes longer time windows.

Both EMA and MA offer performance gains, with EMA typically being more robust, might be due to the challenges mentioned above (for more details see Section 4.4). Averaging methods are simple to implement and have minimal computation overhead. As they operate outside of the training loop, they do not influence optimal points of the adversarial game. Hyperparameter search for EMA can be done with a single training run by simultaneously keeping track of multiple averages with different β parameters.

4.3 Why Does Averaging Work?

In this section we provide two types of theoretical analyses to answer empirical performance of averaging in GANs. We start from the global convergence analysis, which uses bilinear games with continuous time updates. In the second type, we make a local convergence analysis with non-convex/concave setting and discrete-time updates.

4.3.1 Global Analysis for Simple Bilinear Games

As we will show experimentally, parameter averaging provides significant benefits in terms of GAN training across a wide range of different setups. Although one cannot hope to completely justify this experimental success theoretically in general non-convex-concave settings, it is at least tempting to venture some informed conjectures about the root causes of this phenomenon. To do so, we focus on the simplest possible class of saddle point problems, the class of planar bilinear problems (i.e. zero-sum games such as Matching Pennies). This is a toy, albeit classic, pedagogical example used to explain phenomena arising in GANs. Specifically, similar to Goodfellow [107], we also move to a continuous time model to simplify the mathematical analysis.

In terms of the moving average method, it is well known that time-averaging suffices to lead to convergence to equilibria [106]. In fact, it has recently been established that the smooth (continuous-time) analogues of first order methods such as on-line gradient descent (follow-the-regularized leader) in bilinear zero-sum games are recurrent (i.e. effectively periodic) with trajectories cycling back into themselves [103]. As a result, the time-average of the system converges fast, at a $\mathcal{O}(1/T)$ rate, to the solution of the saddle point problem [103]. These systems have connections to physics [108].

In contrast, not much is known about the behavior of EMA method even in the simplest case of zero-sum games. As we will show by analyzing the simplest case of a bilinear saddle problem $\max_x \min_y xy$, even in this toy case EMA does not lead to convergent behavior, but instead reduces the size of the oscillations, leading to more stable behavior whilst enabling exploration in a close neighborhood around the equilibrium. In this case the gradient descent dynamics are as follows:

$$\left\{ \begin{array}{l} \frac{dx}{dt} = y \\ \frac{dy}{dt} = -x \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{l} \frac{dx}{dt} = \frac{\partial H}{\partial y} \\ \frac{dy}{dt} = -\frac{\partial H}{\partial x} \end{array} \right\} \quad (4.3)$$

This corresponds to a standard Hamiltonian system whose energy function is equal to $H = \frac{x^2+y^2}{2}$. All trajectories are cycles centered at 0. In fact, the solution to the system has the form of a periodic orbit $(x(t), y(t)) = (a \cos(t), a \sin(t))$. These connections have recently been exploited by Balduzzi et al. [109] to design training algorithms for GANs (see also Bailey and Piliouras [108]). Here we take a different approach. For simplicity let's assume $a = 1$. The Exponential Moving Average method now corresponds to an integral of the form $C \cdot \int_0^T \beta^{T-t} \theta(t) dt$ where $\theta(t) = (\cos(t), \sin(t))$ and C a normalizing term such that the EMA of a constant function is equal to that constant (i.e. $C = 1 / \int_0^T \beta^{T-t} dt$). Some simple algebra implies that $C \cdot \int_0^T \beta^{T-t} \sin(t) dt = \frac{-\ln(\beta)}{1+\ln^2(\beta)} \frac{\beta^T - \cos(T) - \ln(\beta) \sin(T)}{1-\beta^T}$. As the terms β^T vanish exponentially fast, this function reduces to $\frac{\ln(\beta)}{1+\ln^2(\beta)} (\cos(T) + \ln(\beta) \sin(T))$. This is a periodic function, similarly to the components of $\theta(t)$, but its amplitude is significantly reduced, and as a result it is more concentrated around its mean value (which is the same as its corresponding $\theta(t)$ component, i.e. zero). For β very close to 1, e.g. $\beta = 0.999$ this periodic function can be further approximated by $\ln(\beta) \cos(T)$. In other words, no matter how close to 1 we choose the discounting factor β , even in the toy case of a planar bilinear saddle problem (i.e. the Matching Pennies game) EMA has a residual non-vanishing periodic component, alas of very small amplitude around the solution of the saddle point problem (see Fig. 4.1).

FIGURE 4.1: MA versus EMA for various values of β .

4.3.2 Local Stability Analysis for GANs

The model of the previous section introduces two simplifications so as to be theoretically tractable: i) it focuses on a (small) bilinear game, and ii) it is a continuous time model. Despite these simplifications the global behavior is still rather complex, indicating the difficulty of completely analyzing a more realistic model. Nevertheless, in this section we examine such a model (discrete-time non-convex loss functions) and show that EMA preserves the stability of locally stable fixed points (Nash equilibria), providing further theoretical evidence in support of averaging. Due to the complexity of GANs, local stability analysis is often used as a method for obtaining theoretical insights, see e.g. Mescheder et al. [9], Nagarajan and Kolter [110].

Suppose we have a generator G parameterized by $\theta \in \mathbf{R}^p$, and a discriminator D parameterized by $\phi \in \mathbf{R}^q$; let $x = (\theta, \phi) \in \mathbf{R}^n$, $n = p + q$. Then, during training, G and D minimize loss functions $\mathcal{L}^{(\theta)}(\theta, \phi)$ and $\mathcal{L}^{(\phi)}(\theta, \phi)$ respectively. Following the same formulation as in Gidel et al. [111]:

$$\theta^* \in \underset{\theta}{\operatorname{argmin}} \mathcal{L}^{(\theta)}(\theta, \phi^*) \quad \phi^* \in \underset{\phi}{\operatorname{argmin}} \mathcal{L}^{(\phi)}(\theta^*, \phi) \quad (4.4)$$

For specific possible instantiations of these loss functions see Goodfellow et al. [4], Arjovsky et al. [55]. We define the gradient vector field $V(\theta, \phi) \in \mathbf{R}^n$ and its Jacobian $J_V(\theta, \phi) \in \mathbf{R}^{n \times n}$ as follows:

$$V(\theta, \phi) = \begin{bmatrix} \nabla_{\theta} \mathcal{L}^{(\theta)}(\theta, \phi) \\ \nabla_{\phi} \mathcal{L}^{(\phi)}(\theta, \phi) \end{bmatrix} \quad J_V(\theta, \phi) = \begin{bmatrix} \nabla_{\theta}^2 \mathcal{L}^{(\theta)}(\theta, \phi) & \nabla_{\phi} \nabla_{\theta} \mathcal{L}^{(\theta)}(\theta, \phi) \\ \nabla_{\theta} \nabla_{\phi} \mathcal{L}^{(\phi)}(\theta, \phi) & \nabla_{\phi}^2 \mathcal{L}^{(\phi)}(\theta, \phi) \end{bmatrix} \quad (4.5)$$

In this analysis, we assume that G and D are trained with some method that can be expressed as the iteration:

$$x_t = F(x_{t-1}) \quad (4.6)$$

for some function F , where $x_t = (\theta_t, \phi_t)$ denotes the parameters obtained after t training steps. This framework covers all training algorithms that rely only on a single past iterate to compute the next iterate, and includes simultaneous gradient descent and alternating gradient descent methods. More concretely, simultaneous gradient descent with a learning rate η can be written in this form as follows:

$$F_\eta(x) = x - \eta V(x). \quad (4.7)$$

EMA then averages the iterates x_t to produce a new sequence of iterates $\hat{x}_t$ such that:

$$\begin{aligned} \hat{x}_t &= \beta \hat{x}_{t-1} + (1 - \beta)x_t && \text{(by definition (4.2))} \\ &= \beta \hat{x}_{t-1} + (1 - \beta)F(x_{t-1}) && \text{(Replace } x_t \text{ using Eq. (4.6))} \\ &= \beta \hat{x}_{t-1} + (1 - \beta)F\left(\frac{\hat{x}_{t-1} - \beta \hat{x}_{t-2}}{1 - \beta}\right) && \text{(Replace } x_{t-1} \text{ using definition (4.2))} \end{aligned}$$

We have thus express the EMA algorithm purely in terms of a new set of iterates $\hat{x}_t$. Note that computing the next iterate now requires the past two iterates. Flammarion and Bach [112] observe that the standard moving average can similarly be expressed as a second-order difference equation. We can then define the EMA operator $\hat{F}_\beta(x_t, x_{t-1})$ such that $\begin{bmatrix} x_t \\ x_{t-1} \end{bmatrix} = \hat{F}_\beta(x_{t-1}, x_{t-2})$:

$$\hat{F}_\beta(x_t, x_{t-1}) = \begin{bmatrix} \beta \mathbf{I}_n & \mathbf{0}_n \\ \mathbf{I}_n & \mathbf{0}_n \end{bmatrix} \begin{bmatrix} x_t \\ x_{t-1} \end{bmatrix} + (1 - \beta) \begin{bmatrix} F(A[x_t \ x_{t-1}]^T) \\ \mathbf{0}_n \end{bmatrix} \quad (4.8)$$

for $A = \frac{1}{1-\beta}[\mathbf{I}_n \ -\beta\mathbf{I}_n]$. The Jacobian of $\hat{F}_\beta$, $J_{\hat{F}_\beta}$ is therefore:

$$\begin{aligned} J_{\hat{F}_\beta}(x_t, x_{t-1}) &= \begin{bmatrix} \beta \mathbf{I}_n & \mathbf{0}_n \\ \mathbf{I}_n & \mathbf{0}_n \end{bmatrix} + (1 - \beta) \begin{bmatrix} J_F(A[x_t \ x_{t-1}]^T) \cdot A \\ \mathbf{0}_n \end{bmatrix} \\ &= \begin{bmatrix} \beta \mathbf{I}_n + J_F(A[x_t \ x_{t-1}]^T) & -\beta J_F(A[x_t \ x_{t-1}]^T) \\ \mathbf{I}_n & \mathbf{0}_n \end{bmatrix} \end{aligned} \quad (4.9)$$

We can now proceed to explore the behavior of the EMA algorithm near a local Nash equilibrium by examining the eigenvalues of its Jacobian $J_{\hat{F}_\beta}$, which are described by Theorem 4.3.1.

Theorem 4.3.1. The eigenvalues of the Jacobian $J_{\hat{F}_\beta}(x^*, x^*)$ at a local Nash equilibrium x^* are β (with multiplicity n) and the eigenvalues of $J_F(x^*)$, the Jacobian of the training operator at the equilibrium.

Proof. We solve $\det(\gamma \mathbf{I}_{2n} - J_{\hat{F}_\beta}) = 0$ to obtain eigenvalues γ . Some algebra reveals that

$$\begin{aligned} \det(\gamma \mathbf{I}_{2n} - J_{\hat{F}_\beta}) &= \begin{vmatrix} (\gamma - \beta) \mathbf{I}_n - J_F(x^*) & \beta J_F(x^*) \\ -\mathbf{I}_n & \gamma \mathbf{I}_n \end{vmatrix} \\ &= |\gamma(\gamma - \beta) \mathbf{I}_n - \gamma J_F(x^*) + \beta J_F(x^*)| \\ &= |(\gamma - \beta)(\gamma \mathbf{I}_n - J_F(x^*))| \\ &= (\gamma - \beta)^n \det(\gamma \mathbf{I}_n - J_F(x^*)) \end{aligned}$$

where we use the following equality (e.g. see Lemma 1 in Gidel et al. [111]): $|\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}| = |AD - BC|$ if C and D commute. Thus, $\det(\gamma \mathbf{I}_{2n} - J_{\hat{F}_\beta}) = 0$ if $\gamma = \beta$ or $\det(\gamma \mathbf{I}_n - J_F(x^*)) = 0$ so γ is an eigenvalue of $J_F(x^*)$. $\square$

Given that $0 < \beta < 1$ in the EMA algorithm, we have the following corollary that formalizes the intuition that the dynamics of EMA near a local Nash equilibrium are tightly linked to those of the underlying training algorithm.

Corollary 4.3.1.1. Assuming that $0 < \beta < 1$, the eigenvalues of J_F lie within the unit ball if and only if the eigenvalues of $J_{\hat{F}_\beta}$ lie within the unit ball.

Also, we see that EMA could potentially slow down the rate of convergence compared to the underlying training method if β is larger than the magnitude of all other eigenvalues of J_{F_β} .

4.4 Experiments

We use both illustrative examples (i.e. mixtures of Gaussians) as well as four commonly used real-world datasets, namely CIFAR-10 [91], STL-10 [92], CelebA [93], and ImageNet [94] to show the effectiveness of averaging. For mixtures of Gaussians, the architecture, hyperparameters and other settings are defined in its own section. For CelebA, 64x64 pixel resolution is used, while all other experiments are conducted on 32x32 pixel images.

All dataset samples are scaled to $[-1, 1]$ range. No labels are used in any of the experiments. For the optimizer, we use ADAM [102] with $\alpha = 0.0002$, $\beta_1 = 0.0$ and $\beta_2 = 0.9$. For the GAN objective, we use the original GAN (Referring to Non-Saturating variant) [4] objective and the Wasserstein-1 [55] objective, with Lipschitz constraint satisfied by gradient penalty [6]. Spectral normalization [7] is only used with the original objective. In all experiments, the updates are alternating gradients. A detailed settings and hyperparameters are listed in Table 4.1.

TABLE 4.1: Settings for CIFAR-10, STL-10, CelebA, ImageNet

---

batch size = 64
discriminator learning rate = 0.0002
generator learning rate = 0.0002
ADAM $\beta_1 = 0.0$
ADAM $\beta_2 = 0.9$
ADAM $\epsilon = 1e - 8$
$\beta = 0.9999$ for EMA
max iteration = 500000
WGAN-GP $\lambda = 10.0$
WGAN-GP $n_{dis} = 5$
MA start point = 100000
GAN objective = GAN or WGAN-GP
Optimizer = ADAM

The network architectures are similar to the full version of a progressively growing GAN [8] as in Table 4.2 and Table 4.3. We call this architecture conventional. ResNet from Gulrajani et al. [6] is used as a secondary architecture. Unless stated

otherwise, the objective is the original one, the architecture is conventional, discriminator to generator update ratio is 1, β value is 0.9999 and MA starting point is $100k$ ($50k$ for CelebA). When spectral normalization is used in ResNet, the feature number in each layer is doubled by 2. Prior distribution for the generator is a 512-dimensional isotropic Gaussian distribution for conventional architecture and 128 for ResNet. Samples from the distribution are normalized to make them lie on a unit hypersphere before passing them into the generator.

TABLE 4.2: Conventional Generator Architecture for 32x32 resolution

Layers	Act.	Output Shape
Latent vector	-	512 x 1 x 1
Conv 4 x 4	BatchNorm - LReLU	512 x 4 x 4
Conv 3 x 3	BatchNorm - LReLU	512 x 4 x 4
Upsample	-	512 x 8 x 8
Conv 3 x 3	BatchNorm - LReLU	256 x 8 x 8
Conv 3 x 3	BatchNorm - LReLU	256 x 8 x 8
Upsample	-	256 x 16 x 16
Conv 3 x 3	BatchNorm - LReLU	128 x 16 x 16
Conv 3 x 3	BatchNorm - LReLU	128 x 16 x 16
Upsample	-	128 x 32 x 32
Conv 3 x 3	BatchNorm - LReLU	64 x 32 x 32
Conv 3 x 3	BatchNorm - LReLU	64 x 32 x 32
Conv 1 x 1	-	3 x 32 x 32

Inception [31] and FID [45] scores are used as quantitative measures. Higher inception scores and lower FID scores are better. We have used the online Chainer implementation¹ for both scores. For Inception score evaluation, we follow the guideline from Salimans et al. [31]. The FID score is evaluated on 10k generated images and statistics of data are calculated at the same scale of generation e.g. 32x32 data statistics for 32x32 generation. The maximum number of iterations for any experiment is 500k. For each experiment, we state the training duration. At every 10k iterations, Inception and FID scores are evaluated, and we pick the best

¹ <https://github.com/pfnet-research/chainer-gan-lib/blob/master/common/evaluation.py>

TABLE 4.3: Conventional Discriminator Architecture for 32x32 resolution

Layers	Act.	Output Shape
Input image	-	3 x 32 x 32
Conv 1 x 1	LReLU	64 x 32 x 32
Conv 3 x 3	LReLU	64 x 32 x 32
Conv 3 x 3	LReLU	128 x 32 x 32
Downsample	-	128 x 16 x 16
Conv 3 x 3	LReLU	128 x 16 x 16
Conv 3 x 3	LReLU	256 x 16 x 16
Downsample	-	256 x 8 x 8
Conv 3 x 3	LReLU	256 x 8 x 8
Conv 3 x 3	LReLU	512 x 8 x 8
Downsample	-	512 x 4 x 4
Conv 3 x 3	LReLU	512 x 4 x 4
Conv 3 x 3	LReLU	512 x 4 x 4
Linear	-	1

point. All experiments are repeated at least 3 times with random initializations to show that the results are consistent across different runs.

In addition to these experiments, we also compare the averaging methods, with *Consensus Optimization* [48], *Optimistic Adam* [46] and *Zero Centered Gradient Penalty* [9] on the Mixture of Gaussians data set and CIFAR-10. These methods are known to improve stability and reduce non-convergence, which make them good comparison (baseline) methods to averaging. For CIFAR-10 comparison, we use a DCGAN-like architecture [5] and follow the same hyperparameter setting as in Mixture of Gaussians except WGAN-GP’s λ which is 10.0.

To ensure a fair comparison, we use the same samples from the prior distribution for both averaged and non-averaged generators. In this way, the effect of averaging can be directly observed, as the generations are semantically similar. We did not cherry-pick any images; all images generated are from randomly selected samples from the prior.

4.4.1 Mixture of Gaussians

This illustrative example is a two-dimensional mixture of 16 Gaussians, where the mean of each Gaussian lies on the intersection points of a 4x4 grid: $(-3, 3)$, $(-1, 3)$, $(1, 3)$, $(3, 3)$, $(-3, 1)$, $(-1, 1)$, $(1, 1)$, $(3, 1)$, $(-3, -1)$, $(-1, -1)$, $(1, -1)$, $(3, -1)$, $(-3, -3)$, $(-1, -3)$, $(1, -3)$, and $(3, -3)$. Each Gaussian is isotropic with $\sigma = 0.2$. Original GAN [4] is used as the objective function with Gradient Penalty from Gulrajani et al. [6], with 1:1 discriminator/generator update and alternative update rule. We run the experiment for 40k iterations, and measure Wasserstein-1² distance with 2048 samples from both distributions at every 2k iterations. We take the average of the last 10 measurements from the same experiment and repeat it 5 times. We compare this baseline with *Optimistic Adam* (OMD), *Consensus Optimization* (CO), and *Zero Centered Gradient Penalty* (Zero-GP) by using the same architecture, and following the original implementations as closely as possible. We have also combined EMA and MA with OMD, CO and Zero-GP to see whether they can improve over these methods. For all the experiments, we have used batch size of 64. Detailed settings for our baseline, *Optimistic Adam*, *Consensus Optimization* and *Zero-GP* settings are listed in Table 4.4, Table 4.5, Table 4.6 and Table 4.7 respectively. Generator has 4 layers with 256 units in each layer and an additional layer that projects into the data space. The discriminator also has 4 layers with 256 units in each layer and a classifier layer on top. ReLU activation function is used after each affine transformation.

Table 4.8 shows the distance for various methods. In all cases, EMA outperforms other methods, and interestingly, it also improves OMD, CO and Zero-GP. This indicates that OMD, CO and Zero-GP do not necessarily converge in non-convex/concave settings, but still cycle. MA also improves the results in certain cases, but not as much as EMA. Our observation is that uniform averaging over long time iterates in the non-convex/concave case hurts performance (more on this in later experiments). Because of this, we have started uniform averaging at later

---

² Python Optimal Transport package at <http://pot.readthedocs.io/en/stable/>

TABLE 4.4: Settings for Mixture of Gaussians

---

discriminator learning rate = 0.0002
generator learning rate = 0.0002
ADAM $\beta_1 = 0.0$
ADAM $\beta_2 = 0.9$
ADAM $\epsilon = 1e - 8$
$\beta = 0.999$ for EMA
GP $\lambda = 1.0$
MA start point = 20000
GAN objective = GAN
Optimizer = ADAM

TABLE 4.5: Settings for Mixture of Gaussians for *OMD*

---

discriminator learning rate = 0.0002
generator learning rate = 0.0002
ADAM $\beta_1 = 0.0$
ADAM $\beta_2 = 0.9$
ADAM $\epsilon = 1e - 8$
$\beta = 0.999$ for EMA
GP $\lambda = 1.0$
MA start point = 20000
GAN objective = GAN
Optimizer = OptimisticADAM

---

stages in training and treat it as a hyper-parameter. For this experiment, the start interval is selected as 20k.

Fig. 4.2 is an illustration of the above results. The blue regions indicate a lack of support, while the red ones indicate a region of support. Without averaging, the generator omits certain modes at different time points, and covers them later while omitting previously covered modes. We observe a clear fluctuation in this case, while the EMA consistently shows more balanced and stable supports across different iterations. OMD, CO and Zero-GP perform better than the baseline, however they seem to be less stable across different iterations than EMA results.

TABLE 4.6: Settings for Mixture of Gaussians for *CO*

---

discriminator learning rate = 0.0002
generator learning rate = 0.0002
RMSProp $\beta = 0.9$
RMSProp $\epsilon = 1e - 10$
$\beta = 0.999$ for EMA
Consensus $\gamma = 10.0$
MA start point = 20000
GAN objective = GAN
Optimizer = RMSPropOptimizer

TABLE 4.7: Settings for Mixture of Gaussians for *Zero-GP*

---

discriminator learning rate = 0.0002
generator learning rate = 0.0002
ADAM $\beta_1 = 0.0$
ADAM $\beta_2 = 0.9$
ADAM $\epsilon = 1e - 8$
$\beta = 0.999$ for EMA
<i>Zero-GP</i> $\lambda = 1.0$
MA start point = 20000
GAN objective = GAN
Optimizer = ADAM

TABLE 4.8: Wasserstein-1 Distance for non-averaged generator, EMA, MA, OMD, CO and Zero-GP

---

	no-average	EMA	MA
Baseline	0.0419 ± 0.0054	0.0251 ± 0.0026	0.0274 ± 0.0016
<i>OMD</i>	0.0494 ± 0.0041	0.0431 ± 0.0042	0.0525 ± 0.0024
<i>CO</i>	0.0286 ± 0.0034	0.0252 ± 0.0035	0.0390 ± 0.0105
<i>Zero-GP</i>	0.0363 ± 0.0018	0.0246 ± 0.0068	0.0317 ± 0.0121

4.4.2 Real-world Datasets

After showing the illustrative examples, we show our main results on CIFAR-10, STL-10, and ImageNet. Table 4.9 tabulates Inception and FID scores with no

FIGURE 4.2: Effect of averaging parameters for Mixture of Gaussians. From top to bottom: without average, EMA, MA, OMD, CO, Zero-GP.

averaging, EMA and MA. In case of CIFAR-10, both EMA and MA outperform baseline (no averaging) in general. In this case MA also outperforms EMA on IS, however FID favors EMA side. For CIFAR-10, there is no clear winner between EMA and MA. However, for the other datasets, EMA clearly surpass MA. Indeed, for STL-10 and ImageNet, MA can get worse than no averaging measured by FID. The main reason why MA gets worse results is that it averages over longer iterates

with equal weights. MA tends to reach its best results shortly after it starts operating, then it saturates or even gets worse (see Fig. 4.4 to Fig. 4.9 for illustration of this behavior). We have observed improvements when the starting point of MA is pushed later in training, however for convenience and ease of experimentation our main results are based on the EMA method.

TABLE 4.9: Inception and FID scores on CIFAR-10, STL-10 and ImageNet. (100k), (250k) etc. refer to number of iterations of the generator. Experiments are repeated 3 times.

	IS			FID		
	no average	EMA	MA	no average	EMA	MA
CIFAR-10						
conventional (500k)	8.14 ± 0.01	8.89 ± 0.12	8.74 ± 0.19	16.84 ± 1.00	12.56 ± 0.17	16.59 ± 0.52
conventional WGAN-GP $n_{dis} = 1$ (250k)	7.18 ± 0.06	7.85 ± 0.08	7.93 ± 0.15	25.51 ± 1.22	19.18 ± 0.95	22.71 ± 0.95
conventional WGAN-GP $n_{dis} = 5$ (250k)	7.22 ± 0.07	7.92 ± 0.10	8.20 ± 0.06	25.32 ± 0.70	19.72 ± 0.31	20.02 ± 0.75
ResNet $n_{dis} = 5$ (120k)	7.86 ± 0.13	8.46 ± 0.13	8.68 ± 0.09	20.64 ± 1.38	17.58 ± 1.84	18.30 ± 1.32
ResNet WGAN-GP $n_{dis} = 1$ (250k)	7.63 ± 0.12	8.22 ± 0.05	8.51 ± 0.07	21.49 ± 0.40	15.85 ± 0.27	16.27 ± 0.56
ResNet WGAN-GP $n_{dis} = 5$ (250k)	7.38 ± 0.26	7.94 ± 0.31	8.32 ± 0.25	23.20 ± 2.62	19.41 ± 2.57	19.33 ± 3.17
STL-10						
conventional (250k)	7.96 ± 0.35	8.39 ± 0.10	8.23 ± 0.24	22.33 ± 1.59	19.64 ± 1.38	26.11 ± 2.27
ImageNet						
conventional (500k)	8.33 ± 0.10	8.88 ± 0.14	8.33 ± 0.03	24.59 ± 0.59	21.83 ± 0.93	26.77 ± 1.12

4.4.3 CIFAR-10

Fig. 4.3 shows images generated with and without EMA for CIFAR-10 after 300k iterations. Objects exhibit fewer artifacts and look visually better with EMA. It is interesting to note that the averaged and non-averaged versions converge to the same object, but can have different color and texture.

Figs. 4.4, 4.5, 4.6, 4.7, 4.8, and 4.9 show Inception and FID scores during training for various settings. EMA model outperforms its non-averaged counter-part consistently by a large margin, which is more or less stable throughout the training. However MA model's FID performance reduces gradually as it considers longer time windows for averaging. This observation also holds in STL10 and ImageNet experiments (see Section 4.4.5). Our intuition is that this phenomenon occurs because generator iterates do not stay in the same parameter region and averaging over parameters of different regions produces worse results. Besides, we have seen clear disagreement between FID and IS in case of MA. We think this happens because

FIGURE 4.3: Generation for CIFAR-10 dataset. (left) Without averaging, (right) with EMA

IS is not a proper metric to measure difference between two distributions and has many known flaws such as it does not measure intra-class diversity [45, 100, 113].

FIGURE 4.4: CIFAR-10 FID and IS scores during training. Setting: original GAN objective, conventional architecture, $n_{dis} = 1$

Fig. 4.10 compares EMA, CO and OMD. EMA clearly improves every method it is included on top in both IS and FID. Especially, on OMD it has a large improvement, while the experiments on CO shows smaller but still consistent improvements. Our conclusion from this experiment is the same with the one in illustrative examples: OMD, CO and Zero-GP do not necessarily converge in non-convex/concave settings, but still cycle. It is beneficial to include EMA into any method we tested so far.

FIGURE 4.5: CIFAR-10 FID and IS scores during training. Setting: original GAN objective/ conventional architecture/ $n_{dis} = 5$

FIGURE 4.6: CIFAR-10 FID and IS scores during training. Setting: WGAN-GP objective/ conventional architecture/ $n_{dis} = 1$

FIGURE 4.7: CIFAR-10 FID and IS scores during training. Setting: WGAN-GP objective/ conventional architecture/ $n_{dis} = 5$

4.4.4 CelebA

Figs. 4.11, 4.12, 4.13, 4.14, 4.15, and 4.16 show the same 10 samples from the prior fed to the non-averaged generator, the EMA generator with various β values, and the MA generator at iterations $100k$, $150k$, $200k$, $250k$, $300k$, and $350k$,

FIGURE 4.8: CIFAR-10 FID and IS scores during training. Setting: original GAN objective/ ResNet architecture/ $n_{dis} = 5$

FIGURE 4.9: CIFAR-10 FID and IS scores during training. Setting: WGAN-GP objective/ ResNet architecture/ $n_{dis} = 1$

respectively. The non-averaged images show strong artifacts, but as we average over longer time windows (increased β), image quality improves noticeably. Interestingly, significant attribute changes occurs as β approaches 1; there is a shift in gender, appearance, hair color, background etc., although images are still similar in pose and composition. EMA results with $\beta = 0.999$ and $\beta = 0.9999$ also look better than MA results, which has higher weights on early iterations. This shows there is an optimal β /window size to tune for the best results. As the number of iteration increases, diversity and quality of images generated with MA decreases considerably, while EMA results more or less the same quality. This observation is parallel to the decrease of IS and FID scores in CIFAR-10 results (see Section 4.4.3).

Fig. 4.17 compares the stability of a non-averaged generator and EMA by using the same 2 samples from the prior. Starting from 50k iteration, images are generated

FIGURE 4.10: Comparison of EMA, OMD, and CO on CIFAR-10.

from both models with 10k interval until 200k iterations. Images from the non-averaged generator change attributes frequently, while only preserving generic image composition. Meanwhile, the averaged generator produces smoother changes, with attributes changing more slowly e.g. similar hair colour & style, gender, background etc.

FIGURE 4.11: Generation for CelebA dataset for various β values at 100k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA

FIGURE 4.12: Generation for CelebA dataset for various β values at 150k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA

FIGURE 4.13: Generation for CelebA dataset for various β values at 200k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA

FIGURE 4.14: Generation for CelebA dataset for various β values at 250k iterations. From top to bottom: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA.

FIGURE 4.15: Generation for CelebA dataset for various β values at 300k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA

FIGURE 4.16: Generation for CelebA dataset for various β values at 350k iteration. From top to bottom rows: (a) non-averaged generator, (b) $\beta = 0.9$, (c) $\beta = 0.99$, (d) $\beta = 0.999$, (e) $\beta = 0.9999$, (f) MA

FIGURE 4.17: Generation for CelebA dataset for the same 2 noise samples from 50k to 200k with 10k intervals. (top 2 rows) without averaging, (bottom two rows) with EMA $\beta = 0.9999$

4.4.5 STL-10 and ImageNet

Fig. 4.18 and Fig. 4.19 show images generated with and without EMA for STL-10 and ImageNet, respectively. Even though quantitative scores are better for EMA (see Table. 4.9), we do not see as clear visual improvements as in the previous results but rather small changes. Both models produce images that are unrecognizable to a large degree, which make the comparison difficult. This observation strengthens our intuition that averaging brings the cycling generator closer to the local optimal point, but it does not necessarily find a good solution as the local optimum may not be good enough. It might be also that the training is not converged yet to a stable region in parameter space, so the averaging does not work well.

Fig. 4.20 shows IS and FID scores during the training similar to the previous CIFAR-10 figures (see Fig. 4.4 to Fig. 4.9). Similar to the CIFAR-10 figures, EMA improves the performance consistently during the all training. Whereas, MA improves it during the initial phases of the training and get worse as the number of iterations increases.

FIGURE 4.18: Generation for STL-10 dataset after 500k iteration. (Left): w/o averaging, (Right): with EMA

FIGURE 4.19: Generation for ImageNet dataset after 500k iteration. (Left): w/o averaging, (Right): with EMA

4.5 Ablation Studies

In this section, we have conducted two types of ablation studies. The first one is investigating parameterized version of MA. Our motivation to apply this approach is to see whether parameterized version of MA can reach the same level performance of EMA. The second one is the effect of different β values for EMA. In the previous section, we have used $\beta = 0.9999$ for all quantitative scores, except the illustrative

FIGURE 4.20: ImageNet FID and IS score during training. Setting: original GAN objective/ conventional architecture/ $n_{dis} = 1$

examples, as it leads to the best performance on various settings. However, in this section we report the performance of the various values for β .

4.5.1 Parameterized Online Averaging

As an additional method to basic MA, we have used Parameterized Moving Average (PMA):

$$\theta_{PMA}^{(t)} = \frac{t - \alpha}{t} \theta_{PMA}^{(t-1)} + \frac{\alpha}{t} \theta^{(t)} \quad (4.10)$$

where α regulates the importance of the terms. When $\alpha = 1$, the equation corresponds to moving average (Eq. 4.1). In order to restrict range of $\frac{t-\alpha}{t}$ to positive values, $t \geq \alpha$ is ensured. For the initial iterations, the operation is not applied till $t \geq \alpha$ is satisfied. Unlike the MA experiments, we have not waited to apply the operation till $100k$, but as soon as $t \geq \alpha$ is satisfied. As PMA weights the recent history more for $t > 1$, its behaviour is expected to be closer to EMA for a well tuned α .

We have experimented (see Table 4.10) with PMA on CIFAR-10 with the same setting of the first row of Table 4.9. $\alpha = 1$ has failed as it started averaging from the first iteration which aligns with our earlier observations about necessity to start MA from later iterations. As α is increased from 1 to 10000, we see a significant increase, then decrease in the performance. However, none of the settings of PMA has reached the same level of performance of EMA's.

TABLE 4.10: IS and FID on CIFAR-10 (the same setting of the first row of Table 4.9) with various α values for PMA.

	IS	FID
nA	8.02 ± 0.20	17.32 ± 0.53
$\alpha = 1$	failed	failed
$\alpha = 10$	8.58 ± 0.06	14.55 ± 1.01
$\alpha = 100$	8.48 ± 0.06	13.85 ± 0.51
$\alpha = 1000$	8.28 ± 0.04	14.64 ± 0.51
$\alpha = 10000$	8.11 ± 0.13	15.60 ± 0.59

4.5.2 Quantitative Effects of Different β for EMA

In our main experiments, we have used $\beta = 0.9999$ except for illustrative examples which uses $\beta = 0.999$. Only for the CelebA, we have showed the qualitative effect of various beta values. In this section we show the effect of β to quantitative scores. We have experimented with different β values ranging from 0.9 to 0.9999 on CIFAR-10 with the same setting of the first row of Table 4.9. We have run the experiment four times and provide the mean and standard deviation in Table 4.11. Each experiment runs for 500k. IS, FID score are collected with 20k iteration interval and best scores are published. From Table 4.11, we can see clear and progressive improvement when β increases from 0.9 to 0.9999 consistently in both IS and FID. It is possible that higher β values reach even higher scores, however these experiments will take longer to conduct, go beyond the time limit for us.

TABLE 4.11: IS and FID on CIFAR-10 (the same setting of the first row of Table 4.9) with various β values for EMA.

	IS	FID
nA	8.09 ± 0.07	18.14 ± 1.08
EMA $\beta = 0.9$	8.19 ± 0.09	17.82 ± 0.43
EMA $\beta = 0.99$	8.32 ± 0.03	15.64 ± 0.58
EMA $\beta = 0.999$	8.69 ± 0.11	13.77 ± 0.38
EMA $\beta = 0.9999$	9.05 ± 0.09	12.91 ± 0.49

4.6 Application to Image In-painting

In this section, we show the application of EMA into a widely used computer vision problem, namely image in-painting. As baseline we have selected recently developed work of Yang et al. [3] which can generate diverse set of in-paintings for a single input. We have implemented this task on TensorFlow by following the the official implementation in PyTorch³ as close as possible. For EMA, we have used $\text{beta} = 0.9999$.

Fig. 4.21 shows the result of our experiments. The first row shows incomplete images, while the second and third rows are in-painted images without EMA and with EMA, respectively. It can be observed that, in general, the averaged in-painting produces samples with less artifacts and looks more realistic. In general, images look smoother. This observation closely follows our previous results; averaging tends to reduce artifacts.

FIGURE 4.21: Image in-painting with Yang et al. [3] on CelebA 128×128 (Top): Incomplete images, (Middle): In-painted images without EMA, (Bottom): In-painted images with EMA. Averaged images tends to be smoother, more realistic and has less artifact.

³<https://github.com/maga33/DSGAN>

4.7 Summary

We have explored the effect of two different techniques for averaging parameters outside of the GAN training loop, moving average (MA) and exponential moving average (EMA). We have shown that both techniques significantly improve the quality of generated images on various datasets, network architectures and GAN objectives. In the case of the EMA technique, we have provided the first theoretical analysis of its implications with discrete time bilinear objective and continuous time non-convex/concave objective. Our theoretical results show that even in simple bilinear settings EMA converges to stable limit cycles of small amplitude around the solution of the saddle problem. Besides, we have applied EMA into image in-painting problem which results in smoother, more realistic images with less artifacts.

Averaging methods are easy to implement, have minimal computation overhead and single hyperparameter. Also, hyperparameter for EMA search can be done with a single training run by simultaneously keeping track of multiple averages with different β parameters. As a result, these techniques are readily applicable in a wide range of settings.

Chapter 5

An Empirical Analysis of Overfitting, Mode Drop and Convergence in GANs Training

5.1 Introduction

In this chapter, we have tackled with overfitting/generalization, mode dropping, convergence of GANs training from empirical perspective at high dimensional data distributions. We hypothesized that all these phenomena are related with stochasticity in GANs training and supported our intuition with various experiments. We can show that specific GANs architectures and optimization methods can overfit to empirical distribution (training set). In this study, we define overfitting as the generator produces images nearly indistinguishable from the data samples and cover all of them¹. By the definition this generator does not show a mode drop behavior (full recall on empirical data distribution).

Our experiments provide an answer to whether the GANs objective and its variants are sufficient to match support of the data distribution as questioned by Arora et al.

---

¹This does not mean the generator can not generalize, as other samples from the prior or interpolations of seen points of latent space can be still mapped into novel samples.

[81]. It leads to a new understanding about how GANs generalize, why mode drop happens and where empirical lack of converge comes from.

Overfitting by removing stochasticity might seem straight forward due to common observation from maximum log-likelihood objectives, however GANs objective is a minimax (a saddle point objective). Also as the objective of the generator is not constant, due to the updates in the discriminator, it may not be the expected behavior. Indeed, Mescheder et al. [9] show that un-regularized GANs objectives fail to converge to correct solution even in a simple case like dirac-GAN.

Our contributions are empirically showing that:

- GANs can overfit to training set when stochasticity is removed from the training.
- When it overfits, by definition, it does not mode drop which goes against the intuition about relation between mode drop/collapse to GANs objective.
- Significant part of oscillation/non-convergence in GANs training comes from the stochasticity in the optimization rather than the objective.

5.2 Method

GANs are a two-player zero-sum game between a discriminator and generator:

$$\min_G \max_D \mathcal{L}(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] \quad (5.1)$$

It utilizes a discriminator to assess a pseudo-divergence between the true data distribution, $p_{data}(\mathbf{x})$, and the generator's distribution, $p_g(\mathbf{x})$. In practice, Eq. 5.2 is approximated with its empirical form:

$$\min_G \max_D \mathcal{L}(D, G) = \frac{1}{n} \sum_{i=1}^n \log D(\mathbf{x}_i) + \frac{1}{k} \sum_{i=1}^k \log(1 - D(G(\mathbf{z}_i))) \quad (5.2)$$

where $\mathbf{x}_i$ is i -th sample from a fixed set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ sampled from $p_{data}(\mathbf{x})$ and $\mathbf{z}_i$ is i -th sample from a fixed set $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ sampled from $p_z(\mathbf{z})$. In practice $k \gg n$, as $p_z(\mathbf{z})$ is readily available, while data samples are limited.

Besides, the approximation of the objective, its gradients are also approximated by using stochastic gradients with a mini-batch size of m and the parameters are updated as:

$$\phi^{(t+1)} = \phi^{(t)} + \frac{\alpha}{m} \sum_{i=1}^m \frac{\partial(\log D(\mathbf{x}_i) + \log(1 - D(G(\mathbf{z}_i))))}{\partial \phi^{(t)}} \quad (5.3)$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\alpha}{m} \sum_{i=1}^m \frac{\partial(\log(1 - D(G(\mathbf{z}_i))))}{\partial \theta^{(t)}} \quad (5.4)$$

where m samples are drawn from $\mathbf{X}$ with replacement and from $\mathbf{Z}$ without replacement², $\phi^{(t)}$ and $\theta^{(t)}$ are the parameters of D and G at t -th iteration, respectively.

There are two main types of stochasticity in this GANs training: (i) samples from the prior and (ii) stochastic gradient updates³. The former is considered as a necessary step to reduce sampling error, while the latter is used to reduce computation time by approximating the true gradient. To remove both types of stochasticity, we select $k = n$ and $m = n$. Under these settings, G generates exactly n samples during the training and D only distinguishes them from n samples from the empirical data distribution. As G only generates n samples, it easy to analyze pairwise distance from generated samples to data samples or vice versa.

After analyzing the deterministic setting, we relax the constraints to see its effect on overfitting/generalization, mode drop and convergence. The first relaxation is from the mini-batch size by reducing m to $\frac{n}{2^l}$, $l = 2, 4, 6, 8$. The second relaxation is introducing noise back into the input space of the generator. For the second type, we did not consider sampling more from the prior, which is the usual setting,

²As new samples are drawn from $p_z(\mathbf{z})$ at each iteration.

³We have not used any stochastic component in the architecture e.g. batchnorm.

as it imbalances real/fake samples ratios⁴. At each type of relaxation, we keep the other one deterministic to factorize out each ones influence.

One important question is whether fixing the samples from the prior changes GANs setting conceptually. Even though this is not a common practice in unconditional setting, it has been used by GANs based image-to-image transformation as their inputs are fixed set of samples [19, 27, 64, 65]. This shows our setting is still a GANs setting and the results are valid to understand the behavior of GANs.

5.2.1 Why to Remove Stochasticity?

Our intuition is that stochastic optimization leads the generator to leave some modes out and stall the fidelity of image generation. We postulate that this happens for the following reason. Initial in the training, the randomness from the optimizer, such as sampling error, is smaller than the estimated difference between the distributions which lead GANs training to progress on capturing the data distribution. However as training goes on, generator manifold gets closer to data manifold and stochasticity (sampling error) dominates the training which forces the discriminator to find superficial explanations for the difference between the distributions. This phenomenon halts the progress to capture true distribution, leads to low fidelity images, mode drop, and non-convergence over the parameters. Indeed, recent work of Chen et al. [114] show that performance of features extracted from the discriminator of GANs deteriorates for a downstream classification task after some time in the training. Even though, they relate it with non-stationary training nature of GANs objective, there are also other types of stochasticities that might explain the phenomenon. Also large batch size training of Brock et al. [1] show significant improvements in commonly used GANs scores. Besides these, the success of conditional GANs with labels as in Brock et al. [1], Odena et al.

---

⁴Another reason to not try this setting is that having more or less generated samples than data samples can lead to mode drop or mode collapse by its own. For example, if there are more generated samples than the training ones, then some of them will be surjective mapping (mode collapse).

TABLE 5.1: The generator architecture

Layers	Act.	Output Shape
Latent vector	-	512 x 1 x 1
Conv 4 x 4, st=1, pad=3	LReLU	1024 x 4 x 4
Upsample	-	1024 x 8 x 8
Conv 3 x 3, st=1, pad=1	LReLU	512 x 8 x 8
Upsample	-	512 x 16 x 16
Conv 3 x 3, st=1, pad=1	LReLU	256 x 16 x 16
Upsample	-	256 x 32 x 32
Conv 3 x 3, st=1, pad=1	Tanh	3 x 32 x 32

[12], Miyato and Koyama [13], Zhang et al. [115], Lucic et al. [116] can be also attributed to reducing the effect of sampling error by including more restriction to the training. We hypothesize that this behaviour can be alleviated by reducing or removing the stochasticity in the training. Even though non-convergence is related with theoretical properties of GANs objectives, we found that most of the empirical cycling can be eliminated by removing stochasticity in the training. We have quantified this behavior with various analysis.

5.2.2 Network Architectures

Before passing into the experiments, we briefly discuss the network architectures for the discriminator and the generator. The network architectures for the generator and the discriminator are as in Table 5.1 and Table 5.2, respectively. We have used a DCGAN Radford et al. [5] like architecture. *LReLU* is LeakyReLU with $\alpha = 0.2$. *Conv* $n \times n$ is convolution operation with $n \times n$ kernel size. *st* is the stride of the convolution operation, while *pad* is the amount of padding applied. In the discriminator we have used spectral normalization Miyato et al. [7] which improves the results considerably. These are the models used on the most part of the work. The lower capacity models (see Section 5.4.3) use convolutions with half of the width both in generator and discriminator.

TABLE 5.2: The discriminator architecture

Layers	Act.	Output Shape
Input image	-	3 x 32 x 32
Conv 5 x 5, st=2, pad=2	LReLU	128 x 16 x 16
Conv 5 x 5, st=2, pad=2	LReLU	256 x 8 x 8
Conv 5 x 5, st=2, pad=2	LReLU	512 x 4 x 4
Conv 5 x 5, st=1, pad=2	LReLU	1024 x 4 x 4
Conv 4 x 4, st=1, pad=0	Squeeze	1

5.3 Experiment Details

We have used 3 common datasets in GANs training which are SVHN [90], CIFAR-10 [91] and FFHQ [10]. We specifically selected these datasets to show our observations are not limited to gray-scale, low variation images but high variation, RGB images. FFHQ is scaled to 32×32 resolution to make the analysis possible with large batch size and sufficiently large data size. In all our experiments the data size is 12.8 k , the GANs objective is the non-saturating version [4], the update rule is alternating gradient descent, and update ratio of discriminator/generator is 1.

As optimizer ADAM [102] with $\alpha = 0.0001$, $\beta_1 = 0$ and $\beta_2 = 0.9$ is used. EMA over generator parameters is used for evaluation and visualization [8, 85]. $z \in \mathbb{R}^{512}$ are sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and fixed during the training.

As mentioned in Section 5.2, we included two types of randomness analysis in this work. Firstly, we show the effect of mini-batch size as the first type of stochasticity. Then by fixing the mini-batch size, we include noise to the fixed input samples of the generator as second type of stochasticity. For the latter approach, we replace $G(z)$ with $G(\hat{z})$ where $\hat{z} = z + \epsilon$, $\epsilon \sim \mathcal{N}(\mathbf{0}, 0.5\mathbf{I})$ during the training. This corresponds to using an equally weighted mixture of n Gaussians (GMM) with fixed mean and covariance. During the inference stage, when analyzing and visualizing the generator, $G(z)$ is used which corresponds to mean value of each Gaussian in GMM. This can

be also viewed as using most likely mode of truncation trick similar to Brock et al. [1], Karras et al. [10], Kingma and Dhariwal [117], Marchesi [118].

5.3.1 Quantitative Metrics

To quantify overfitting and mode drop we have utilized two types of metric: (i) pixel-wise loss in data space with ℓ_1 -norm and (ii) semantic loss in a latent space with ℓ_2 -norm. The reason for the second metric is to detect overfitting/ mode drop over semantic space. Indeed, it is possible to have a high pixel-wise score with low latent space loss as explained in Theis et al. [98]. Though, as we show in Section 5.4, overfitting, in our experiments, reaches a degree that can be easily captured with the pixel-wise score. Also, in Section 5.4 we see that latent space loss shows higher disturbance to noise. We think that this is mainly due to the low resolution setting of our experiments, where the evaluation network is trained on high resolution ImageNet dataset. Still, we used both scores for the evaluation.

With the above metrics, nearest neighbor algorithm (1-NN) is used to evaluate the distance to the first nearest neighbor and to visualize it. For overfitting this analysis is done by searching the nearest neighbor of each generated samples in all data samples. While for mode drop analysis it is done in the reverse order, namely searching the nearest neighbor of each data samples in all generated samples. The former can be interpreted as precision while the latter is recall. As finding the nearest sample is an expensive operation for large number of samples, $\mathcal{O}(n)$, we have approximated it 200 samples from the domain. For example, for the overfitting analysis, the nearest neighbor of 200 generated samples is searched in all data samples.

For the semantic loss, we have used InceptionV3 [99] architecture’s penultimate layer activations similar to FID [45]. For pixel-wise loss, images are scaled to $[0, 255]$ to be able to interpret the results easier. For convergence analysis, ℓ_1 -norm between generated images at t and $t + 1$ are evaluated by using the fixed set of latent code during the training.

The above metrics take average over all the samples which does not clearly demonstrates the worse case scenarios in that average. To include it, we have used the same metric on the worst 10%, 5% and 1% nearest neighbor (Table 5.3). This shows the samples with large distance to their nearest neighbors.

5.4 Results

In our early experimentation we have used full-batch setting that did not show large difference from 4 mini-batch setting ($3.2k$) in terms of performance, thus we stick with highest mini-batch size of $3.2k$ to approximate deterministic case. We have trained each case till convergence of our metrics.

We, first show the effect of the batch size, then the randomness in the latent space of the generator. Later, we also discuss the importance of the capacity in our results. In each subsection, to make our point, we start the analysis from the visual part which motivated this work. Our qualitative results can be best interpreted together with quantitative results as we can only show limited (but not cheery picked) number of samples.

5.4.1 Effect of the Batch Size

In Fig. 5.1, Fig. 5.2, and Fig. 5.3, larger batch sizes show more overfitting while smaller ones miss certain data points and produce more artifacts over the samples. Even though some of the artifacts are not easy to see, they influence the nearest neighbor in latent space as explained in Section 5.3.1. Similar to overfitting visuals, Fig. 5.4, Fig. 5.5, and Fig. 5.6 show mode drop behavior on various mini-batch size setting. The pattern closely resembles to the overfitting setting where larger batch sizes (less stochasticity) show less mode drop behaviour. In both analyses, SVHN is effected less from batch size change compared to CIFAR-10 and FFHQ. We believe

that this is due to the complexity of the dataset as CIFAR-10 and FFHQ include more diverse features compared to SVHN.

FIGURE 5.1: Analysis of overfitting on SVHN. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.

The quantification of the above results can be seen from Fig. 5.7 and Table 5.3. From overfitting and mode drop figures of Fig. 5.7, we can see that the two analyses are closely tied to one another e.g. similar patterns. This shows that modeling all parts of the distribution (no mode drop) is not sacrificed for image generation quality (overfitting). Another pattern is that convergence of overfitting and mode drop for smaller mini-batches are slower and higher in value; in some cases, Fig 5.7(f), it even gets worse. Additional to these figures, Table 5.3 contains information about worse case samples. As expected the scores get higher with lower percentages. Small percentage of the samples (5% - 1%) shows very high values in case of CIFAR-10 and FFHQ. Similar to the above figures, SVHN shows more robust scores when batch size changes compared to the other datasets. Interestingly, CIFAR10 seems easier to model than FFHQ which contains roughly the same face alignments.

FIGURE 5.2: Analysis of overfitting on CIFAR-10. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.

FIGURE 5.3: Analysis of overfitting on FFHQ. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.

FIGURE 5.4: Analysis of mode drop on SVHN. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.

FIGURE 5.5: Analysis of mode drop on CIFAR-10. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.

FIGURE 5.6: Analysis of mode drop on FFHQ. Mini-batch size of 3200/800/200/50 from top to bottom. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.

TABLE 5.3: The table of overfitting, mode drop and convergence with varying batch size on SVHN, CIFAR10 and FFHQ. All the results are evaluated with pixel-wise metric. The percentages (10% 5% 1%) shows the worst case scenarios as explained in Section 5.3.1.

Dataset	Mini-batch	Overfitting				Mode Drop				Convergence			
		Avg	10%	5%	1%	Avg	10%	5%	1%	Avg	10%	5%	1%
SVHN	3.2k	1.55	2.82	3.58	7.55	1.55	2.5	2.8	3.27	0.31	0.57	0.72	1.64
SVHN	800	3.05	4.87	5.55	7.86	3.26	6.35	7.88	11.91	1.94	3.91	4.69	6.58
SVHN	200	7.43	11.59	12.42	13.76	7.84	12.66	13.99	16.93	3.14	5.7	6.6	9.22
SVHN	50	9.06	16.63	18.6	22.24	9.36	20.8	25.98	38.04	1.83	3.73	4.51	6.85
CIFAR10	3.2k	4.90	7.40	8.59	14.82	4.88	6.90	7.39	8.50	1.85	3.65	4.50	8.65
CIFAR10	800	8.04	12.96	14.51	17.05	8.43	13.73	15.6	21.67	3.49	5.1	5.63	7.84
CIFAR10	200	24.47	37.01	38.87	42.01	24.71	40.44	43.36	47.42	4.03	6.98	7.77	9.91
CIFAR10	50	28.31	42.42	44.35	47.51	29.49	47.37	50.89	57.33	2.63	4.21	4.61	5.59
FFHQ	3.2k	6.39	9.71	11.81	25.35	6.57	12.55	17.41	41.00	2.53	4.14	4.73	7.17
FFHQ	800	10.7	16.2	18.96	29.6	10.72	17.64	21.74	36.22	3.43	4.93	5.49	8.01
FFHQ	200	27.81	38.78	40.78	44.11	28.38	39.57	41.59	46.1	4.39	7.00	7.71	9.67
FFHQ	50	32.84	43.79	45.19	49.16	32.57	44.37	46.27	50.5	2.49	3.43	3.64	4.09

When it comes to convergence of samples, the pattern does not exactly follow the above results. From the last row of Fig 5.7, we can see that higher batch sizes show better convergence except the batch size of 50. In case of FFHQ, $BS = 50$ catches convergence rate of $BS = 12.8 k$ (also see Table 5.3). This indicates that noisy

optimization procedure can find a stable point as good as, in terms of convergence, deterministic one.

FIGURE 5.7: Quantitative analysis of overfitting (top), mode drop (middle) and convergence (bottom) on SVHN (left), CIFAR10 (middle) and FFHQ (right) with pixel-wise metric for various mini-batch sizes.

A common analysis in GANs to show that the generator does not overfit but generalize is interpolation in the latent space of the generator. If the interpolations result in semantically smooth changes in data space, then it is interpreted as an indication of generalization [4, 5]. Fig. 5.8 shows scaled linear interpolation [119] for SVHN, CIFAR10 and FFHQ with batch size of 3200 and 50. In large batch size regime, non-smooth (noisy) interpolations, especially at the mid range of the interpolation, further support our observations with overfitting. While the interpolations of smaller batch size are generally smoother. This behavior can be best

observed from FFHQ case. It is interesting to see that the noise in the interpolations disappears as mini-batch size shrinks, even though they are not the seen points of the training. This observation further supports our hypothesis about the relation between overfitting/ generalization and stochasticity in the training.

Our interpretation is that, in case of large batch size, the generator utilizes its capacity to model fine details of seen samples of the training (the left-most and right-most columns of each subfigure in Fig. 5.8) while pushing interpolations out of data manifold. On the contrary, with smaller batch size, the interpolations are closer to the manifold at an expense of not faithfully generating the training samples (recall the overfitting results).

FIGURE 5.8: Illustration of scaled linear interpolation in $\text{code}(z)$ space of the generator. Upper row corresponds to SVHN (left), CIFAR10 (middle) and FFHQ (right) with mini-batch size of 3200, while the lower row is the same except the mini-batch size of 50.

5.4.2 Noisy Latent Code

After showing the effect of mini-batch size, now we show the effect of second type of stochasticity that is noisy latent code of the generator. We have designed an experiment where both models are the same, mini-batch size of $3.2k$, except one includes noise to its input (as explained in Section 5.3). They are trained for the

same amount of iterations, but the results are drastically different especially in CIFAR10 and FFHQ (see Fig 5.9, Fig 5.10, Fig 5.11 Fig 5.12, Fig 5.13, Fig 5.14 and Table 5.4).

Overfitting analysis of noisy input (Fig 5.9, Fig 5.10, Fig 5.11) can only show limited similarity between generated and training data (some ships and dogs in case of CIFAR10), while its without noise counterpart shows exact matching in each column. Similar behaviour can be seen from the mode drop analysis (Fig 5.12, Fig 5.13, and Fig 5.14). SVHN results seem to be robust against the change in both analyses, however a significant difference at quantitative analysis can be seen in Table 5.4. As both input types reach a good level for SVHN, it is not easy to see the difference from the qualitative analysis.

FIGURE 5.9: Overfitting analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on SVHN. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.

FIGURE 5.10: Overfitting analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on CIFAR-10. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.

FIGURE 5.11: Overfitting analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on FFHQ. For each subfigures, the first row is generated images, the second row is the closest data sample in pixel space and the third row is the closest data sample in latent space.

FIGURE 5.12: Mode drop analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on SVHN. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.

FIGURE 5.13: Mode drop analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on CIFAR-10. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.

The difference between the noisy and non-noisy setting can be seen even better from the qualitative analysis as in Table 5.4. There is a large gap between noisy and non-noisy setting, especially in CIFAR10 and FFHQ.

FIGURE 5.14: Mode drop analysis of without noise (upper row) and with noise (lower row) to the input space of the generator on FFHQ. For each subfigures, the first row is data samples, the second row is the closest generated sample in pixel space and the third row is the closest generated sample in latent space.

Comparing Table 5.4 and Table 5.3 shows that, noisy input influences overfitting and mode drop more than the noise coming from stochastic optimization due to batch-size (compared to the worse case of it i.e. $BS = 50$). More importantly image generation does not seem to reach a satisfactory level of fidelity, specifically in CIFAR10, when noise is included. We believe that this behaviour should be further studied to increase performance of GANs.

Other than quantifying the above results, Table 5.4 also illustrates convergence of samples where without noise models reaches lower levels of convergence. Unlike the first type of stochasticity, we did not observe the stochastic model to reach similar level of performance with the deterministic one. As suspected from the visual results, there is a significant gap between noisy input and its counterpart in all three metrics.

TABLE 5.4: The table of overfitting, mode drop and convergence on SVHN, CIFAR10 and FFHQ w/o and w/- noisy input. All the results are evaluated with pixel-wise metric. The percentages (10% 5% 1%) shows the worst case scenarios as explained in Section 5.3.1.

Dataset	Noisy Input	overfitting				Mode Drop				Convergence			
		Avg	10%	5%	1%	Avg	10%	5%	1%	Avg	10%	5%	1%
SVHN	True	6.09	10.71	11.85	13.99	7.15	18.07	22.76	30.84	0.77	1.58	2.12	4.52
SVHN	False	1.55	2.82	3.58	7.55	1.55	2.5	2.8	3.27	0.31	0.57	0.72	1.64
CIFAR10	True	34.39	49.57	51.41	54.66	32.16	48.25	51.31	56.03	5.23	9.03	9.85	11.56
CIFAR10	False	4.90	7.40	8.59	14.82	4.88	6.90	7.39	8.50	1.85	3.65	4.50	8.65
FFHQ	True	35.56	45.83	47.53	49.08	33.9	44.51	46.04	49.19	3.78	5.95	6.48	7.66
FFHQ	False	6.39	9.71	11.81	25.35	6.57	12.55	17.41	41.00	2.53	4.14	4.73	7.17

5.4.3 Importance of Capacity

It is important to note that capacity is a crucial part to achieve the results above. In order to fit into 12.8 k samples we have used an architecture for both generator and discriminator whose largest channel width is 1024 (see Section 5.2.2 for the details of the architecture). When channel width in each layers is halved, the model does not perform as good as the results above (see Fig 5.15). In each type of analysis, lower capacity model converges to higher values then its high capacity counterpart. Also the convergence time is longer for the low capacity model. Recall that we have used data size of 12.8 k ; with all samples from the from CIFAR10 (50 k samples), the results would be even worse for the both models. This means that GANs architectures like DCGAN Radford et al. [5] may not have enough capacity to fit into empirical distribution of CIFAR-10 or similar datasets. Significant gain of higher width (and depth) experiments of Brock et al. [1] also shows the same direction. It would be also interesting to investigate the architectures whose channel width is bigger than 1024, however with the hardware limitation we could not push it further.

5.5 Summary

We empirically show that given enough capacity GANs can overfit, show little to no mode drop and reduce non-convergence when stochasticity is removed from the training. Even though we did not suggest a new algorithm in this work, the results of the experiments show important implications such as: (i) Can GANs overfit? (ii) Is mode drop a direct consequence of GANs objective or it is more about the optimization? (iii) Is empirical non-convergence related more with non-convergence of GANs objective or with optimization type? (iv) Do GANs architectures like DCGAN have sufficient capacity to model CIFAR10 or similar empirical distributions with high fidelity? We believe that this empirical investigation will enhance the training of better GANs models.

FIGURE 5.15: Quantitative analysis of overfitting (top), mode drop (middle) and convergence (bottom) on CIFAR-10 with pixel-wise metric for mini-batch of $3.2k$ and two different setting for channel size (width).

Chapter 6

Autoregressive Generative Adversarial Networks

6.1 Introduction

Even though GANs show some successes recently at realistic face and bedroom images [5], its performance still does not scale well to large datasets with high sample variation. It can generate realistic textures, but the global consistency is still lacking [31]. Recently several variations of GAN have been proposed to improve image generation [55, 56, 120] with limited success and the issue of global inconsistency still persists.

Natural images contain a rich amount of features with specific co-occurrence configurations. A generative model should be able to learn these configurations in order to perform well. In a GAN setting, a discriminator is used to learn features in data and helps the generator to imitate them. The generator should be able to produce samples with certain feature configurations – which are seen as likely by the discriminator – to satisfy the discriminator’s criteria.

We postulate that the issue of global inconsistency has to do with the discriminator of GANs and especially its classification nature to model detailed global structure.

The discriminator architecture is similar to ones used in classification, whose aim is modeling $p(y|x)$ but not $p(x)$, where y , x , $p(x)$ represent class probabilities, data sample and data distribution, respectively. However the overall aim of generative modeling is learning $p(x)$ ¹. A good “teacher” a.k.a. discriminator, should be able to learn $p(x)$ in order to teach it to its “student” a.k.a. generator, but the discriminator only learns $p(x)$ implicitly by modeling $p(y|x)$ in an adversarial way. Maximum likelihood models such as Van Den Oord et al. [95] learn $p(x)$ implicitly, however direct modeling of $p(x)$ is time consuming and poses difficulties for capturing global structure in high resolution images.

In order to alleviate this problem, we propose a latent space modeling where an adversarial encoder learns the high dimensional feature vectors of real samples that are distinguishable from the fake ones. The distinguishability criterion is the modeling error of an autoregressive model that tries to model latent space distribution of real images $p(f)$ where f represents high-dimensional features. A good adversarial encoder should come up with features that produce low error if they belong to real images and high error if they belong to fake ones. Different from the original GAN, $p(y|x)$, discriminator, is not modeled but rather $p(f|x)$, the encoder, and $p(f)$, the autoregressive model.

Our proposed Autoregressive GAN (ARGAN) learns features in the real data distribution without classification but instead by modeling the latent space of the encoder with an autoregressive model. Since an autoregressive model factorizes latent space and learns features in a conditional way, it can model feature co-occurrences more effectively than a classifier as used in conventional GAN.

Furthermore, we propose a combination of ARGAN and Patch-GAN [19], which we call PARGAN. In this setting, Patch-GAN ensures realistic local features while ARGAN deploys its capacity at the global level. We show that PARGAN improves both qualitative and quantitative results of the generated images.

---

¹Goodfellow et al. [4] prove that under certain assumptions optimizing the GANs objective leads to learning $p(x)$, however the optimization is done in the function space. There is no proof that practical setting of GANs leads to $p(x)$.

6.2 Relation to Other Methods

In various GAN models, the discriminator can be partitioned into two parts, a feature learner (encoder) and an assigner, which assigns values to distinguish two distributions either real or synthetic. A vanilla GAN learns features by using an adversarial encoder (E) and a linear classifier (c) as assigner (Fig. 6.1). The Wasserstein GAN (WGAN) [55] follows a similar idea and uses an adversarial encoder with a linear separator (s) as assigner, which separates real features from generated ones on a single dimensional space. Energy-based GAN (EBGAN) [56] slightly departs from this trend and uses an encoder to learn features and a decoder (D) as assigner, which reconstructs input of encoder. In EBGAN, different from the previous methods, energy value is not the output of the assigner but the reconstruction error. In our proposed ARGAN model, features are learned with an encoder, and an autoregressive model (R) is used as assigner. Similar to EBGAN, energy value is not the assigner's output but rather its difference from the target, which corresponds to the modeling error of the assigner. For the generator all models are the same, a noise sample (z) is fed into a generator (G) that maps it into the data space.

FIGURE 6.1: A simplified comparison of GAN models including the proposed Autoregressive GAN.

Another type of generative model is proposed by Van Den Oord et al. [95] and Theis and Bethge [121], who model the joint distribution of pixels, $p(x)$, with an RNN in an autoregressive way. During test time samples are generated by sampling one pixel at a time from the model. In our work, autoregressive modeling is used in the latent space, and the features incoming to autoregressor are learned in

an adversarial manner. Hence instead of modeling data in the data space, they are modeled in the feature space, which is learned by an adversarial encoder concurrently. Modeling distribution in the latent space can focus on semantics of data rather than low level features like correlation between adjacent pixels. Furthermore, our model runs as fast as vanilla GAN during test time since autoregressive modeling is not a part of the generator.

6.3 Method

6.3.1 Preliminary

GAN is a two-player min-max game between a generator G and a discriminator D , as formulated by Eq. 6.1. G takes a noise sample z as input and maps it into data space as $G(z; \theta_g)$, while D tries to discriminate between samples coming from G and real samples x . Overall, D 's aim is to learn features that separate samples coming from p_g , generator's distribution, and samples coming from p_{data} , real data distribution. G 's aim is to produce samples that are indistinguishable from the real ones. In practice, as suggested by Goodfellow et al. [4], the objective for G is changed with Eq. 6.3 in order to improve gradient flow, especially during the initial part of the training. At each iteration, D maximizes Eq. 6.2 while G maximizes Eq. 6.3. This procedure continues until it reaches the equilibrium $p_g \approx p_{data}$, which is a saddle point of Eq. 6.1.

$$\min_G \max_D E_{x \sim p_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (6.1)$$

$$\max_D E_{x \sim p_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (6.2)$$

$$\max_G E_{z \sim p_z(z)} [\log(D(G(z)))] \quad (6.3)$$

6.3.2 Autoregressive GAN (ARGAN)

The discriminator of GAN can be separated into two parts: $D = d(E)$ where E is an adversarial encoder, and d is a linear binary classifier. The objective of E is to produce linearly separable distributions of features $f_{real} \sim E(x)$ and $f_{fake} \sim E(G(z))$. Instead of using a classifier d to separate distributions, we propose to use an autoregressive model, R , to model the latent distribution of real samples $p(f_{real})$. This model should produce a small error for samples coming from $p(f_{real})$ and a large error for samples coming from $p(f_{fake})$ so that it can differentiate one distribution from the other.

FIGURE 6.2: Proposed Autoregressive GAN (ARGAN) architecture.

An overview of the proposed ARGAN architecture is depicted in Fig. 6.2. When compared to vanilla GAN (Fig. 6.1), this model replaces the classifier layer (d) with a factorization of features and autoregressive modeling of the factorized features. The autoregressive model can be represented with a multi-layer LSTM or other types of RNN models. Instead of using class discrimination as the learning signal, the autoregressive model's prediction loss is used. The intuition is that an autoregressive model would model the feature distribution better than fully connected layers by factorizing f and modeling conditional distributions:

$$p(f) = p(f_1, \dots, f_N) = \prod_{n=1}^N p(f_n | f_{<n}). \quad (6.4)$$

The proposed Autoregressive GAN (ARGAN) is formulated as in Eq. 6.5, where G is the generator function, E is the encoder, $\mathcal{L}$ is the loss function as defined in Eq. 6.6, and $\theta_G, \theta_E, \theta_R$ are the parameters for generator, encoder, and autoregressive model, respectively.

$$\min_{\theta_G} \max_{\theta_E, \theta_R} E_{x \sim p_{data}(x)} [\mathcal{L}(E(x; \theta_E), \theta_R)] - E_{z \sim p_z} [\mathcal{L}(E(G(z; \theta_G); \theta_E); \theta_R)] \quad (6.5)$$

$$\mathcal{L}(f; \theta_R) = \sum_{i=1}^{N_f} \log p_{\theta_R}(f_i | f_{<i}) \quad (6.6)$$

In practice, we found that Eq. 6.5 is unstable due to θ_R 's maximization of the second term being unbounded². Instead, θ_R is optimized only over the first term of Eq. 6.5. As we show in Section 6.4, this approach works well in practice. We postulate that it works because E is still adversarial which forces f_{real} and f_{fake} to be different. In this case, the objective for G and E stays the same (Eq. 6.7), while θ_R is optimized only on real sample features that are coming from $E(x)$ but not on $E(G(z))$ (Eq. 6.8):

$$\min_{\theta_G} \max_{\theta_E} E_{x \sim p_{data}(x)} [\mathcal{L}(E(x; \theta_E); \theta_R)] - E_{z \sim p_z} [\mathcal{L}(E(G(z; \theta_G); \theta_E); \theta_R)] \quad (6.7)$$

$$\max_{\theta_R} E_{x \sim p_{data}(x)} [\mathcal{L}(E(x; \theta_E); \theta_R)] \quad (6.8)$$

To explain the above objective intuitively, when θ_E maximizes Eq. 6.7, it tries to learn features that yield low error for real samples and high error for generated ones. In this way, E exploits structures that are common on data samples but not on generated ones. When θ_R maximizes Eq. 6.8, it fits to features coming from real examples only to reduce the error of autoregressive modeling on real samples. Algorithm 1 provides a pseudo code for ARGAN training procedure.

---

²This is the same issue encountered in EBGAN. In their case, a hinge type loss is proposed to limit the bounds. We follow a different approach.

Algorithm 1 ARGAN Training Procedure

```

1:  $\theta_G, \theta_E, \theta_R \leftarrow$  initialize network parameters
2: repeat
3:    $x^{(1)}, \dots, x^{(M)} \sim p_{data}(x)$   $\triangleright$  Sample from the data distribution
4:    $z^{(1)}, \dots, z^{(M)} \sim p_{noise}(z)$   $\triangleright$  Sample from the noise distribution
5:    $f_r^{(j)} \leftarrow E(x^{(j)})$ , for  $j = 1, \dots, M$   $\triangleright$  Compute features for the real samples
6:    $f_f^{(j)} \leftarrow E(G(z^{(j)}))$ , for  $j = 1, \dots, M$   $\triangleright$  Compute features for the generated
     samples
7:    $p_r^{(j)} \leftarrow R(f_r^{(j)})$ , for  $j = 1, \dots, M$   $\triangleright$  Compute predictions for the real samples
8:    $p_f^{(j)} \leftarrow R(f_f^{(j)})$ , for  $j = 1, \dots, M$   $\triangleright$  Compute predictions the generated
     samples
9:    $\mathcal{L}_{\theta_G} \leftarrow \frac{1}{M} \sum_{j=1}^M \sum_{i=1}^{N_f} |p_{f_i}^{(j)} - f_{f_{i+1}}^{(j)}|$   $\triangleright$  Compute loss for the generator
10:   $\mathcal{L}_{\theta_E} \leftarrow \frac{1}{M} \sum_{j=1}^M \sum_{i=1}^{N_f} |p_{r_i}^{(j)} - f_{r_{i+1}}^{(j)}| - \frac{1}{M} \sum_{j=1}^M \sum_{i=1}^{N_f} |p_{f_i}^{(j)} - f_{f_{i+1}}^{(j)}|$   $\triangleright$  Compute
     loss for the encoder
11:   $\mathcal{L}_{\theta_R} \leftarrow \frac{1}{M} \sum_{j=1}^M \sum_{i=1}^{N_f} |p_{r_i}^{(j)} - f_{r_{i+1}}^{(j)}|$   $\triangleright$  Compute loss for the autoregressive
     model
12:   $\theta_G \leftarrow \theta_G - \nabla_{\theta_G} \mathcal{L}_{\theta_G}$   $\triangleright$  Update parameters of the generator
13:   $\theta_E \leftarrow \theta_E - \nabla_{\theta_E} \mathcal{L}_{\theta_E}$   $\triangleright$  Update parameters of the encoder
14:   $\theta_R \leftarrow \theta_R - \nabla_{\theta_R} \mathcal{L}_{\theta_R}$   $\triangleright$  Update parameters of the autoregressive model
15: until convergence

```

In practice, we have not used any distribution function (such as Gaussian, GMM etc.) at the output layer of R since it is not meant to be sampled but rather used during training as an auxiliary model. Also, using the model only during training removes test time sampling error [122], which happens when conditional samples come from the model distribution rather than the data distribution. This makes it robust against accumulation of error during the inference unlike Van Den Oord et al. [95] and Theis and Bethge [121].

As shown in Fig. 6.2 and formulated in Eq. 6.6, our model includes a factorization step. If we choose to factorize every single feature, the computation of R would be expensive as it is recurrent. In order to reduce computation time during training, we make independence assumption when factorizing the features. Instead of conditioning every single feature on another, feature groups are conditioned on other feature groups by making the assumption that features in the same groups are independent from each other. Fig. 6.3 shows two types of factorization, which are namely spatial-wise and channel-wise factorization. We call the models with

FIGURE 6.3: Two different types of factorization for our proposed model. The blue tensor block represents activation map after a convolution operation with size $H \times W \times C$. (a) Spatial Factorization: $2 \times 2 \times 4$ feature tensor factorized into $1 \times 1 \times 4$ tensors which corresponds to factorization along spatial dimensions, (b) Channel-wise Factorization: $2 \times 2 \times 4$ feature tensor factorized into $2 \times 2 \times 1$ tensors which corresponds to factorization along depth (or channel) dimensions.

these factorizations as Spatial-wise ARGAN (S-ARGAN) and Channel-wise ARGAN (C-ARGAN). Even though the independence assumption does not strictly hold, in practice both of them work equally well.

6.3.3 Patch-GAN + S-ARGAN (PARGAN)

Our proposed methods reach good results during practice (see Section 6.4). However in order to boost the performance further, we propose to include Patch-GAN [19, 22] into our modeling. Unlike vanilla GAN which produces single value for each sample, Patch-GAN takes multiple patches from a sample and produces a score for each patch. Hence for a single sample, multiple scores from different regions are evaluated. Even though Patch-GAN can help the generator to produce locally realistic samples, it does not take the global arrangement into account. Isola et al. [19] use Patch-GAN combined with L1 loss for image-to-image translation. While Patch-GAN ensures realism of local content, a pixel-wise loss with ground truth is used to enforce global arrangement.

For image generation as in our work, there is no ground truth to relate with $G(z)$, hence it is not clear how to enforce global arrangement. Therefore, we propose a

combination of Patch-GAN and S-ARGAN to ensure global consistency. In this case, Patch-GAN decides whether individual patches are realistic or not, while S-ARGAN examines whether their arrangement is spatially consistent. Note that S-ARGAN by itself can give feedback into individual channels to ensure the individual patches are realistic, however using it together with Patch-GAN makes training easier and produces better results. Fig. 6.4 shows how the latent space of the encoder is factorized and feeds into S-ARGAN and Patch-GAN.

FIGURE 6.4: Left: A feature tensor factorized in spatial fashion. Right (top): An autoregressive model on factorized features. Right (bottom): Patch-GAN: a single discriminator is used for each spatial tensor to ensure local realism.

The objective of the Patch-GAN for discriminator and generator is given in Eq. 6.9 and Eq. 6.10 where $D = d \oplus E$, $\oplus$ is convolution operation, $E(.) \in \mathbb{R}^{h \times w \times c}$ is the same encoder used in previous section and $d \in \mathbb{R}^{c \times 1 \times 1 \times 1}$ is the convolutional kernel. $D(.)$ produces an activation map with size $\mathbb{R}^{h \times w}$. Each activation is matched to its corresponding target by using least square loss. We use the least square version of GAN objective [120] rather than vanilla GAN objective (Eq. 6.2, Eq. 6.3) since vanilla GAN produces high error for generator later in the training.

$$\begin{aligned} \max_D E_{x \sim p_{data}(x)} & \left[\frac{1}{hw} \sum_h \sum_w (D(x)_{h,w} - 1)^2 \right] + \\ & E_{z \sim p_z(z)} \left[\frac{1}{hw} \sum_h \sum_w (D(G(z))_{h,w})^2 \right] \end{aligned} \quad (6.9)$$

$$\max_G E_{z \sim p_z(z)} \left[\frac{1}{hw} \sum_h \sum_w (D(G(z))_{h,w} - 1)^2 \right] \quad (6.10)$$

PARGAN objective is a simple summation of ARGAN objective (Eq. 6.7, Eq. 6.8) and Patch-GAN objective (Eq. 6.9, Eq. 6.10) without any hyperparameter.

6.3.4 Network Architectures

In the previous section, we have discussed objective and overall structure of our proposed model without providing details about the network architectures G , E , R , and d for CIFAR-10, STL-10, and CelebA. The network architecture for the generator and the discriminator (E , R , and d) for CIFAR-10, STL-10, and CelebA are as in Table 6.1, Table 6.2, Table 6.3, Table 6.4, Table 6.5, and Table 6.6. For the encoder and generator, we have used architectures similar to DCGAN [5]. $ReLU$ is ReLU function and $LReLU$ is LeakyReLU with $\alpha = 0.2$. BN is batch-normalization [123]. *reshape* is reshaping the matrix into a tensor in order to make it a proper input for convolution operator. $FC(I_c, O_c)$ is fully connected layer with I_c input channel, and O_c output channel. $Conv(I_c, O_c) n \times n$ is convolution operation with $n \times n$ kernel size, I_c input channel, and O_c output channel. st is the stride of the convolution operation, while pad is the amount of padding applied. $De-Conv(I_c, O_c) n \times n$ is de-convolution operation with $n \times n$ kernel size, I_c input channel, and O_c output channel. In this case, st is fractional stride of the de-convolution operation, while pad is the amount of padding applied similar to convolution operation. LSTM is long short-term memory [124] with multiple layers and hidden unit side specified in the tables. In each table, output shape is provided to make the architecture easier to understand.

TABLE 6.1: The generator architecture for CIFAR-10

Layers	Act.	Output Shape
Latent vector	-	100
FC(100, 8192) & reshape	BN-ReLU	512 x 4 x 4
De-Conv(512, 256) 4 x 4, st=2	BN-ReLU	256 x 8 x 8
De-Conv(256, 128) 4 x 4, st=2	BN-ReLU	128 x 16 x 16
De-Conv(128, 3) 4 x 4, st=2	Tanh	3 x 32 x 32

TABLE 6.2: The discriminator (E , R , and d) architecture for CIFAR-10

Layers	Act.	Output Shape
Input image	-	3 x 32 x 32
Conv(3, 64) 4 x 4, st=2	BN-LReLU	64 x 16 x 16
Conv(64, 128) 4 x 4, st=2	BN-LReLU	128 x 8 x 8
Conv(128, 256) 4 x 4, st=2	BN-LReLU	256 x 4 x 4
Conv(256, 512) 3 x 3, st=1, pad=1	BN-LReLU	512 x 4 x 4
R for C-ARGAN: 2 layer LSTM, hidden size:128	-	512 x 4 x 4
R for S-ARGAN: 1 layer LSTM, hidden size:512	-	512 x 4 x 4
R for PARGAN: 1 layer LSTM, hidden size:512	-	512 x 4 x 4
d for PARGAN: Conv(512,1) 1 x 1, st=1, pad=0	-	1 x 4 x 4

TABLE 6.3: The generator architecture for STL-10

Layers	Act.	Output Shape
Latent vector	-	100
FC(100, 4608) & reshape	BN-ReLU	512 x 3 x 3
De-Conv(512, 256) 4 x 4, st=2	BN-ReLU	256 x 6 x 6
De-Conv(256, 256) 4 x 4, st=2	BN-ReLU	256 x 12 x 12
De-Conv(256, 128) 4 x 4, st=2	BN-ReLU	128 x 24 x 24
De-Conv(128, 3) 4 x 4, st=2	Tanh	3 x 48 x 48

TABLE 6.4: The discriminator (E , R , and d) architecture for STL-10

Layers	Act.	Output Shape
Input image	-	3 x 48 x 48
Conv(3, 64) 4 x 4, st=2	BN-LReLU	64 x 24 x 24
Conv(64, 128) 4 x 4, st=2	BN-LReLU	128 x 12 x 12
Conv(128, 256) 4 x 4, st=2	BN-LReLU	256 x 6 x 6
Conv(256, 512) 3 x 3, st=1, pad=1	BN-LReLU	512 x 3 x 3
R for C-ARGAN: 2 layer LSTM, hidden size:128	-	512 x 3 x 3
R for S-ARGAN: 1 layer LSTM, hidden size:512	-	512 x 3 x 3
R for PARGAN: 1 layer LSTM, hidden size:512	-	512 x 3 x 3
d for PARGAN: Conv(512,1) 1 x 1, st=1, pad=0	-	1 x 3 x 3

TABLE 6.5: The generator architecture for CelebA

Layers	Act.	Output Shape
Latent vector	-	100
FC(100, 8192) & reshape	BN-ReLU	512 x 4 x 4
De-Conv(512, 256) 4 x 4, st=2	BN-ReLU	256 x 8 x 8
De-Conv(256, 256) 4 x 4, st=2	BN-ReLU	256 x 16 x 16
De-Conv(256, 128) 4 x 4, st=2	BN-ReLU	128 x 32 x 32
De-Conv(128, 3) 4 x 4, st=2	Tanh	3 x 64 x 64

TABLE 6.6: The discriminator (E , R , and d) architecture for CelebA

Layers	Act.	Output Shape
Input image	-	3 x 64 x 64
Conv(3, 64) 4 x 4, st=2	BN-LReLU	64 x 32 x 32
Conv(64, 128) 4 x 4, st=2	BN-LReLU	128 x 16 x 16
Conv(128, 256) 4 x 4, st=2	BN-LReLU	256 x 8 x 8
Conv(256, 512) 3 x 3, st=1, pad=1	BN-LReLU	512 x 4 x 4
R for C-ARGAN: 2 layer LSTM, hidden size:128	-	512 x 4 x 4
R for S-ARGAN: 1 layer LSTM, hidden size:512	-	512 x 4 x 4
R for PARGAN: 1 layer LSTM, hidden size:512	-	512 x 4 x 4
d for PARGAN: Conv(512,1) 1 x 1, st=1, pad=0	-	1 x 4 x 4

6.4 Experiments

We have conducted experiments on CIFAR-10 [91], STL-10 [92] and CelebA [93] datasets. CIFAR-10 has 60,000 images of 32×32 pixels with 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. We have used 50k training set images to train our model. STL-10 is a subset of ImageNet and more complex than CIFAR-10. There are 13k labeled and 100k unlabeled images with 96×96 pixels resolution. CelebA contains around 200k aligned face images. The images are cropped to roughly leave face area. All of our experiments have been conducted without any labels (unconditional modeling).

There is no agreed-upon evaluation metric for generative models [98]. One commonly used metric is the Inception Score (IS) [31] which is well correlated with

human judgment. Inception score is $\exp(E_x[KL(p(y|x)||p(y))])$, where $p(y|x)$ is the conditional class probability which should ideally have low entropy, and $p(y)$ is the marginal class probability which should ideally have high entropy. Higher values of inception score are better. For evaluation, 50k images are generated and fed into the official Inception score code³ with 10 splits.⁴

Detailed network architecture is given in Section 6.3.4. We have used transposed convolution with strides 2 in generator until it matches the size of the data. In the discriminator, convolutions with stride 2 are used until it reaches a pre-defined feature map size. BatchNorm [123] is used in both generator and discriminator, except the last layer of generator. When C-ARGAN is used, the channel number in the last layer in E restricted to 256 in order to reduce computation time during training. In the case of S-ARGAN and PARGAN, the channel number in the last layer is 512. We have used single layer LSTM for S-ARGAN, and multi-layer LSTM for C-ARGAN since single level produces blurry results. More layers of LSTM might perform better but we leave it to the feature study due to computational constraints. ADAM [102] is used as optimizer with learning rate 0.0001 and $\beta_1 = 0.5$.

Interestingly, L2-norm loss could not produce results as good as L1-norm loss in R 's objective (Eq. 6.6). Hence, we have conducted all our experiments using L1-norm loss in the autoregressive model. In the case of C-ARGAN and S-ARGAN, the generator is updated t times to catch encoder. We found t to be an important hyper-parameter to tune. By using grid search, we choose $t = 3$ in all our experiments except for PARGAN which uses $t = 1$. With this setting, the error of the generator closely follows the error of the encoder and the training is stable.

We first present our qualitative results, then present the quantitative ones. The qualitative results are best judged together with quantitative ones. For quantitative

³ https://github.com/openai/improved-gan/tree/master/inception_score

⁴ In other papers information about which model is used when reporting the score is often missing. For example, one can run the same model several times and publish the best score. In order to produce unbiased estimates, we have run our model and collected the score more than 10 times when there is no perceptual improvement over generated images. We published the averages of the scores.

results, we also compare our proposed method with other state-of-the-art GANs methods.

6.4.1 Qualitative Results

Generated images for the proposed S-ARGAN, C-ARGAN and PARGAN for CIFAR-10 are shown in Fig. 6.5. Note that none of the images in this chapter are cherry picked. Overall images look sharp even though not all of them feature recognizable objects. We see slightly better image generation with less artifacts for PARGAN compared to S-ARGAN and C-ARGAN which correlates with our quantitative results. Fig. 6.6 shows 48x48 pixel image generation from STL-10 dataset for S-ARGAN, C-ARGAN and PARGAN. We have also used PARGAN on the 96x96 STL-10 to show it can scale to higher resolutions (Fig. 6.7). Certain objects like car, ship, horse, and some other animals are recognizable, but it is still hard to interpret most of the images. For CelebA generation we have used S-ARGAN with 64x64 resolution (Fig. 6.8). The face images look realistic in general with small artifacts in certain samples.

FIGURE 6.5: Generation for CIFAR-10 dataset. (Left): S-ARGAN. (Middle): C-ARGAN. (Right): PARGAN

6.4.2 Quantitative Results

Inception scores for the proposed S-ARGAN, C-ARGAN and PARGAN on CIFAR-10 and STL-10 are given in Table 6.7. We compare our results with other baseline

FIGURE 6.6: Generation for STL-10 dataset with 48x48 resolution. (Left): S-ARGAN. (Middle): C-ARGAN. (Right): PARGAN

FIGURE 6.7: Generation for STL-10 dataset with 96x96 resolution and PARGAN modeling.

FIGURE 6.8: Generation for CelebA dataset with 64x64 resolution and S-ARGAN modeling.

GAN models⁵ like DCGAN [5], EBGAN [56], WGAN [55] rather than extensions such as D2GAN [125], MGGAN [57], DFM [87] etc. These extensions can be combined with our model in the same way as they are combined with vanilla GAN.

The first row of Table 6.7 shows Inception score for real data samples, which is

⁵To be clear, each baseline GAN optimizes different divergence, for example basic GAN, WGAN, EBGAN, optimizes Jensen-Shannon, Wasserstein and total variance respectively. While extension does not change divergence but changes model architecture or include some regularization etc. Our method is compatible with the extensions, which we leave as future work.

11.24 and 26.08 for CIFAR-10 and STL-10, respectively. Our proposed models, C-ARGAN and S-ARGAN show competitive results with DCGAN, while PARGAN outperforms them. C-ARGAN and S-ARGAN fall slightly short against EBGAN and WGAN-GP on the same architecture (DCGAN), while PARGAN slightly outperforms them. Our methods can be further improved by utilizing Gradient Penalty (GP) regularization as it is done in WGAN-GP.

TABLE 6.7: Inception scores for our proposed models and state-of-the-art baselines on CIFAR-10 and STL-10.

Model	CIFAR-10	STL-10
Real data	11.24 ± 0.16	26.08 ± 0.26
DCGAN [5]	6.40 ± 0.05	7.54
EBGAN [56]	6.74 ± 0.09	-
WGAN-GP (ResNet) [6]	7.86 ± 0.07	9.05 ± 0.12
WGAN-GP (DCGAN) (Our implementation)	6.80	-
S-ARGAN (Proposed)	6.50	7.44
C-ARGAN (Proposed)	6.46	7.60
PARGAN (Proposed)	6.86	7.89

6.5 Summary

We have proposed a novel GAN architecture and objective and successfully trained it on CIFAR-10, STL-10, and CelebA datasets. It models the latent distribution of real samples to learn feature co-occurrences implicitly. We propose two variants to our model by utilizing efficient factorization, namely Spatial-wise ARGAN and Channel-wise ARGAN. Also, we have combined our proposed Spatial-wise ARGAN with Patch-GAN to further improve the stability of training and the quality of image generation. Our results show that the proposed model outperforms DCGAN, EBGAN, and WGAN on the same architecture.

Chapter 7

Conclusion and Future Work

In this thesis, we have proposed novel algorithms, architectures and provided theoretical/empirical analyses to mitigate various problems of GANs. The main challenges we tackle here are improving high dimensional sample generation, non-convergence of the objective function, overfitting/ generalization, mode drop (not modeling certain parts of the distribution), and lack of efficient multi-distribution modeling. Below we conclude this thesis with brief summary of our works and future works we plan to pursue.

In Chapter 3, we have proposed a novel multi-distribution GAN method which can discovers particularities and commonalities of the distributions. Our method models each data distribution with a mixture of generator distributions. As the generators are partially shared between the modeling of different true data distributions, shared ones capture the commonality of the distributions, while non-shared ones capture unique aspects of them. We have successfully trained it on various datasets and show its effectiveness. Additionally, we have illustrated how this method can be used to incorporate external (prior) knowledge to distribution modeling for better generative modeling. We believe this method has good potential for new applications and better data modeling. In the future, we would like to improve our method by learning mixture of the weights, $\mathbf{O}$, end-to-end with the model parameters.

In Chapter 4, we have explored the effect of two different techniques for averaging parameters outside of the GAN training loop, moving average (MA) and exponential moving average (EMA). We have shown that both techniques significantly improve the quality of generated images on various datasets, network architectures and GAN objectives. In the case of the EMA technique, we have provided the first theoretical analysis of its implications, showing that EMA shrinks the cycles around the equilibrium for bilinear games and also enhances the stability of general GANs training. Besides, we have applied EMA into image in-painting problem which results in smoother, more realistic images with less artifacts. Averaging methods are easy to implement and have minimal computation overhead. As a result, these techniques are readily applicable in a wide range of settings. In the future, we plan to explore their effect on larger scales (e.g. ImageNet [94]) as well as on other GANs applications.

In Chapter 5, we have shown empirically that given enough capacity GANs can overfit, show little to no mode drop and reduce non-convergence when stochasticity is removed from the training. Our analysis answers important questions in GANs literature such as: Can GANs overfit to training set? Is mode drop direct consequence of GAN objective? Is empirical non-convergence related more with non-convergence of GAN objective or optimization type? Do GAN architectures like DCGAN have sufficient capacity to model CIFAR-10 or similar empirical distributions with high fidelity? Our experiments provide an answer to whether the GANs objective and its variants are sufficient to match support of the data distribution as questioned by Arora et al. [81]. It leads to a new understanding about how GANs generalize, why mode drop happens and where empirical lack of converge comes from. Due to computation and memory limitations we have restricted our study to 32×32 resolution and sample size of $12.8 k$. In the future, we would like to scale this study to larger resolutions, more samples and more complex distributions e.g. ImageNet [94].

In Chapter 6, we aim to improve the quality of image generation with GANs by refining the discriminator architecture and the objective. We have proposed a novel

GAN architecture & objective and successfully trained it on CIFAR-10, STL-10, and CelebA datasets. Different from original GANs setting, it models the latent distribution of real samples to learn feature co-occurrences implicitly. We propose two types of ARGAN model, namely Spatial-wise ARGAN and Channel-wise ARGAN. Also, we have combined our proposed Spatial-wise ARGAN with Patch-GAN to further improve the stability of training and the quality of image generation. Our results show that the proposed model outperforms DCGAN, EBGAN, and WGAN on the same architecture. In this work, we have used a simple autoregressive model. Our model can be further enhanced by utilizing more advanced latent space modeling, such as bidirectional modules [126], 2D LSTM [121], hierarchical modules [127] etc. As future work, we would like to utilize such advanced autoregressive models and extend this method to ImageNet [94] which has even larger sample variation. Combining our models with MGGAN [57] and utilizing gradient penalty in the objective [6] are also potential future directions.

List of Author's Publications

Conference Proceedings

- **Y. Yazici**, C.S. Foo, S. Winkler, K.H. Yap, V. Chandrasekhar, “An Empirical Analysis of Overfitting and Mode Drop in GAN Training,” in *27th IEEE International Conference on Image Processing, 2020*.
- **Y. Yazici**, C.S. Foo, S. Winkler, K.H. Yap, G. Piliouras, V. Chandrasekhar, “The Unusual Effectiveness of Averaging in GAN Training,” in *International Conference on Learning Representations, 2019*.
- **Y. Yazici**, S. Winkler, K.H. Yap, “Autoregressive Generative Adversarial Networks,” in *International Conference on Learning Representations (Workshop Track), 2018*.

Journal Articles

- **Y. Yazici**, B. Lecouat, C.S. Foo, S. Winkler, K.H. Yap, G. Piliouras, V. Chandrasekhar, “Venn GAN: Discovering Commonalities and Particularities of Multiple Distributions,” Under preparation for *IEEE Transactions on Neural Networks and Learning Systems, (TNNLS)*.

- **Y. Yazici**, C.S. Foo, S. Winkler, K.H. Yap, V. Chandrasekhar, “An Empirical Analysis of Overfitting, Mode Drop and Convergence in GAN Training,” Under preparation for *IEEE Transactions on Neural Networks and Learning Systems*, (*TNNLS*).

Other Publications

- V. Vonikakis, **Y. Yazici**, V.D. Nguyen, S. Winkler, “Group Happiness Assessment Using geometric Features and Dataset Balancing,” in *18th ACM International Conference on Multimodal Interaction*, 2016.

Bibliography

- [1] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=B1xsqj09Fm>. [xxi](#), [6](#), [7](#), [9](#), [18](#), [27](#), [90](#), [93](#), [104](#)
- [2] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017. [xxi](#), [7](#)
- [3] Dingdong Yang, Seunghoon Hong, Yunseok Jang, Tiangchen Zhao, and Honglak Lee. Diversity-sensitive conditional generative adversarial networks. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=rJliMh09F7>. [xxiii](#), [84](#)
- [4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014. URL <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>. [1](#), [2](#), [3](#), [6](#), [9](#), [10](#), [18](#), [20](#), [25](#), [27](#), [29](#), [32](#), [33](#), [57](#), [62](#), [65](#), [68](#), [92](#), [99](#), [108](#), [110](#)
- [5] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR*, abs/1511.06434, 2015. URL <https://arxiv.org/abs/1511.06434>. [6](#), [10](#), [12](#), [20](#), [27](#), [29](#), [38](#), [57](#), [67](#), [91](#), [99](#), [104](#), [107](#), [116](#), [121](#), [122](#)
- [6] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. In *Proc. 31st International Conference on Neural Information Processing Systems (NIPS)*, pages 5769–5779, December 2017. arxiv: 1704.00028. [10](#), [21](#), [57](#), [65](#), [68](#), [122](#), [125](#)
- [7] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=B1QRgziT->. [6](#), [38](#), [65](#), [91](#)

- [8] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=Hk99zCeAb>. 6, 9, 18, 27, 38, 65, 92
- [9] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine learning (ICML)*, 2018. 6, 8, 20, 29, 41, 44, 58, 62, 67, 88
- [10] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. URL <http://arxiv.org/abs/1812.04948>. 6, 21, 24, 92, 93
- [11] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. URL <http://arxiv.org/abs/1411.1784>. 6, 16, 53
- [12] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proc. 34th International Conference on Machine Learning*, volume 70 of *Proc. Machine Learning Research*, pages 2642–2651, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL <http://proceedings.mlr.press/v70/odena17a.html>. 6, 16, 17, 26, 91
- [13] Takeru Miyato and Masanori Koyama. cGANs with projection discriminator. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=ByS1VpgRZ>. 6, 16, 38, 91
- [14] Scott Reed, Zeynep Akata, Xincheng Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proc. 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1060–1069, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <http://proceedings.mlr.press/v48/reed16.html>. 6, 16
- [15] Ayushman Dash, John Cristian Borges Gamboa, Sheraz Ahmed, Muhammad Zeshan Afzal, and Marcus Liwicki. Tac-gan-text conditioned auxiliary classifier generative adversarial network. *arXiv preprint arXiv:1703.06412*, 2017.
- [16] Scott E Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 217–225. Curran Associates, Inc., 2016. URL <http://papers.nips.cc/paper/6111-learning-what-and-where-to-draw.pdf>.

- [17] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5908–5916, Oct 2017. doi: 10.1109/ICCV.2017.629.
- [18] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1947–1962, Aug 2019. ISSN 1939-3539. doi: 10.1109/TPAMI.2018.2856256. 6
- [19] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, pages 5967–5976. IEEE Computer Society, 2017. 7, 12, 16, 27, 90, 108, 114
- [20] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1857–1865, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL <http://proceedings.mlr.press/v70/kim17a.html>.
- [21] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings* DBL [128]. URL <https://openreview.net/forum?id=Sk2Im59ex>.
- [22] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III*, volume 9907 of *Lecture Notes in Computer Science*, pages 702–716. Springer, 2016. ISBN 978-3-319-46486-2. doi: 10.1007/978-3-319-46487-9_43. URL https://doi.org/10.1007/978-3-319-46487-9_43. 114
- [23] Donggeun Yoo, Namil Kim, Sunggyun Park, Anthony S. Paek, and In-So Kweon. Pixel-level domain transfer. *CoRR*, abs/1603.07442, 2016. URL <http://arxiv.org/abs/1603.07442>.
- [24] Wenqi Xian, Patsorn Sangkloy, Varun Agrawal, Amit Raj, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Texturegan: Controlling deep image synthesis with texture patches. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8456–8465. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00882. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Xian_TextureGAN_Controlling_Deep_CVPR_2018_paper.html.

- [25] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep photo style transfer. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6997–7005, July 2017. doi: 10.1109/CVPR.2017.740.
- [26] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 7
- [27] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 105–114. IEEE Computer Society, 2017. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.19. URL <https://doi.org/10.1109/CVPR.2017.19>. 7, 16, 90
- [28] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *The European Conference on Computer Vision Workshops (ECCVW)*, September 2018.
- [29] A. Lucas, A. K. Katsaggelos, S. Lopez-Tapuia, and R. Molina. Generative adversarial networks and perceptual losses for video super-resolution. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 51–55, Oct 2018. doi: 10.1109/ICIP.2018.8451714.
- [30] Subeesh Vasu, Nimisha T.M., and A.N. Rajagopalan. Analyzing perception-distortion tradeoff using enhanced perceptual super-resolution network. In *European Conference on Computer Vision (ECCV) Workshops*, 2018. 7
- [31] 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. URL <http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf>. 8, 9, 10, 17, 21, 25, 26, 57, 66, 107, 118
- [32] Abhishek Kumar, Prasanna Sattigeri, and Tom Fletcher. Semi-supervised learning with gans: Manifold invariance with improved inference. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5534–5544. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7137-semi-supervised-learning-with-gans-manifold-invariance.pdf>. 8

- [33] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Russ R Salakhutdinov. Good semi-supervised learning that requires a bad gan. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6510–6520. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7229-good-semi-supervised-learning-that-requires-a-bad-gan.pdf>.
- [34] Bruno Lecouat, Chuan Sheng Foo, Houssam Zenati, and Vijay Chandrasekhar. Manifold regularization with gans for semi-supervised learning. *CoRR*, abs/1807.04307, 2018. URL <http://arxiv.org/abs/1807.04307>. 8
- [35] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *CoRR*, abs/1703.05921, 2017. URL <http://arxiv.org/abs/1703.05921>. 8
- [36] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection. *CoRR*, abs/1802.06222, 2018. URL <http://arxiv.org/abs/1802.06222>. 8
- [37] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martín Arjovsky, Olivier Mastropietro, and Aaron C. Courville. Adversarially learned inference. *CoRR*, abs/1606.00704, 2016. URL <http://arxiv.org/abs/1606.00704>. 8
- [38] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *CoRR*, abs/1605.09782, 2016. URL <http://arxiv.org/abs/1605.09782>. 8
- [39] Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelfi. A survey on gans for anomaly detection. *CoRR*, abs/1906.11632, 2019. URL <http://arxiv.org/abs/1906.11632>. 8
- [40] Vladimir V. Kniaz, Vladimir A. Knyaz, Jiří Hladůvka, Walter G. Kropatsch, and Vladimir Mizginov. Thermalgan: Multimodal color-to-thermal image translation for person re-identification in multispectral dataset. In Laura Leal-Taixé and Stefan Roth, editors, *Computer Vision – ECCV 2018 Workshops*, pages 606–624, Cham, 2019. Springer International Publishing. ISBN 978-3-030-11024-6. 8
- [41] J. Zhang, Y. Peng, and M. Yuan. Sch-gan: Semi-supervised cross-modal hashing by generative adversarial network. *IEEE Transactions on Cybernetics*, 50(2):489–502, 2020. 8
- [42] Yue Cao, Bin Liu, Mingsheng Long, and Jianmin Wang. Hashgan: Deep learning to hash with pair conditional wasserstein gan. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 8

- [43] Hoang Thanh-Tung, Truyen Tran, and Svetha Venkatesh. Improving generalization and stability of generative adversarial networks. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=ByxPYjC5KQ>. 8, 20
- [44] Hao Ge, Yin Xia, Xu Chen, Randall Berry, and Ying Wu. Fictitious GAN: Training GANs with historical models. *CoRR*, abs/1803.08647, 2018. URL <http://arxiv.org/abs/1803.08647>. 9, 17
- [45] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6626–6637. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7240-gans-trained-by-a-two-time-scale-update-rule.pdf>. 17, 25, 26, 53, 57, 66, 73, 93
- [46] Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. Training GANs with optimism. In *Proc. International Conference on Learning Representations (ICLR)*, 2018. 17, 57, 67
- [47] Panayotis Mertikopoulos, Houssam Zenati, Bruno Lecouat, Chuan-Sheng Foo, Vijay Chandrasekhar, and Georgios Piliouras. Optimistic mirror descent in saddle-point problems: Going the extra (gradient) mile. In *ICLR*, 2019. 17, 57
- [48] Lars M. Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of gans. In *NIPS*, pages 1823–1833, 2017. 9, 17, 57, 67
- [49] Ryan Webster, Julien Rabin, Loïc Simon, and Frédéric Jurie. Detecting overfitting of deep generative networks via latent recovery. *CoRR*, abs/1901.03396, 2019. URL <http://arxiv.org/abs/1901.03396>. 9, 18
- [50] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings DBL [128]*. URL <https://openreview.net/forum?id=Bydr0Icle>. 9, 18
- [51] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (GANs). In Doina Precup and Yee Whye Teh, editors, *Proc. 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 224–232, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL <http://proceedings.mlr.press/v70/arora17a.html>. 9, 15, 16, 19

- [52] Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings* DBL [128]. URL https://openreview.net/forum?id=Hk4_qw5xe. 9, 19
- [53] Chang Xiao, Peilin Zhong, and Changxi Zheng. Bourgan: Generative networks with metric embeddings. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2269–2280. Curran Associates, Inc., 2018. URL <http://papers.nips.cc/paper/7495-bourgan-generative-networks-with-metric-embeddings.pdf>. 9, 19
- [54] Emily L Denton, Soumith Chintala, arthur szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1486–1494. Curran Associates, Inc., 2015. URL <http://papers.nips.cc/paper/5773-deep-generative-image-models-using-a-laplacian.pdf>. 10, 20
- [55] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proc. 34th International Conference on Machine Learning (ICML)*, pages 214–223, 2017. URL <http://proceedings.mlr.press/v70/arjovsky17a.html>. 10, 12, 21, 57, 62, 65, 107, 109, 121
- [56] Junbo Jake Zhao, Michaël Mathieu, and Yann LeCun. Energy-based generative adversarial network. *CoRR*, abs/1609.03126, 2016. URL <http://arxiv.org/abs/1609.03126>. 12, 107, 109, 121, 122
- [57] Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Q. Phung. Multi-generator generative adversarial nets. *CoRR*, abs/1708.02556, 2017. URL <http://arxiv.org/abs/1708.02556>. 15, 16, 121, 125
- [58] Arnab Ghosh, Viveka Kulharia, Vinay P. Namboodiri, Philip H. S. Torr, and Puneet Kumar Dokania. Multi-agent diverse generative adversarial networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 8513–8521, Salt Lake City, UT, 2018. doi: 10.1109/CVPR.2018.00888. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Ghosh_Multi-Agent_Diverse_Generative_CVPR_2018_paper.html. 15, 16
- [59] Ishan P. Durugkar, Ian Gemp, and Sridhar Mahadevan. Generative multi-adversarial networks. *CoRR*, abs/1611.01673, 2016. URL <http://arxiv.org/abs/1611.01673>. 16
- [60] Behnam Neyshabur, Srinadh Bhojanapalli, and Ayan Chakrabarti. Stabilizing GAN training with multiple random projections. *CoRR*, abs/1705.07831, 2017. URL <http://arxiv.org/abs/1705.07831>. 16

- [61] Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. Gang of gans: Generative adversarial networks with maximum margin ranking. *CoRR*, abs/1704.04865, 2017. URL <http://arxiv.org/abs/1704.04865>. 16
- [62] T. Doan, J. Monteiro, I. Albuquerque, B. Mazouze, A. Durand, J. Pineau, and R Devon Hjelm. On-line adaptative curriculum learning for GANs. *ArXiv e-prints*, July 2018. 16
- [63] Ilya O Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann SIMON-GABRIEL, and Bernhard Schölkopf. Adagan: Boosting generative models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5424–5433. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7126-adagan-boosting-generative-models.pdf>. 16
- [64] Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Phung. MGAN: Training generative adversarial nets with multiple generators. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=rkmu5b0a->. 16, 37, 90
- [65] Zili Yi, Hao (Richard) Zhang, Ping Tan, and Minglun Gong. Dualgan: Unsupervised dual learning for image-to-image translation. In *ICCV*, pages 2868–2876. IEEE Computer Society, 2017. 16, 90
- [66] Harm de Vries, Florian Strub, Jeremie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron Courville. Modulating early visual processing by language. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 6597–6607, December 2017. arxiv: 1707.00683. 16, 38
- [67] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJgw_sRqFQ. 16, 38
- [68] Yang Yu and Wen-Ji Zhou. Mixture of gans for clustering. In *Proc. Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3047–3053. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/423. URL <https://doi.org/10.24963/ijcai.2018/423>. 16
- [69] M. K. Titsias and A. C. Likas. Shared kernel models for class conditional density estimation. *IEEE Transactions on Neural Networks*, 12(5):987–997, Sep. 2001. ISSN 1045-9227. doi: 10.1109/72.950129. 16
- [70] M. K. Titsias and A. Likas. Class conditional density estimation using mixtures with constrained component sharing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):924–928, July 2003. ISSN 0162-8828. doi: 10.1109/TPAMI.2003.1206521. 16

- [71] Takuhiro Kaneko, Yoshitaka Ushiku, and Tatsuya Harada. Class-distinct and class-mutual image generation with gans. *CoRR*, abs/1811.11163, 2018. URL <http://arxiv.org/abs/1811.11163>. 17
- [72] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control and Optimization*, 30(4):838–855, July 1992. doi: 10.1137/0330046. 17
- [73] Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, et al. Parallel WaveNet: Fast high-fidelity speech synthesis. *CoRR*, abs/1711.10433, 2017. URL <http://arxiv.org/abs/1711.10433>.
- [74] B. Athiwaratkun, M. Finzi, P. Izmailov, and A. G. Wilson. Improving consistency-based semi-supervised learning with weight averaging. *ArXiv e-prints*, abs/1806.05594, June 2018.
- [75] Pavel Izmailov, Dmitrii Podoprikin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *CoRR*, abs/1803.05407, 2018. URL <http://arxiv.org/abs/1803.05407>.
- [76] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1195–1204. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/6719-mean-teachers-are-better-role-models-weight-averaged.pdf>. 17
- [77] Gauthier Gidel, Hugo Berard, Pascal Vincent, and Simon Lacoste-Julien. A variational inequality perspective on generative adversarial nets. *CoRR*, abs/1802.10551, 2018. URL <http://arxiv.org/abs/1802.10551>. 18
- [78] Olivier Bousquet, Roi Livni, and Shay Moran. Passing tests without memorizing: Two models for fooling discriminators. *CoRR*, abs/1902.03468, 2019. URL <http://arxiv.org/abs/1902.03468>. 18
- [79] Tatjana Chavdarova, Gauthier Gidel, François Fleuret, and Simon Lacoste-Julien. Reducing Noise in GAN Training with Variance Reduced Extragradient. *arXiv e-prints*, art. arXiv:1904.08598, Apr 2019. 18, 20
- [80] Vaishnavh Nagarajan, Colin Raffel, and Ian Goodfellow. Theoretical insights into memorization in gans. 2018. URL <https://colinraffel.com/publications/idlt2018theoretical.pdf>. 19
- [81] Sanjeev Arora, Andrej Risteski, and Yi Zhang. Do GANs learn the distribution? some theory and empirics. In *International Conference on*

- Learning Representations*, 2018. URL <https://openreview.net/forum?id=BJehNfW0->. 19, 88, 124
- [82] Matan Ben-Yosef and Daphna Weinshall. Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images. *CoRR*, abs/1808.10356, 2018. URL <http://arxiv.org/abs/1808.10356>. 19
- [83] Mahyar Khayatkhoei, Maneesh K. Singh, and Ahmed Elgammal. Disconnected manifold learning for generative adversarial networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7343–7353. Curran Associates, Inc., 2018. URL <http://papers.nips.cc/paper/7964-disconnected-manifold-learning-for-gans.pdf>. 19
- [84] Hoang Thanh-Tung, Truyen Tran, and Svetha Venkatesh. On catastrophic forgetting and mode collapse in generative adversarial networks. *CoRR*, abs/1807.04015, 2018. URL <http://arxiv.org/abs/1807.04015>. 19
- [85] Yasin Yazıcı, Chuan-Sheng Foo, Stefan Winkler, Kim-Hui Yap, Georgios Piliouras, and Vijay Chandrasekhar. The unusual effectiveness of averaging in GAN training. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SJgw_sRqFQ. 20, 38, 92
- [86] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M. Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: GANs do not need to decrease a divergence at every step. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=ByQpn1ZA->. 21
- [87] David Warde-Farley and Yoshua Bengio. Improving generative adversarial networks with denoising feature matching. In *International Conference on Learning Representations 2017 (Conference Track)*, 2017. URL <https://openreview.net/forum?id=S1X7nhsxl>. 21, 121
- [88] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL <http://yann.lecun.com/exdb/mnist/>. 21, 22
- [89] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL <https://arxiv.org/abs/1708.07747>. 21, 22, 40
- [90] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. URL http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf. 21, 22, 92

- [91] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL <http://www.cs.toronto.edu/~kriz/cifar.html>. 21, 22, 40, 65, 92, 118
- [92] A. Coates, H. Lee, and A.Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proc. 14th International Conference on Artificial Intelligence and Statistics*, volume 15 of *JMLR Workshop and Conference Proceedings*, pages 215–223. JMLR W&CP, 2011. URL <http://jmlr.csail.mit.edu/proceedings/papers/v15/coates11a.html>. 21, 23, 65, 118
- [93] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proc. International Conference on Computer Vision (ICCV)*, 2015. 21, 24, 40, 65, 118
- [94] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015. doi: 10.1007/s11263-015-0816-y. 21, 24, 65, 124, 125
- [95] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML’16, pages 1747–1756. JMLR.org, 2016. URL <http://dl.acm.org/citation.cfm?id=3045390.3045575>. 24, 108, 109, 113
- [96] Ali Borji. Pros and cons of GAN evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019. doi: 10.1016/j.cviu.2018.10.009. URL <https://doi.org/10.1016/j.cviu.2018.10.009>. 25
- [97] K. Shmelkov, C. Schmid, and K. Alahari. 25
- [98] L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, Apr 2016. URL <http://arxiv.org/abs/1511.01844>. 25, 93, 118
- [99] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016. URL <http://arxiv.org/abs/1512.00567>. 26, 27, 93
- [100] S. Barratt and R. Sharma. A Note on the Inception Score. *ArXiv e-prints*, January 2018. 26, 73
- [101] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M. Dai, Shakir Mohamed, and Ian J. Goodfellow. Many paths to equilibrium: Gans do not need to decrease a divergence at every step. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada*,

- April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL <https://openreview.net/forum?id=ByQpn1ZA->. 41
- [102] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL <http://arxiv.org/abs/1412.6980>. 41, 42, 44, 65, 92, 119
- [103] Panayotis Mertikopoulos, Christos Papadimitriou, and Georgios Piliouras. Cycles in adversarial regularized learning. In *Proc. Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 2703–2717, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. ISBN 978-1-6119-7503-1. URL <http://dl.acm.org/citation.cfm?id=3174304.3175476>. 57, 60
- [104] Christos Papadimitriou and Georgios Piliouras. From nash equilibria to chain recurrent sets: An algorithmic solution concept for game theory. *Entropy*, 20(10), 2018. 57
- [105] James P. Bailey and Georgios Piliouras. Multiplicative weights update in zero-sum games. In *Proc. ACM Conference on Economics and Computation (EC)*, pages 321–338. ACM, 2018. doi: 10.1145/3219166.3219235. 57
- [106] Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999. URL <https://EconPapers.repec.org/RePEc:eee:gamebe:v:29:y:1999:i:1-2:p:79-103>. 58, 60
- [107] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017. URL <http://arxiv.org/abs/1701.00160>. 60
- [108] James P Bailey and Georgios Piliouras. Multi-agent learning in network zero-sum games is a Hamiltonian system. In *Int. Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2019. 60, 61
- [109] David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *Proc. ICML*, 2018. 61
- [110] Vaishnavh Nagarajan and J. Zico Kolter. Gradient descent gan optimization is locally stable. In *Advances in Neural Information Processing Systems*, pages 5585–5595, 2017. 62
- [111] G. Gidel, R. Askari Hemmat, M. Pezeshki, R. Lepriol, G. Huang, S. Lacoste-Julien, and I. Mitliagkas. Negative momentum for improved game dynamics. *ArXiv e-prints*, abs/1807.04740, July 2018. 62, 64
- [112] N. Flammarion and F. Bach. From averaging to acceleration, there is only a step-size. *ArXiv e-prints*, abs/1504.01577, April 2015. 63

- [113] Zhiming Zhou, Han Cai, Shu Rong, Yuxuan Song, Kan Ren, Weinan Zhang, Jun Wang, and Yong Yu. Activation maximization generative adversarial nets. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=HyyP33gAZ>. 73
- [114] Ting Chen, Xiaohua Zhai, Marvin Ritter, Mario Lucic, and Neil Houlsby. Self-supervised generative adversarial networks. *CoRR*, abs/1811.11212, 2018. URL <http://arxiv.org/abs/1811.11212>. 90
- [115] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-Attention Generative Adversarial Networks. *ArXiv e-prints*, May 2018. 91
- [116] Mario Lucic, Michael Tschannen, Marvin Ritter, Xiaohua Zhai, Olivier Bachem, and Sylvain Gelly. High-fidelity image generation with fewer labels. *CoRR*, abs/1903.02271, 2019. URL <http://arxiv.org/abs/1903.02271>. 91
- [117] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 10236–10245, 2018. URL <http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions>. 93
- [118] Marco Marchesi. Megapixel size image creation using generative adversarial networks. *CoRR*, abs/1706.00082, 2017. URL <http://arxiv.org/abs/1706.00082>. 93
- [119] Eirikur Agustsson, Alexander Sage, Radu Timofte, and Luc Van Gool. Optimal transport maps for distribution preserving operations on latent spaces of generative models. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=BklCusRct7>. 99
- [120] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Multi-class generative adversarial networks with the L2 loss function. *CoRR*, abs/1611.04076, 2016. URL <http://arxiv.org/abs/1611.04076>. 107, 115
- [121] Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’15, pages 1927–1935, Cambridge, MA, USA, 2015. MIT Press. URL <http://dl.acm.org/citation.cfm?id=2969442.2969455>. 109, 113, 125
- [122] Ferenc Huszar. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *CoRR*, abs/1511.05101, 2015. URL <http://arxiv.org/abs/1511.05101>. 113
- [123] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine*

- Learning* - Volume 37, ICML'15, pages 448–456. JMLR.org, 2015. URL <http://dl.acm.org/citation.cfm?id=3045118.3045167>. 116, 119
- [124] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <http://dx.doi.org/10.1162/neco.1997.9.8.1735>. 116
- [125] Tu Nguyen, Trung Le, Hung Vu, and Dinh Phung. Dual discriminator generative adversarial nets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2667–2677. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/6860-dual-discriminator-generative-adversarial-nets.pdf>. 121
- [126] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997. ISSN 1053-587X. doi: 10.1109/78.650093. URL <http://dx.doi.org/10.1109/78.650093>. 125
- [127] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. 2017. URL <https://openreview.net/forum?id=SkxKPDv5xl>. 125
- [128] *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. OpenReview.net. URL <https://openreview.net/group?id=ICLR.cc/2017/conference>. 131, 134, 135