Creating Logos Using

GENERATIVE ADVERSARIAL NETWORKS

SV Sriharsha

Supervisor:
Prof. S Bapi Raju

This thesis is presented as part of the requirements for the conferral of the degree:

Integrated MSc in Mathematics

University of Hyderabad
School of Mathematics and Statistics

May, 2018

Abstract

The objective of this project is to explore Generative Adversarial Networks by understanding the internal working of the network and then applying it to artificially generate Logos. The first chapter introduces GAN and it's loss function. It then discusses a few problems with traditional GANs. The second chapter explores a few models of GAN that address the problems pointed out in the first chapter.
All the models in chapter 2 are individually used to generate Logos. Chapter 3 describes the precise architecture of each model, and the dataset used to train these models. It then discusses and compares the results obtained from each model. An additional idea of latent space exploration - interpolation- is described whose results are then shown.

Acknowledgments

I would like to thank Prof. S Bapi Raju for guiding me with his expertise in the subject. I would also like to thank School of Computer and Information Sciences (particularly Prof. Chakravarthy Bhagvati) for providing me with the infrastructure required for carrying out this project.

I would like to thank my family and all my friends for their constant support.

Lastly, I would also like to thank School of Mathematics and Statistics for giving me complete autonomy over project selection and allowing me to pursue this project outside the department.

Contents

Chapter 1

Generative Adversarial Network

1.1 Introduction

Generative Adversarial Network is a framework proposed by Ian Goodfellow et al. in 2014[1] where two models; a generative and a discriminative model; are simultaneously trained. The generative model G tries to capture the data distribution it is trained on, while the discriminator D tries to determine the probability that a given sample came from the original training data rather than G.

As shown in figure 1.1, a noise vector z is presented to G as an input. G outputs x', which is a fake sample generated by G. D is presented with either this fake sample or a real data sample, and it has to determine the probability that the given sample is from the real data.

G aims is to make D output 1 even for fake samples (i.e., to fool the Discriminator into believing that the fake sample is indeed real). Whereas the aim of D is to output 1 for all the real samples and 0 for all the fake samples. This setup corresponds to a two player minimax zero-sum game.

As an analogy, the above discussed system is similar to a con artist trying to imitate an acclaimed piece of art and the police trying to detect the fake art. The

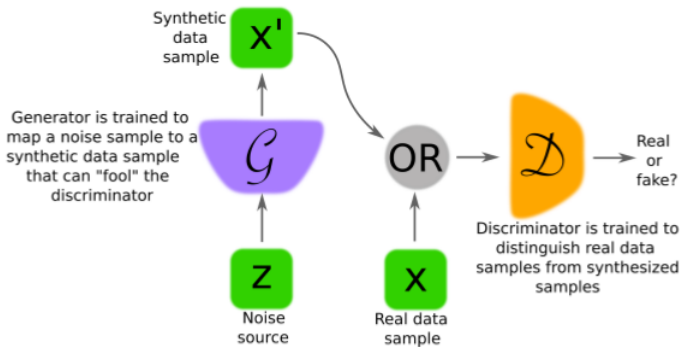


Figure 1.1: Cresswell et al., Generative Adversarial Networks: An Overview [2].

con artist imitates the artwork, which goes to the police who realizes it is fake. With some feedback, the con artist attempts to imitate the artwork again (generating a better imitation than before), which again goes to the police. This process goes on. A *"balanced"* competition results in both sides improving till a point where the art generated by the con artist is indistinguishable from the original art.

Similarly, an ideal outcome of the above described system is when both G and D continue to improve over iterations and ultimately, the samples output by G are indistinguishable from the training data. This results in D having to output $\frac{1}{2}$ for samples from both sources since it cannot differentiate them anymore.

1.2 Quantifying the similarity between two probability distributions

1.2.1 Kullback Leibler Divergence

The measure of how a probability distribution p diverges from a second probability distribution q is,

$$D_{KL}(p,q) = \int_x \frac{p(x)}{q(x)} dx \tag{1.1}$$

It is noticeable from the formula that KL divergence is not symmetric. In cases where $p(x)$ is close to zero, q's effect is disregarded.

1.2.2 Jensen Shannon Divergence

JS divergence, on the other hand is symmetric, and also bounded by $[0,1]$

$$D_{JS}(p,q) = D_{KL}(p, \frac{p+q}{2}) + D_{KL}(\frac{p+q}{2}, q) \tag{1.2}$$

While other traditional methods such as Maximum Likelihood Approach use Kullback Leibler Divergence, GANs use Jensen Shannon divergence. More on this in appendix A.1

1.3 Loss function for GANs

The learning process of GANs involve training D and G simultaneously. G's target is to learn the real probability distribution p_g over data x. G starts by sampling input noise z from a uniform or gaussian distribution $p_z(z)$ which is mapped to the data space $G(z)$. D on the other hand is a classifier that tries to recognize if a given sample is from the training data or from G.

The minimax game of D and G can be viewed as an optimization of the following function,

$$\begin{aligned}\min_G \max_D L(D,G) &= E_{x\in p_r(x)}[\log(D(x))] + E_{z\in p_z(z)}[\log(1-D(G(z)))] \\ &= E_{x\in p_r(x)}[\log(D(x))] + E_{x\in p_g(x)}[\log(1-D(x))]\end{aligned} \tag{1.3}$$

The loss function,

$$L(G,D) = \int_x (p_r(x)\log(D(x)) + p_g(x)\log(1-D(x)))dx \tag{1.4}$$

where G is trying to minimize $L(G,D)$ while D is trying to maximize it. This is in accordance with the final target of both G and D. G can only control the $\log(1-D(G(z)))$ term in the equation 1.3 to minimize it. So it only minimizes $E_{z\in p_z(z)}[\log(1-D(G(z)))]$, which implies that G wants $D(G(z))$ to be close to 1.This means, the generator wants discriminator to believe that the samples from the generator are coming from the real data. D, however, can control both terms on the right hand side of equation 1.3. While maximizing equation 1.3, D tries to ensure that when $x \in p_r(x)$, $D(x)$ is close to 1, and when $x \in p_g(x)$, $D(x)$ is close to 0.

1.4 Problems with the training of GAN

Although GANs have shown promising results by generating realistic images, the training process is slow and very unstable. The following problems were observed.

1.4.1 Nash equilibrium might not be achieved

During a GAN's training, G and D update their weights independently without taking into account other player's actions. In the paper titled 'Improved Techniques for training GANs', Salimans et al.[3] point out that concurrently updating the gradients of the models cannot guarantee convergence.

1.4.2 Supports of p_g and p_r are low dimensional

Although the images seem to belong to very high dimensional space (for example, color images with a size of 100×100 pixels belong to a 30000 dimensional space), all the *meaningful* images lie in a relatively smaller dimensional manifold. Hence, p_r - the data distribution over real sample x - lies in a low dimensional manifold. This is true for G's distribution over data x, p_g too. Since both p_r and p_g are in a significantly lower dimensional manifold than the space containing all the images of

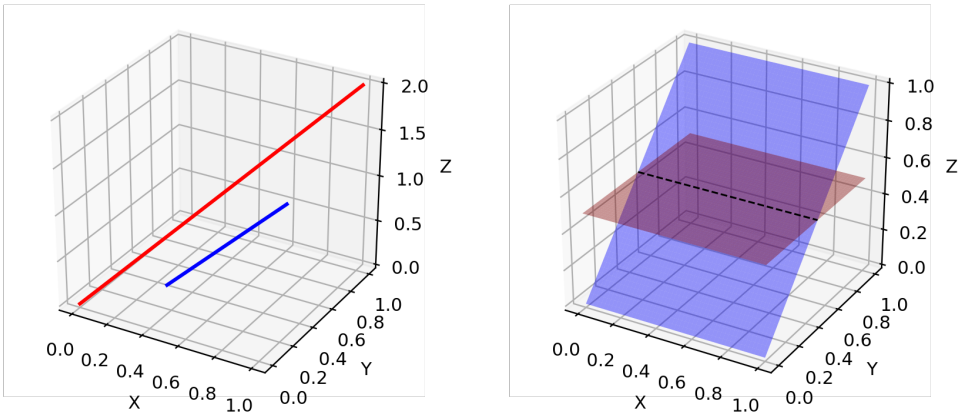


Figure 1.2: The image on the right shows that two planes in R^3 have little chance of overlapping. In the left, the two lines have an even lesser chance of overlap. (*Source: Lilian Weng, From GAN to WGAN*)

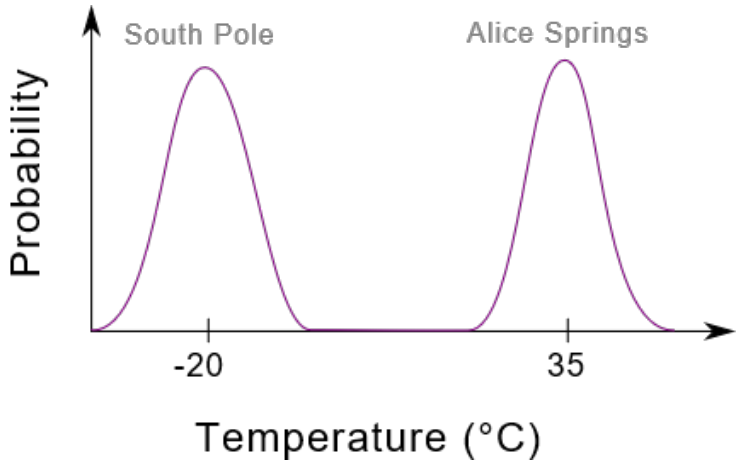


Figure 1.3: Distribution of temperatures in South pole and Alice Springs (*Source: Aiden Nibali's blog*)

similar dimensions, they are almost always disjoint (see Figure 1.2). In such case, there always exists a discriminator that seperates the real and the fake samples with complete accuracy as proved by Arjovsky et al. in their paper titled, *Towards principled methods for training Generative Adversarial Networks* [4].

1.4.3 Vanishing Gradient

If the discriminator is perfect, then $D(x) = 1 \ \forall x \in p_r$ and $D(x) = 0 \ \forall x \in p_g$. The loss function then becomes zero and hence, there is no gradient to update the weights during training.

This results in a dilemma during training a GAN. If the discriminator is weak, then the generator does not receive accurate feedback for the weight update. Whereas, if the discriminator is strong, the gradients drop down to zero making the learning very slow. This makes the training of GANs very difficult as care must be taken to

ensure the correct balance between the strengths of discriminator and generator.

1.4.4 Mode Collapse

Most of the *real-world* data distributions are complex and multimodal, i.e., the probability distribution has multiple peaks. Consider a case where we want to train a GAN which produces temperature values similar to the distribution shown in figure 1.3.

Under ideal conditions, we expect the generator to produce cold and hot temperatures with equal probability. But sometimes, the generator produces only one kind *(say cold)* temperatures. The discriminator realizes this and updates itself in such a way that it categorizes all hot temperatures as real. Now the generator observes this and starts producing only hot temperatures. This cycle continues and results in a model that is highly biased towards only one peak (i.e., it produces very similar looking samples).

Chapter 2

A few models of GANs

2.1 Deep Convolutional GAN

The paper of Radford et al.[5] gives a few hints on what is a good DCGAN architecture:

1. All pooling layers should be replaced with strided convolutions for the discriminator and fractional-strided convolutions for the generator.
2. Batch normalization should be used for both the generator (except at the output layer) and the discriminator (except at the input layer).
3. Fully connected hidden layers should be removed for deeper architectures.
4. ReLU activation should be used in generator for all layers except the output layer. Leaky ReLU activation should be used for all layers in the discriminator.

The paper also recommends the following hyper-parameter settings for the training:

1. All weights should be initialized from a normal distribution with mean as 0 and standard deviation as 0.2
2. The slope for the Leaky ReLU should be set to 0.02.
3. Adam optimizer should be used with learning rate as 0.0002 and momentum term as 0.5.

DCGANs perform better than the original *vanilla* GAN. Most models (including all the subsequent ones discussed in this chapter) follow these suggestions.

In-spite of the refinements given by the architecture of DCGAN, they still suffer from all the problems listed out in 1.4. Salimans et al. [3] proposed the following improvements in order to mitigate these problems:

1. **Feature Matching**:
   Feature matching suggests that the discriminator should be optimized to check whether output of the generator matches the statistics of real samples. A new loss function of the form, $\left\|E_{x\ p_r} f(x) - E_{z\ p_z(z)} f(G(z))\right\|_2^2$, can be used where $f(x)$ can be any statistical feature such as mean, median etc.

2. **Minibatch discrimination**:
   In minibatch discrimination, instead of processing each data point independently, the discriminator digests the relation between data points in a batch of data points. In a minibatch, the closeness between every pair of samples, $c(x_i, x_j)$ is calculated and an overall summary of a datapoint, $o(x_i) = \sum_j c(x_i, x_j)$ is added to the input of the model.

3. **Historic averaging**:
   Consider Θ to be a model parameter and Θ_i to be the configuration of Θ in the i^{th} iteration. Then, $\left\|\Theta - \frac{1}{t}\sum_{i=1}^{t}\Theta_i\right\|$ is added to the loss function for both models. This penalizes the weights that are rather far away from their historic average values.

4. **One sided label smoothing**:
   Softer values such as 0.1 and 0.9 are used instead of 0 and 1 to feed the discriminator to reduce the vulnerability of the network.

5. **Virtual batch normalization**:
   A fixed batch of data is used as a reference batch and each data sample is normalized based on the reference batch instead of normalizing within it's minibatch. The reference batch stays the same throughout the training.

2.2 Least Squares GAN

The discriminator of a GAN can be viewed as a classifier that uses sigmoid cross entropy loss function. This leads to vanishing gradients when the generator is updated using fake samples that are far away from the real data, but are still correctly classified.

As shown in figure 2.1, when the fake samples in magenta are used for updating the generator, they will cause no weight update since these samples are already classified as real data *(which the generator wants, hence the error becomes 0)*.However, the samples are actually far away from the real data and ideally, we want these points to get pulled closer to the real data. Since the sigmoid cross entropy saturates very fast, such points are not penalized for being far away from the real data. Mao et al.[6] propose in their paper, *Least Squares Generative Adversarial Networks*

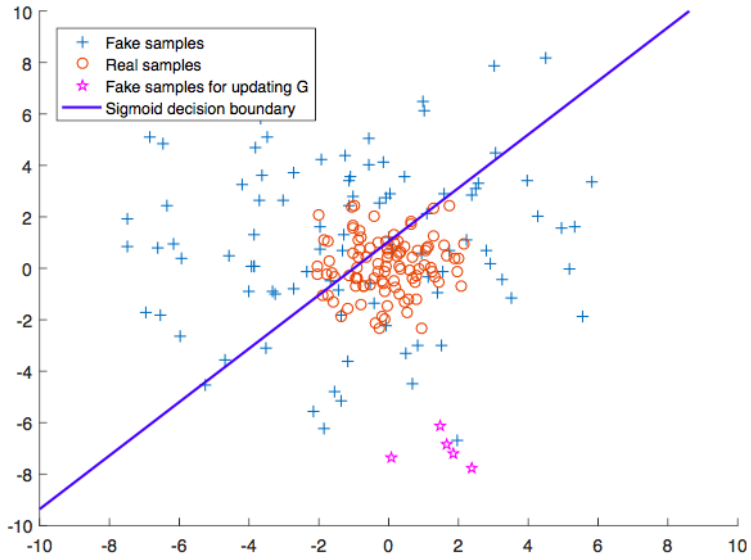


Figure 2.1: Fake samples that are far away from the real data but classified as real data are not penalized by the traditional loss function. *Source: LSGAN, Mao et. al [6]*

a correction in the loss function that penalizes samples that are far away from the decision boundary.

The minimax objective function for GANs was formulated in equation 1.3. The new objective function proposed in LSGAN is,

$$\begin{aligned}\min_D V_{LSGAN}(D) = &\frac{1}{2}\, E_{x\in p_{data}(x)}[(D(x)-b)^2] \\ &+ \frac{1}{2}\, E_{z\in p_z(z)}[(D(G(z))-a)^2] \\ \min_G V_{LSGAN}(G) \;= &\frac{1}{2} E_{z\in p_z(z)}[(D(G(z))-c)^2]\end{aligned}$$

where a and b are the labels for fake and real data respectively, and c is the value G wants D to output for fake data.

This new loss function penalizes the the samples that are far away from the decision boundary to ensure they are pulled closer to the real data. It also provides such samples with a gradient for weight update unlike the original loss function of GAN which saturates for samples that are far away. The new loss function hence, solves the problem of vanishing gradient too.

2.3 Wasserstein GAN

2.3.1 Wasserstein distance

Also known as Earth mover distance, Wasserstein distance is a measure of the the distance between two probability distributions. Informally, it is interpreted as the minimum effort required move piles of dirt that follow one probability distribution to another probability distribution.

For continuous probability domain, the distance formula is,

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_g)} E_{(x,y)\sim\gamma}[\|x - y\|] \tag{2.1}$$

where $\Pi(p_r, p_g)$ is a set of all the possible joint probability distributions between p_r and p_g. Arjovsky et al. proposed the use of Wasserstein distance as the loss function in GANs instead of the originally proposed JS divergence based loss function in the paper, *Wasserstein GAN* [7].

2.3.2 Why use Wasserstein distance

Consider a problem where a GAN is to be trained to generate images of some object. As pointed out in section 1.4.2, the supports of p_r and p_g are most likely disjoint. JS divergence, in such case, does not give any feedback for weight update of the network. Wasserstein distance, on the other hand, still gives gradients for effective weight update. An illustration of this is presented in Appendix A.2

Since it is intractable to exhaust all possibilities of γ to compute the infimum, Arjovsky et al. propose a transformation of the wasserstein distance based on Kantorovich-Rubenstein duality resulting in,

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} E_{x\sim p_r}[f(x)] - E_{x\sim p_g}[f(x)] \tag{2.2}$$

2.3.3 Lipschitz condition

The new metric proposed above in equation 2.2 admits only K-Lipschitz functions (i.e., $f : R \to R$ such that $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$). To enforce this condition, the authors propose to clamp the weights of the critic function after every gradient update to a fixed range, $[-c, c]$.

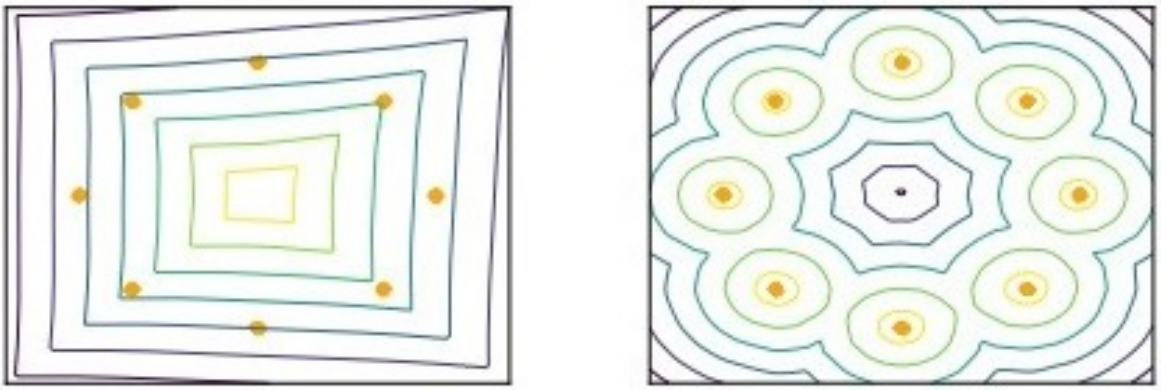


Figure 2.2: A WGAN fails to capture the higher moments of 8-Gaussians. *(source: Gulrazani et al., Improved Training of WGANs [8]).*

2.4 Improved WGAN

Gulrajani et al. pointed out in their paper titled *Improved Training of Wasserstein GANs* [8] point out the following shortcomings of weight clipping in WGANs to enforce K-Lipschitz constraint.

Capacity underuse:
The authors claim that weight clipping biases the critic of the WGAN to much simpler functions. They demonstrate it using the example shown in figure 2.2 where a WGAN trained to simulate 8 Gaussians fails to capture the higher moments of the given data distribution.

Exploding and vanishing gradients:
The authors observe that optimization process of the WGAN is difficult because of the interactions between the cost function and the weight constraint that result in either exploding or vanishing gradients if the clipping threshold is not carefully chosen.

2.4.1 Gradient Penalty

An alternate method to impose Lipschitz constraint is proposed by the authors. They observe that a differentiable function is 1-Lipschitz iff it's gradients have norm 1 everywhere. Hence, they propose to directly constraint the gradient norms of the output of the critic with respect to the input. To circumvent the tractability issues, a soft version of the constraint with penalty on gradient norm was enforced for random samples $\hat{x} \sim P_{\hat{x}}$. The new objective is,

$$L = E_{\tilde{x}\sim P_g}[D(\tilde{x})] - E_{\sim P_r}[D(x)] + E_{\hat{x}\sim P_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\tilde{x})\|_2 - 1)^2\right] \tag{2.3}$$

Although most GAN implementations use batch normalization in both generator and discriminator network to stabilize the training, the penalized training objective proposed above in equation 2.3 is not valid with the use of batch normalization. The authors hence suggest omission of batch normalization in the critic's network finding that WGAN-GP (WGAN with the Gradient Penalty) performs well without it.

Chapter 3

Generating Logos: Experiments and Results

The four different GANs that were discussed in Chapter 2, i.e., DCGAN, LSGAN, WGAN and WGAN-GP were used to generate samples similar to the ones in the dataset described below. This chapter discusses each model's architecture and the results obtained from these models. Code for all four models is obtained from the GitHub repository of Zhenliang He[a].

3.1 The Dataset

The dataset used for all experiments in this chapter is called the Large Logo Dataset [9]. It contains 1,22,920 high resolution logos of 400 × 400 pixels (see figure 3.1).

Figure 3.1: A few sample logos from LLD [9]. *Source: LLD, Large Logo Dataset*

The logos were resized to 64 × 64 pixels to stabilize the training of GAN.

[a]https://github.com/LynnHo/DCGAN-LSGAN-WGAN-WGAN-GP-Tensorflow

3.2 Architecture of the various models used

All weights are initialized from a truncated normal distribution[b] with mean 0 and standard deviation 0.2.
Batch normalization has decay of 0.9 and $\epsilon = 10^5$. All Leaky ReLU activations have leak set to 0.2.

3.2.1 Generator

All four models use this as their generator. The generator takes in a 100 dimensional input vector (say z).
All hidden layers have batch normalization and ReLU activation. The output layer has a tan hyperbolic activation without batch normalization.

1. The first layer is a fully connected layer with $4 \times 4 \times 64 \times 8 = 8192$ nodes. The output is then reshaped to an array of shape 4×4 with 512 filters.

2. The next is a deconvolutional layer with a kernel size of 5×5, 256 filters, and 2 strides.

3. This is followed by another deconvolutional layer with the same kernel size and number of strides (i.e., 5×5 and 2),and with 128 filters.

4. Another deconvolutional layer follows with the same kernel size and strides again, but with only 64 filters.

5. The output layer too is deconvolutional layer with the same kernel size and strides, but has 3 filters. Tan hyperbolic activation is used here.

The generator outputs an array of size $64 \times 64 \times 3$ which results in a 64×64 pixel colored image.

3.2.2 Discriminator

DCGAN, LSGAN and WGAN use this architecture with batch normalization in hidden layers 2,3 and 4 for their discriminator.WGAN-GP on the other hand uses layer normalization in those hidden layers.
All layers except the output layer use Leaky ReLU activation with a leak of 0.2.
The discriminator takes in an array of shape $64 \times 64 \times 3$ as input.

1. The first layer is a convolutional layer with kernel size of 5×5, 64 filters, and 2 strides.

[b] truncated normal distribution only takes values that are within two standard deviations from the mean.

2. Another Convolutional layer with same kernel size and strides follows with 128 filters.

3. There is another convolutional layer with same kernel size and strides but with 256 filters.

4. This is followed by yet another convolutional layer with again same kernel size and strides but 512 filters.

5. The output layer is a fully connected layer with just one node.

3.3 Results

3.3.1 DCGAN

DCGAN starts producing *logo-like* pictures very soon (see figure 3.3). Initially, the training goes well as evident from the discriminator loss function in figure 3.2, and the pictures reach the highest visual quality from epoch 8 to epoch 10 (see figure 3.4 and figure 3.5). But soon after epoch 10, the images start showing signs of mode collapse (figure 3.6). By the end of the training, the logos have heavy mode collapse (figure 3.7).

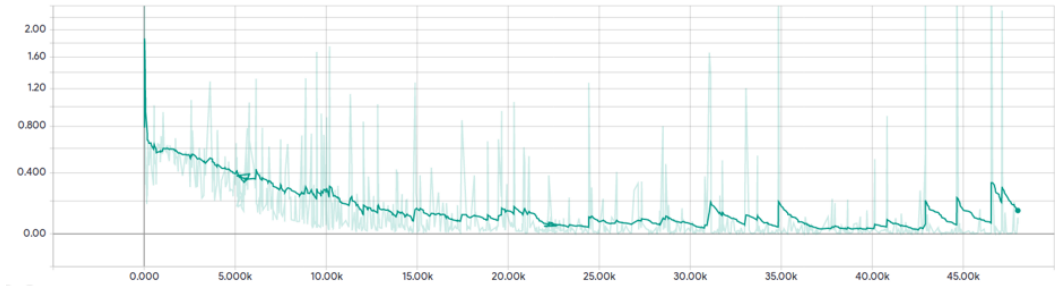


Figure 3.2: Discriminator loss function of DCGAN over iterations

This renders the model unusable since the training does not converge. Irrespective of the number of epochs the model is trained for, this trend of mode collapse will continue. But this is not completely unexpected since DCGANs are in general, not capable of producing such high dimensional and multimodal images.

Figure 3.3: Images produced by DCGAN at epoch 2: *Training starts off well with images looking like logos right after little training.*

Figure 3.4: Images produced by DCGAN at epoch 8:*This is the region where the images looked good*

Figure 3.5: Images produced by DCGAN at epoch 10: *These are the best looking images that were produced by this model.*

Figure 3.6: Images produced by DCGAN at epoch 11. The orange border around some of the images is an overlay put to highlight the images that show mode collapse.

Figure 3.7: Images produced by DCGAN at epoch 24. Heavy mode collapse as evident from the images with orange and yellow borders to highlight their similarity

3.3.2 LSGAN

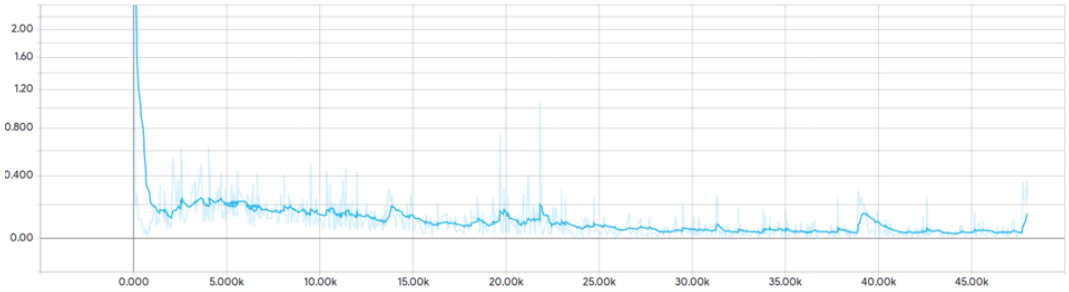


Figure 3.8: Discriminator loss function of DCGAN over iterations

LSGAN starts off a bit slower when compared to DCGAN. The training seems to be at it's infancy even after epoch 2 (see figure 3.9). Although there are a few peaks in the loss function (figure 3.8) at various places, there is no mode collapse during training. The images slowly get better while training (figure 3.10). But eventually, the images start to degrade along with the peak in loss function near the end and ultimately, after the training, the images look very degraded (figure 3.11).

Figure 3.9: Images produced by LSGAN at epoch 2: *The training is still at infancy at this point.*

Figure 3.10: Images produced by LSGAN at epoch 18. *These are the best logos produced by this architecture. Images start to degrade after this point during training.*

Figure 3.11: Images produced by LSGAN at epoch 24. *It is evident that the logos have degraded from previous epochs.*

3.3.3 WGAN

The training of WGAN goes a lot smoother with no sudden peaks, and the loss seems to converge. This reflects in the generation of images. Although the images don't show any mode collapse or degradation over training, not all of them are very realistic looking.

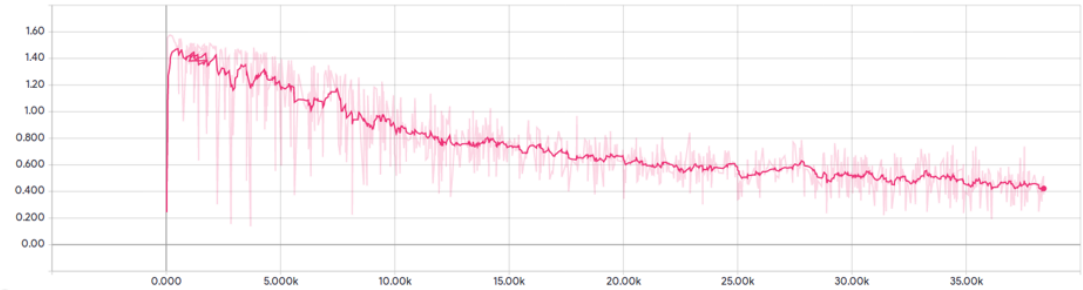


Figure 3.12: Discriminator loss function of WGAN over iterations: The loss function looks smooth and seems to converge

3.3.4 WGAN-GP

The final model used was WGAN with Gradient Penalty. The discriminator loss function again seems to converge very smoothly without any noticeable peaks through-

Figure 3.13: Images produced by WGAN after training. *The model produces images which don't degrade or suffer mode collapse, but not all images are very good.*

out the training (figure 3.14). This is the most advanced model of all and as expected, it produces the best results of all. The images generated improve gradually throughout the training and towards the end of the training, this model produces very realistic logos consistently (see figure 3.15).

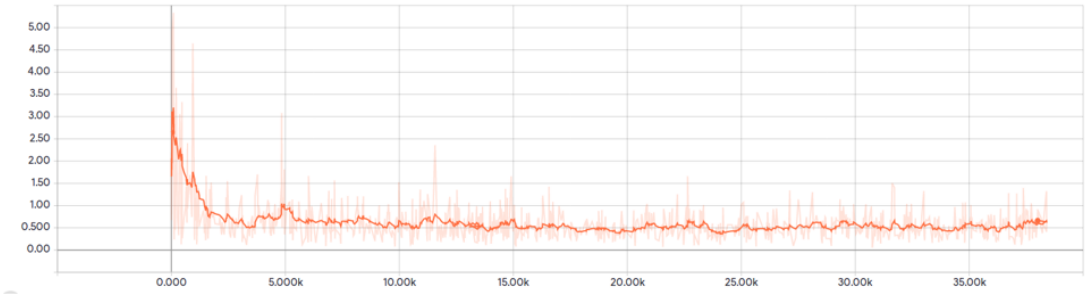


Figure 3.14: Discriminator loss function of WGAN-GP over iterations: The loss function looks smooth and seems to converge fast

3.3.5 Interpolation

After training, the generator becomes the main model. When it is given an input (of random noise), it outputs an image of a logo as shown in figure 3.16.
Now consider two images of different logos. Each of then would have a corresponding

Figure 3.15: Images produced by WGAN-GP after training. *The model produces very realistic images with no degradation or mode collapse. These are the best images of the four models that were trained.*

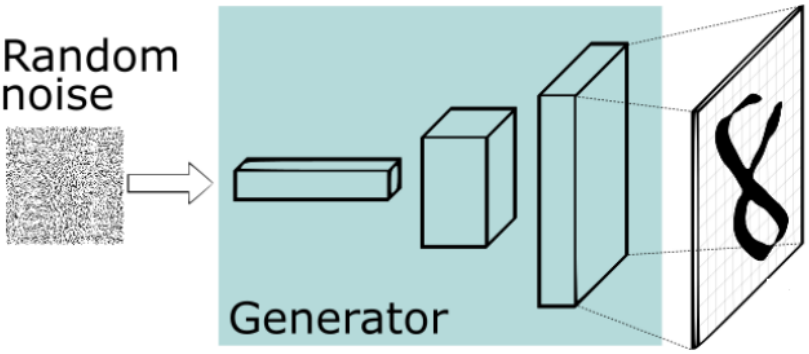


Figure 3.16: The generator takes in random noise as input and gives a meaningful image as output. *(Source: Deeplearning4j)*

noise vector (that generates the image of logo when given to the generator as an input). Let the two noise vector corresponding to the first image be n and the noise vector corresponding to the second image be m. Let,

$$n = [n_1, n_2, n_3, ...n_{100}]$$
$$m = [m_1, m_2, n_4, ...m_{100}]$$

Now, we look at the convex combination of these two noise vectors, i.e., noise of the

form,

$$noise = [\alpha.m_1 + (1-\alpha).n_1, \alpha.m_2 + (1-\alpha).n_2, ..., \alpha.m_{100} + (1-\alpha).n_{100}]$$

where $0 < \alpha < 1$.

Generating logos from the noise vectors obtained in equation 3.3.5 by gradually increasing α from 0 to 1 (i.e., changing noise from n to m gradually) results in a series of images which slowly transform from the first image to the second. An example of this is shown in figure 3.17. More such images are shown in Appendix A.3.

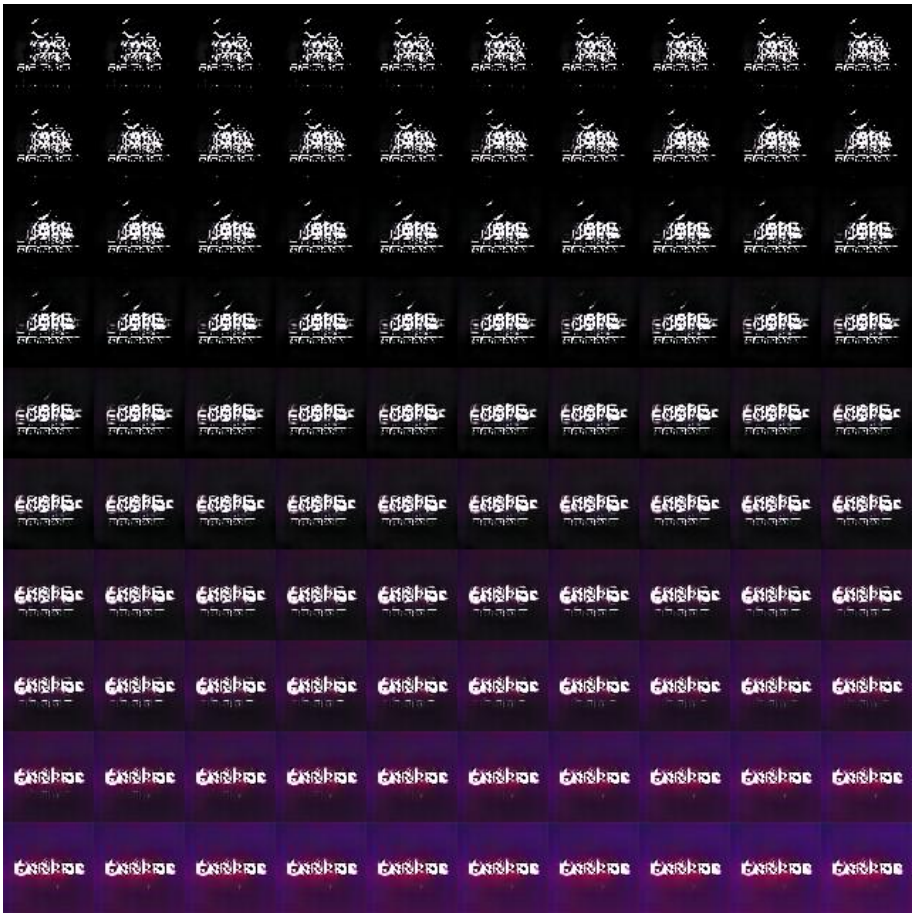


Figure 3.17: Interpolation between two logos*(First image on top left corner and the second image on bottom right)*

Similarly, this sort of interpolation can be done among four images of logos. The results of such interpolations are interesting too (as seen in figure 3.18). More such images are shown in Appendix A.3. This method of interpolation can help in choosing a logo that is a combination of the given two (or four) logos.

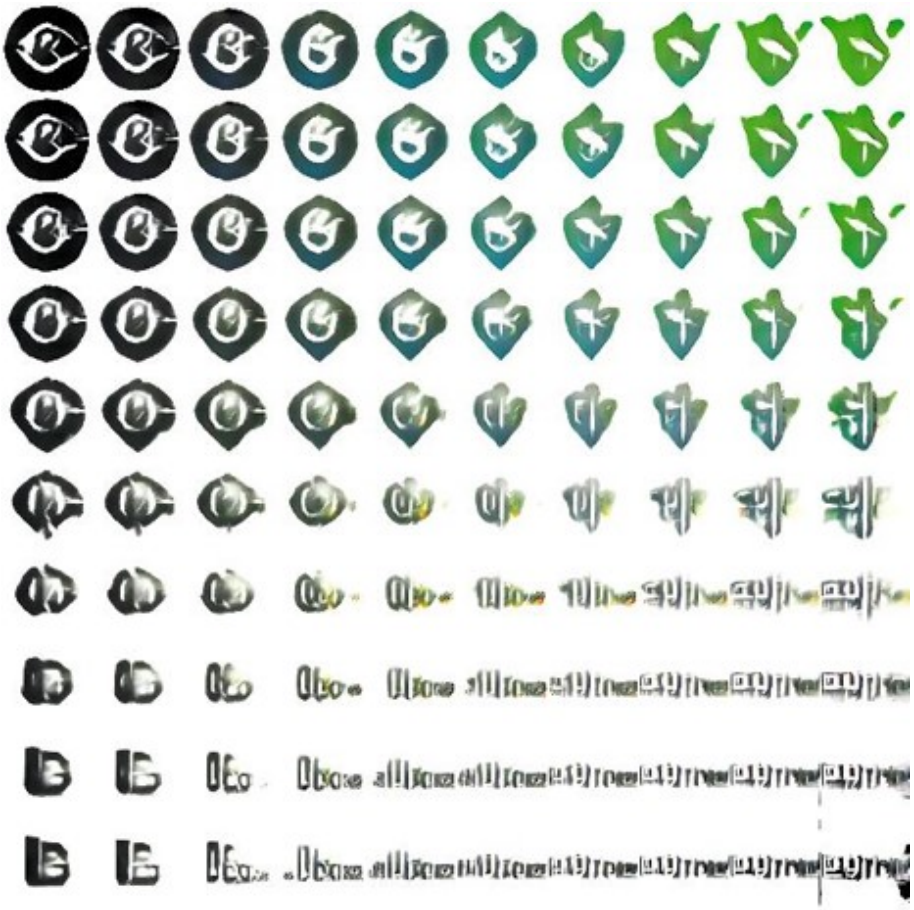


Figure 3.18: Interpolation between 4 images*(One in each corner)*

3.3.6 Conclusion

Four prominent models of GANs were implemented to generate images of logos. The performance of each model was consistent with the theoretical expectations. WGAN-GP gave the best results of all four models with very realistic looking images of logos. This was closely followed by WGAN where not all images looked very good. The images generated by LSGAN were not very realistic towards the end when the discriminator loss started to diverge. The images by DCGAN showed severe mode collapse rendering the model useless. A method for interpolation of logos was proposed which showed promising results in finding logos with the combined features of the given ones.

Chapter 4

Future Work

1. Along with interpolation, another idea of latent space exploration was to generate logos similar to a given logo. An attempt was made to do this by slightly disturbing the noise pertaining to the given logo. To do this, a gaussian noise with mean 0 was added to the original noise. The standard deviation was varied in order to find the perfect one. This; however; did not yield good results. The images almost the same when the standard deviation was low, and they were giving completely different logos when the standard deviation was increased. The problem might not be just about fine tuning standard deviation. Further exploration should be done by examining how varying each dimension in the noise vector affects the output logo. This would help in making controlled changes to a given logo.

2. Another way of modifying a particular logo is by *logo style transfer*. To do this, the logos in the training dataset are clustered into an appropriate number of classes. Each cluster corresponds to a specific style of logos. Consider a logo belonging to cluster i. To convert it to a logo whose style corresponds to another cluster (say, j), the class transfer vector, $v = c_j - c_i$ is computed where c_i and c_j are centers of clusters i and j. This v is then added to the logo resulting in a new logo which theoretically would be similar to the original logo, but with the style of logos of cluster j. Such vector arithmetics need to be explored.

3. A very important computation required to practically implement all the ideas presented here is to find the input noise corresponding to a given image. This is like finding the inverse of the generator- a function that takes in an image and outputs it's corresponding noise. But the generator, when viewed as a function, might not even be one-one. This prevents it from being invertible. An alternate method of computing the input noise needs to be developed.

Bibliography

(1) I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, Advances in neural information processing systems, 2014, pp. 2672–2680.

(2) A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta and A. A. Bharath, "Generative Adversarial Networks: An Overview", *IEEE Signal Processing Magazine*, 2018, **35**, 53–65.

(3) T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, Advances in Neural Information Processing Systems, 2016, pp. 2234–2242.

(4) M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks", *arXiv preprint arXiv:1701.04862*, 2017.

(5) A. Radford, L. Metz and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks", *arXiv preprint arXiv:1511.06434*, 2015.

(6) X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang and S. P. Smolley, 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2813–2821.

(7) M. Arjovsky, S. Chintala and L. Bottou, "Wasserstein gan", *arXiv preprint arXiv:1701.07875*, 2017.

(8) I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin and A. C. Courville, Advances in Neural Information Processing Systems, 2017, pp. 5769–5779.

(9) A. Sage, E. Agustsson, R. Timofte and L. Van Gool, *LLD - Large Logo Dataset - version 0.1*, `https://data.vision.ee.ethz.ch/cvl/lld`, 2017.

Appendix A

Proofs and Examples

A.1 Jensen Shannon divergence and the loss function for GANs

As discussed earlier, the JS Divergence between two probability distributions p and q is,

$$D_{JS}(p,q) = D_{KL}(p, \frac{p+q}{2}) + D_{KL}(\frac{p+q}{2}, q) \tag{A.1}$$

where D_{KL} is Kullback-Leibler divergence. It was also observed that the loss function for GANs is $L(G,D)$ where,

$$L(G,D) = \int_x (p_r(x)\log(D(x)) + p_g(x)\log(1-D(x)))dx \tag{A.2}$$

Ian Goodfellow proved the following proposition in 2014. [1]

Proposition 1. *For a fixed G, the optimal Discriminator is,*

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{A.3}$$

Proof. While trying to maximize $L(G,D)$, D essentially maximizes the terms inside the integral,

$$p_r(x)\log(D(x)) + p_g(x)\log(1-D(x)) \tag{A.4}$$

For convenience, let's label,

$$y = D(x) \qquad a = p_r(x) \qquad b = p_g(x)$$

Now, the equation A.4 becomes,

$$a\log(y) + b\log(1-y)$$

Consider,

$$\begin{aligned} f(y) &= a\log(y) + b\log(1-y) \\ f'(y) &= a\frac{1}{\ln 10}\frac{1}{y} - b\frac{1}{\ln 10}\frac{1}{1-y} \\ &= \frac{1}{\ln 10}(\frac{a}{y} - \frac{b}{1-y}) \\ &= \frac{1}{\ln 10}\frac{a-(a+b)y}{y(1-y)} \end{aligned}$$

By setting $f'(y) = 0$, we get

$$y = \frac{a}{a+b}$$

i.e.,

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

□

The Jensen-Shannon divergence between p_r and p_g is,

$$\begin{aligned} D_{JS}(p_r, p_g) =& \frac{1}{2}D_{KL}\left(p_r, \frac{p_r+p_g}{2}\right) + \frac{1}{2}D_{KL}\left(p_g, \frac{p_r+p_g}{2}\right) \\ =& \frac{1}{2}\left(\log 2 + \int_x p_r(x)\log\frac{p_r(x)}{p_r(x)+p_g(x)}dx\right) \\ &+ \frac{1}{2}\left(\log 2 + \int_x p_r(x)\log\frac{p_g(x)}{p_r(x)+p_g(x)}dx\right) \\ =& \frac{1}{2}\left(2\log 2 + L(G, D^*)\right) \end{aligned}$$

Thus,

$$L(G, D^*) = 2D_{JS}(p_r, p_g) - 2\log 2$$

When both G and D are optimum, we get $p_g(x) = p_r(x)$, and hence $D^*(x) = \frac{1}{2}$.

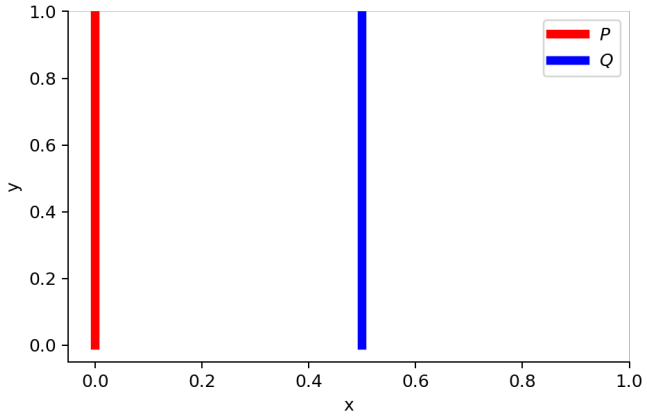


Figure A.1: When $\theta \neq 0$, P and Q are disjoint. *(Source:From GAN to WGAN, Lil'log)*

The loss function then becomes

$$\begin{aligned} L(G^*, D^*) &= \int_x (p_r(x) \log(D^*(x)) + p_g(x) \log(1 - D^*(x)))dx \\ &= \log \frac{1}{2} \int_x p_r(x)dx + \log \frac{1}{2} \int_x p_g(x)dx \\ &= -2 \log 2 \end{aligned}$$

since $\int_x p_r(x)dx = \int_x p_g(x)dx = 1$.

Hence,

$$L(G, D^*) - L(G^*, D^*) = 2D_{JS}(p_r, p_g) \tag{A.5}$$

where G^* and D^* are optimal generator and discriminator respectively. This is the relation between the JS Divergence and the Loss function for GANs.

A.2 Wasserstein distance between two distributions with disjoint supports: An example

Even if the two distributions have disjoint support, wasserstein distance still provides a smooth and meaningful representation of their distance.

As shown in figure A.1, consider two distributions P and Q where,

$$\forall (x, y) \in P, x = 0 \text{ and } y \sim U(0, 1) \forall (x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0, 1)$$

A.2.1 When $\theta \neq 0$:

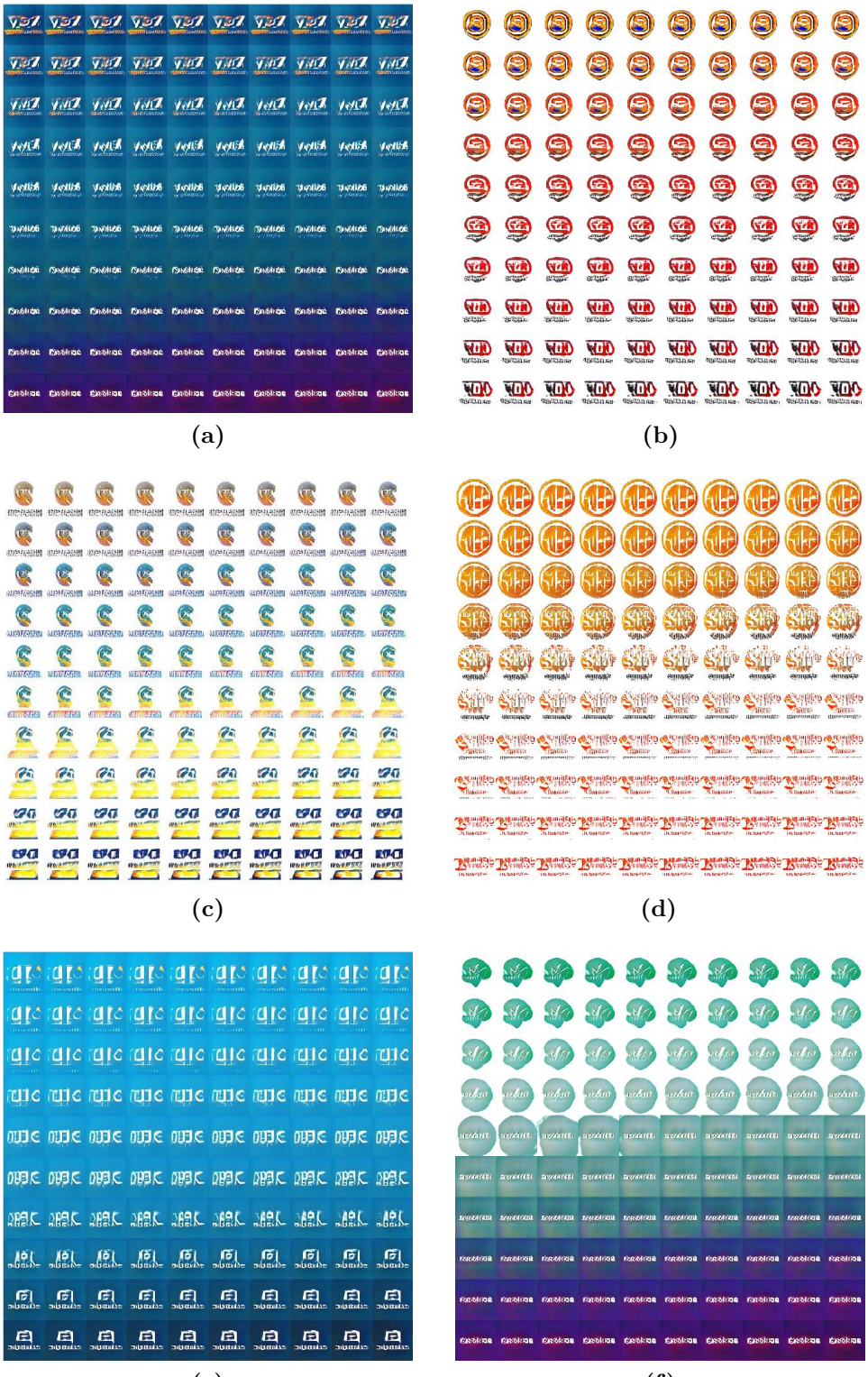
$$
\begin{aligned}
D_{KL}(P\|Q) =& \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty \\
D_{KL}(Q\|P) =& \sum_{x=\theta, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty \\
D_{JS}(P,Q) =& \frac{1}{2} \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} \\
&+ \frac{1}{2} \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} \\
=& \log 2 \\
W(P,Q) =& |\theta|
\end{aligned}
$$

A.2.2 When $\theta = 0$:

$$
\begin{aligned}
D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P,Q) = 0 \\
W(P,Q) = 0 = |\theta|
\end{aligned}
$$

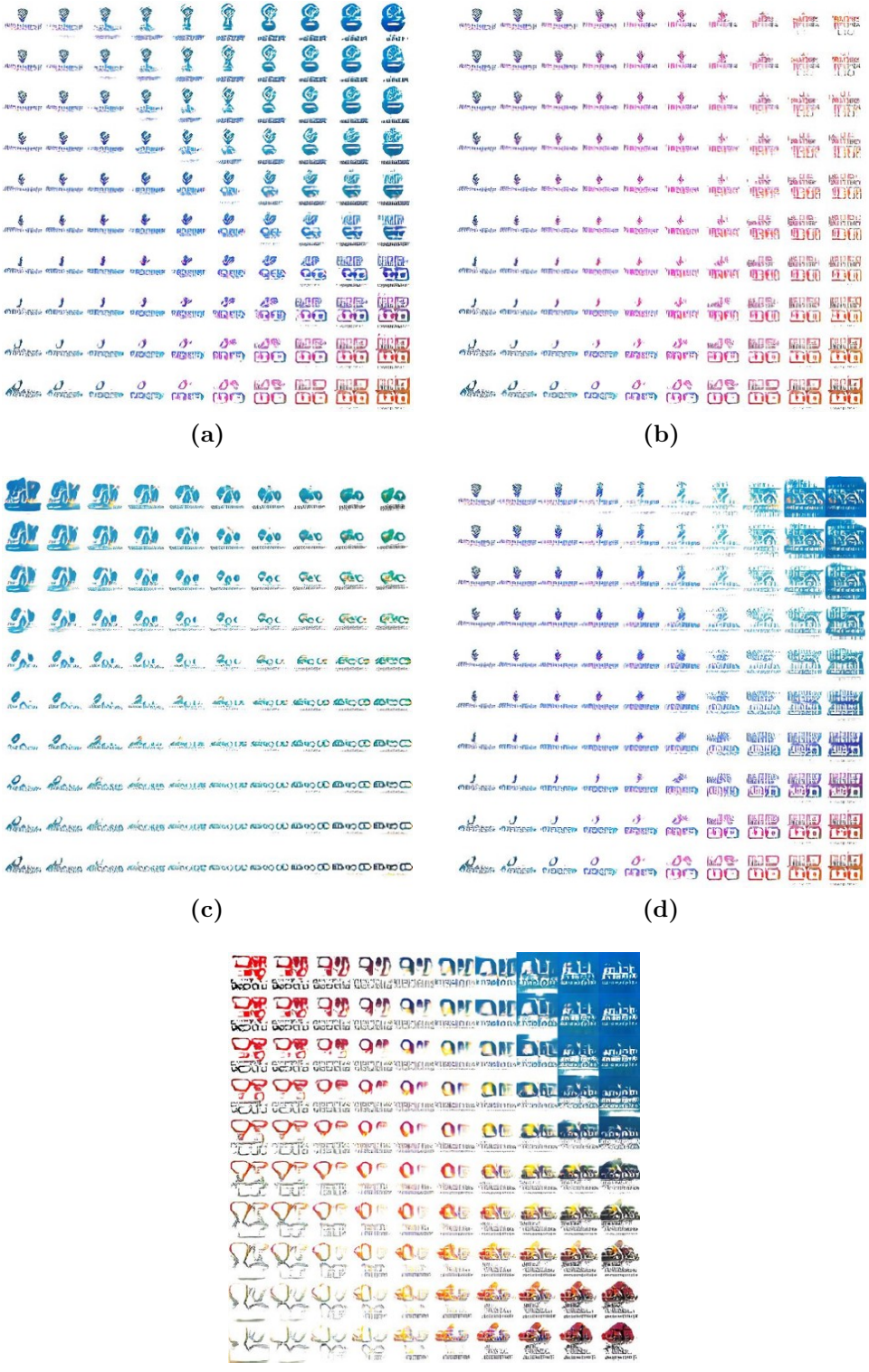
When the two distributions are disjoint, D_{KL} is infinite, and D_{JS} in discontinuous. But Wasserstein distance provides a smooth and helpful metric for a stable learning.

A.3 Interpolation: more examples



(a) (b) (c) (d) (e) (f)

Figure A.2: Interpolation between two images



(a) **(b)** **(c)** **(d)** **(e)**

Figure A.3: Interpolation among four images