Augmenting GAN with continuous depth Neural ODE

A thesis submitted for the partial fulfillment of
the conditions for the award of the degree **M.Tech. Computer Science.**

by

Love Varshney
Roll No: CS1711

Supervised by:
Prof. Sushmita Mitra
Machine Intelligence Unit

Indian Statistical Institute
Kolkata, India

July, 2019

To my family and the professors of ISI...

Certification

This is to certify that the dissertation entitled **"Augmenting GAN with continuous depth Neural ODE"** submitted by **Love Varshney (CS1711)** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of degree **Master of Technology (M.Tech) in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Prof. Sushmita Mitra
Machine Intelligence Unit
Indian Statistical Institute
Kolkata 700 108, INDIA

Acknowledgements

I would like to show my highest gratitude to my supervisor, Prof. Sushmita Mitra, Machine Intelligence Unit, Indian Statistical Institute, Kolkata, for accepting my request to work with her and her constant support and guidance. I want to thank Prof. B. Uma Shankar for his support and guidance. I also want to thank Subhashis Banerjee for his valuable comments.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work. Finally, I am very much thankful to my parents and family for their everlasting support.

Abstract

Generative adversarial networks are extremely powerful tools for generative modeling of complex data distributions. Research is being actively conducted towards further improving them as well as making their training easier and more stable. In this thesis, we present Neural ODE Generative Adversarial Network (NGAN), a framework that uses Neural ODE blocks instead of the standard convolutional neural networks (CNNs) as discriminators and generators within the generative adversarial network (GAN) setting. We show that NGAN outperforms convolutional-GAN at modeling image data distribution on MNIST dataset, evaluated on the generative adversarial metric.

Contents

List of Figures

Chapter 1

Introduction

Deep learning has made significant contributions to areas including natural language processing and computer vision. Most accomplishments involving deep learning use supervised discriminative modeling. However, the intractability of modeling probability distributions of data makes deep generative models difficult which makes generative modeling of data a very challenging and interesting machine learning problem. Image generation is one of the most difficult task in Computer Vision. Generative adversarial networks (GANs)[5] help alleviate this issue through setting a Nash Equilibrium between a generative neural network model (Generator) and a discriminative neural network (Discriminator). The discriminator is trained to determine whether its input is from a real data distribution or a fake distribution that was generated by the generative network.

Since the advent of GANs, many applications and variants[1, 8, 9, 14] have risen. Most of its applications are inspired by computer vision problems, and involve image generation as well as (source) image to (target) image style transfer. GANs have shown great promise in modeling highly complex distributions underlying real world data, especially images. However, they are notorious for being difficult to train and have problems with stability, vanishing gradients, mode collapse and inadequate mode coverage. Consequently, there has been a large amount of work towards improving GANs by using better objective functions [1, 10], sophisticated training strategies [16], using structural hyper parameters [14, 12] and adopting empirically successful tricks. In [14], authors provide a set of architectural guidelines, formulating a class of convolution neural networks (CNNs) that have since been extensively used to create GANs (referred to as Deep Convolution GANs or DCGANs) for modeling image data and other related applications.

1.1 Problem Statement

Generative Modeling of data is a challenging machine learning problem. Recently [5], introduced Generative Adversarial Networks for generating data. But, GANs are notoriously difficult to train and therefore there are less variety of model architectures known for GANs. We are improving GAN by augmenting them with Neural ODE.

1.2 Outline of This Thesis

In Chapter 2, we discuss preliminaries which includes GAN and Neural ODE. Chapter 3 discusses related work. Chapter 4 presents our work i.e. NGAN. We conclude and discuss future scope in Chapter 5.

Chapter 2

Preliminaries

In this chapter we will explain fundamentals behind GAN and Neural ODE. We started with explaining the fundamentals behind GAN and its training algorithm. Then, we explain the concept behind Neural ODE and forward and back propagation in Neural ODE.

2.1 GAN

Generative adversarial networks (GANs) are an example of generative models.The term "generative model" is used in many different ways. When talking about GAN, the term refers to any model that takes a training set, consisting of samples drawn from a distribution p_{data}, and learns to represent an estimate of that distribution somehow. This can be explicit or implicit. GANs focus primarily on sample generation. The basic idea of GANs is to set up a game between two players. One of them is called the generator. The generator creates samples that are intended to come from the same distribution as the training data. The other player is the discriminator. The discriminator examines samples to determine whether they are real of fake. The discriminator learns using traditional supervised learning techniques, dividing inputs into two classes (real or fake). The generator is trained to fool the discriminator. Generator is fed up with nooise z. The two players in the game are represented by two functions, each of which is differentiable both with respect to its inputs and with respect to its parameters. The discriminator is a function D that takes x as input and uses $\theta^{(D)}$ as parameters. The generator is defined by a function G that takes z(noise) as input and uses $\theta^{(G)}$ as parameters.

2.1.1 Cost function

Specificaaly, GAN solves the following minmax game:

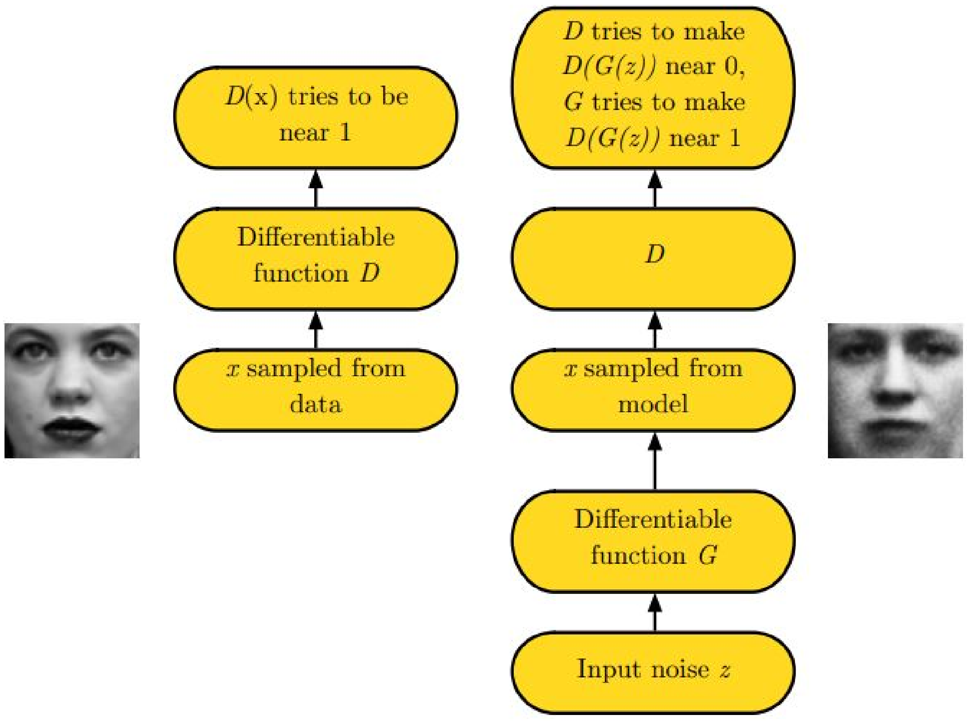
$$\min_G \max_D Loss(D,G) = \mathbb{E}_{x\sim Ps}[\log D(x)] + \mathbb{E}_{x\sim Pz}[\log(1 - D(G(z)))]$$

where Ps and Pz are sample and noise distribution; $G(z)$ is the geneartor that maps z to input space X; $D(x)$ is the discriminator that takes $x \in X$ and outputs a scaler between $[0,1]$. The meaning of this minmax cost function is that generator tries to fool the discriminator and discriminator tries to maximize the differentiation power between real and generated fake data. There are many versions of GAN[6] which slightly modifies this cost fuunction to achieve robustness and efficiency.

2.1.2 Training Algorithm

Algorithm 1 GAN

```
Require: Generator $G$ and Discriminator $D$, $\eta$ : the learning rate, $\beta_1$ and $\beta_2$ for Adam
    Optimizer, $m$ : batch size
Require: All parameters in $G$ and $D$ should be initialized
 1: procedure ADVERSARIAL TRAINING($G$,$D$)
 2:     for number of training iterations do
 3:         for number of minibatchs do
                                                              ▷ Train Discriminator D
 4:             Sample minibatch of $m$ noise samples $\mathbf{Z} = \{z^{(i)}\}_{i=1}^{m} \sim p(z)$ (noise prior)
 5:             Sample minibatch of $m$ examples $\mathbf{X} = \{x^{(i)}\}_{i=1}^{m} \sim p_{data}(x)$
 6:             Update the discriminator by ascending its stochastic gradient:
 7:             $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m}[logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$
                                                              ▷ Train Generator G
 8:             Sample minibatch of $m$ noise samples $\mathbf{Z} = \{z^{(i)}\}_{i=1}^{m} \sim p(z)$ (noise prior)
 9:             Update the generator by descending its stochastic gradient:
10:             $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m}[log(1 - D(G(z^{(i)})))]$
11:         end for
12:     end for
13: end procedure
```



$D(x)$ tries to be near 1
Differentiable function D
x sampled from data
D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

Figure 2.1: GAN training (Source [6])

2.1.3 Issues

It is well-known that the training GAN is difficult. In particular, the authors in [6] have identified the following sources of the difficulties:

- when the discriminator becomes accurate, the gradient for generator vanishes (a popular fixation to reduce the effect is to use gradient updating in generator with $\mathbb{E}_{x \sim P_z}[-\log(D(G(z)))$
- when discriminator becomes poor, the gradient for generator contains less valuable information
- Sometimes generator G gets stuck at a point with producing limited varieties of samples or one sample repeatedly during or after training the GAN, called Mode Collapse
- Hard to find nash equilibrium since GAN is a non cooperative game
- No proper evaluation metric

2.2 Neural ODE

Residual networks build a series of transformations by learning the difference between two consecutive transformation hidden states:

$$h_{t+1} = h_t + f(h_t, \theta_t)$$

where $t \in \{0...T-1\}$, $h_t \in R^D$ and T is depth of residual network and D is dimension of hidden state i.e. number of neurons. This can be seen as Euler discretisation of a continuous transformation [11, 7, 15]. Now as we add more layers and take smaller steps, in limit we parameterize the continuous dynamics of hidden units using ODE:

$$\frac{d(h(t))}{dt} = f(h(t), \theta_t)$$

Here $h(0)$ is input layer and we have to find $h(T)$ for some T. In [2], authors gives a reverse mode differentiation of ODE initial value problem. Neural ODE have several benefits like memory efficiency, Adaptive computation, Parameter efficiency.

2.2.1 Forward Propagation

Forward propagation in a neural ode block can be done by solving a initial value problem. We can use a numerical approximation solver for that purpose.

$$\frac{\partial \boldsymbol{z(t)}}{\partial t} = f(\boldsymbol{z(t)}, \theta_t, t) \tag{2.1}$$

$$\boldsymbol{z(t_0)} = x \tag{2.2}$$

where x is input to NODE block. Now suppose we are using $ODEsolver()$ as our approximate initial value solver. This can use any method i.e. euler, runga-kutta etc. So, $\boldsymbol{z(t_1)}$ will be:

$$\boldsymbol{z(t_1)} = ODESolver(\boldsymbol{z(t_0)}, f(\boldsymbol{z(t)}, \theta_t, t), t_0, t_1)$$

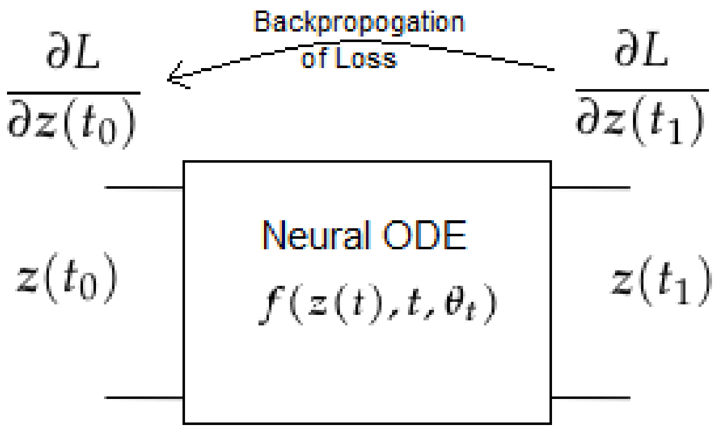

Backpropogation of Loss
$\frac{\partial L}{\partial z(t_0)}$
$\frac{\partial L}{\partial z(t_1)}$
Neural ODE
$f(z(t), t, \theta_t)$
$z(t_0)$
$z(t_1)$

Figure 2.2: Neural ODE (NODE) Block

2.2.2 Back Propagation

In a NODE block we can back propagate either through the operations of $ODESolver()$ or we can use algorithm 2 [2]. Back-propagation through operations of NODE block is time consuming and depends on the particular method used. In [2], authors presented a novel reverse-mode derivative of an ODE initial value problem. (we are assuming $\theta_t = \theta$ i.e. θ_t is constant function of t) (see algorithm 2)

2.2.3 Issues and Augmented Neural ODE

In [4], authors highlighted many problems in neural ode. For example, for arbitary d, let $0 < r_1 < r_2 < r_3$ and let $g : \mathbb{R}^d \to \mathbb{R}$ be a function such that:

$$g(x) = \begin{cases} -1 & \text{if } ||x|| \leq r_1 \\ 1 & \text{if } r_2 \leq ||x|| \leq r_3 \end{cases}$$

and proof that $g(x)$ can not be represented by a ODE transformation and to overcome that give a modified version called augmented neural ode.

Algorithm 2 Reverse-mode derivative of an ODE initial value problem

```
Require: $t_0$ : lower limit for ode integration
         $t_1$ : upper limit for ode integration
         output $\boldsymbol{z(t_1)}$
         loss gradient $\frac{\partial L}{\partial z(t_1)}$
         $d$ : dimension of input and output
         $n$ : size of $\theta$ i.e. number of parameters
         parameters $\theta$
Require: All parameters in NODE block should be initialized
 1: procedure AUGMENTDYNAMICS( $x, t, \theta$)
 2:     $\boldsymbol{z(t)} = x[1 : d]$
 3:     $\boldsymbol{a(t)} = x[d + 1 : 2 * d]$
 4:     return $[f(\boldsymbol{z(t)}, \theta, t), -\boldsymbol{a(t)}^T \frac{\partial f}{\partial z(t)}, -\boldsymbol{a(t)}^T \frac{\partial f}{\partial \theta}]$
 5: end procedure

 6: procedure REVERSE-MODE DERIVATIVE
                                                    ▷ $x$ is initial state of NODE block
 7:     $x[1 : d] = \boldsymbol{z(t_1)}$
 8:     $x[d + 1 : 2 * d] = \frac{\partial L}{\partial z(t_1)}$
 9:     $x[2 * d + 1 : 2 * d + n] = \mathbf{0}$
                        ▷ fill zeroes, this part represent gradient of $L$ w.r.t. $\theta$ at $t_1$
10:     $[\boldsymbol{z(t_0)}, \frac{\partial L}{\partial z(t_0)}, \frac{\partial L}{\partial \theta}] = ODESolver(x, augementDynamics, t_1, t_0, \theta)$
11:     return $\frac{\partial L}{\partial z(t_0)}, \frac{\partial L}{\partial \theta}$
12: end procedure
```

Chapter 3

Related Work

GANs were originally implemented as feed-forward multi-layer perceptrons, which did not perform well on generating complex images. They suffered from mode collapse and were highly unstable to train [14, 16]. In an attempt to solve these problems, [14] presented a set of guidelines to design GANs as a class of CNNs, giving rise to DCGANs, which have since been a dominating approach to GAN network architecture design. In [8], authors later proposed the use of Recurrent Neural Networks instead of CNNs as generators for GANs, creating a new class of GANs referred to as Generative Recurrent Adversarial Networks or GRANs. On a related note, [13] proposed an architectural change to GANs in the form of a discriminator that also acts as a classifier for class-conditional image generation. This approach for designing discriminators has been a popular choice for conditional GANs [12] recently. These are all architectural changes in Original GAN. We are also proposing an architectural change in GAN by augmenting them with Neural ODE.

3.1 DCGAN

Most GANs today are at least loosely based on the DCGAN architecture [14]. DCGAN stands for "Deep Convolution GAN". Though GANs were both deep and convolutional prior to DCGANs [3], the name DCGAN is useful to refer to this specific style of architecture. Some of the key insights of the DCGAN architecture were to:

- Use batch normalization layers after most layers of both the discriminator and generator, with the two mini-batches for the discriminator normalized separately. The last layer of the generator and first layer of the discriminator are not batch normalized, so that the model can learn the correct mean and scale of the data distribution.
- The overall network structure is mostly borrowed from the all-convolutional net. This architecture contains neither pooling nor "un-pooling" layers. When the generator needs to increase the spatial dimension of the representation it uses transposed convolution with a stride greater than 1.
- The use of the Adam optimizer rather than SGD with momentum.

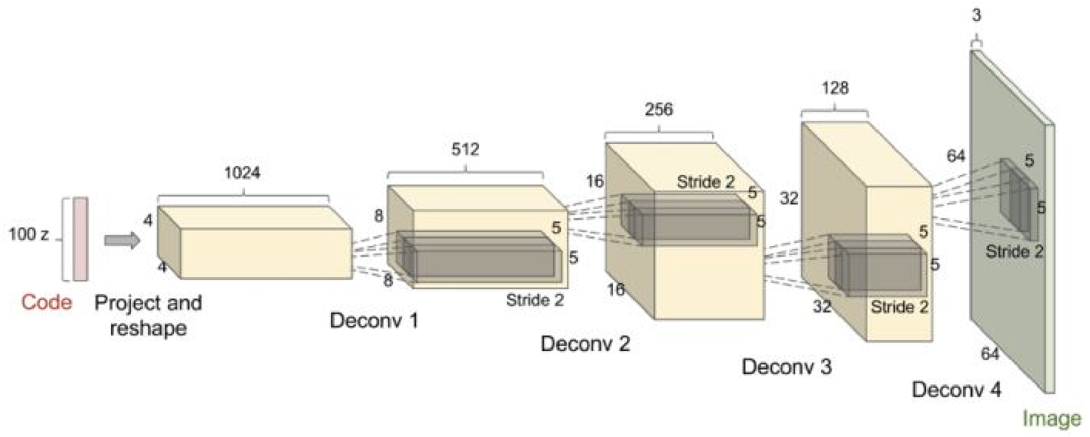


Figure 3.1: Generator Architecture in DCGAN (Source [6])

3.2 GRAN

In [8], Generative Recurrent Adversarial Networks(GRAN) has been proposed. The main difference between GRAN and other generative adversarial models is that the generator G consists of a recurrent feedback loop that takes a sequence of noise samples drawn from the prior distribution $z \sim p(z)$ and draws an output at multiple time steps $\boldsymbol{\Delta C_1, \Delta C_2, ..., \Delta C_T}$. Accumulating the updates at each time step yields the final sample drawn to the canvas C. At each time step t, a sample z from the prior distribution

$p(z)$ is passed to a function f along with the hidden states $\boldsymbol{h_{c,t}}$. Where $\boldsymbol{h_{c,t}}$ represent the hidden state, or in other words, a current encoded status of the previous drawing $\boldsymbol{\Delta C_{t-1}}$. Here, $\boldsymbol{\Delta C_t}$ represents the output of function f. Henceforth, the function g can be seen as a way to mimic the inverse of function f.

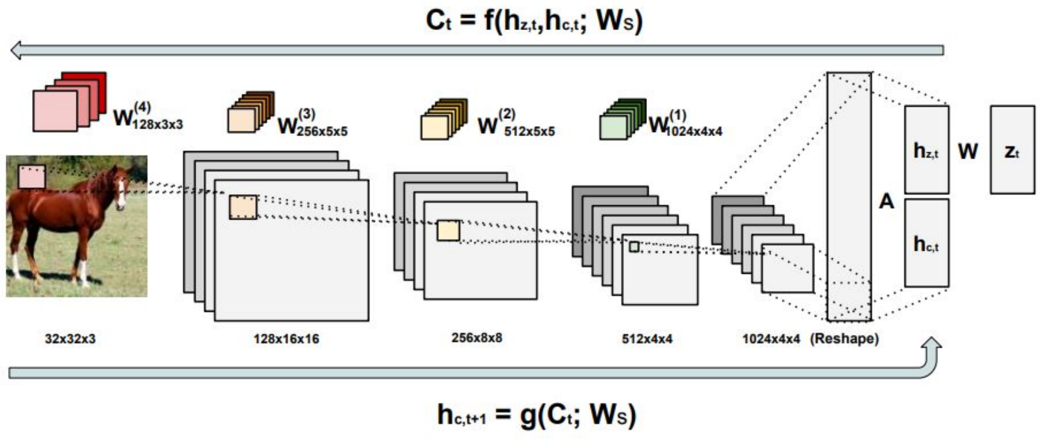


Figure 3.2: Generative Recurrent Adversarial Networks architecture (Source [8])

We have an initial hidden state $\boldsymbol{h_{c,0}}$ that is set as a zero vector in the beginning. We then compute the following for each time step $t = 1....T$:

$$\begin{aligned} z &\sim p(z) && (3.1)\\ \boldsymbol{h_{c,t}} &= g(\boldsymbol{\Delta C_{t-1}}) && (3.2)\\ \boldsymbol{h_{z,t}} &= \tanh W\boldsymbol{z_t} + b && (3.3)\\ \boldsymbol{\Delta C_t} &= f([\boldsymbol{h_{z,t}}, \boldsymbol{h_{c,t}}]) && (3.4) \end{aligned}$$

where $[\boldsymbol{h_{z,t}}, \boldsymbol{h_{c,t}}]$ denotes the concatenation of $\boldsymbol{h_{z,t}}$ and $\boldsymbol{h_{c,t}}$. Finally, we sum the generated images and apply the logistic function in order to scale the final output to be in $(0, 1)$:

$$\boldsymbol{C} = \sigma(\sum_{t=1}^{T} \boldsymbol{\Delta C_t})$$

3.3 Conditional GAN

In an unconditioned generative model, there is no control on modes of the data being generated. In the Conditional GAN(CGAN) [12], the generator learns to generate a

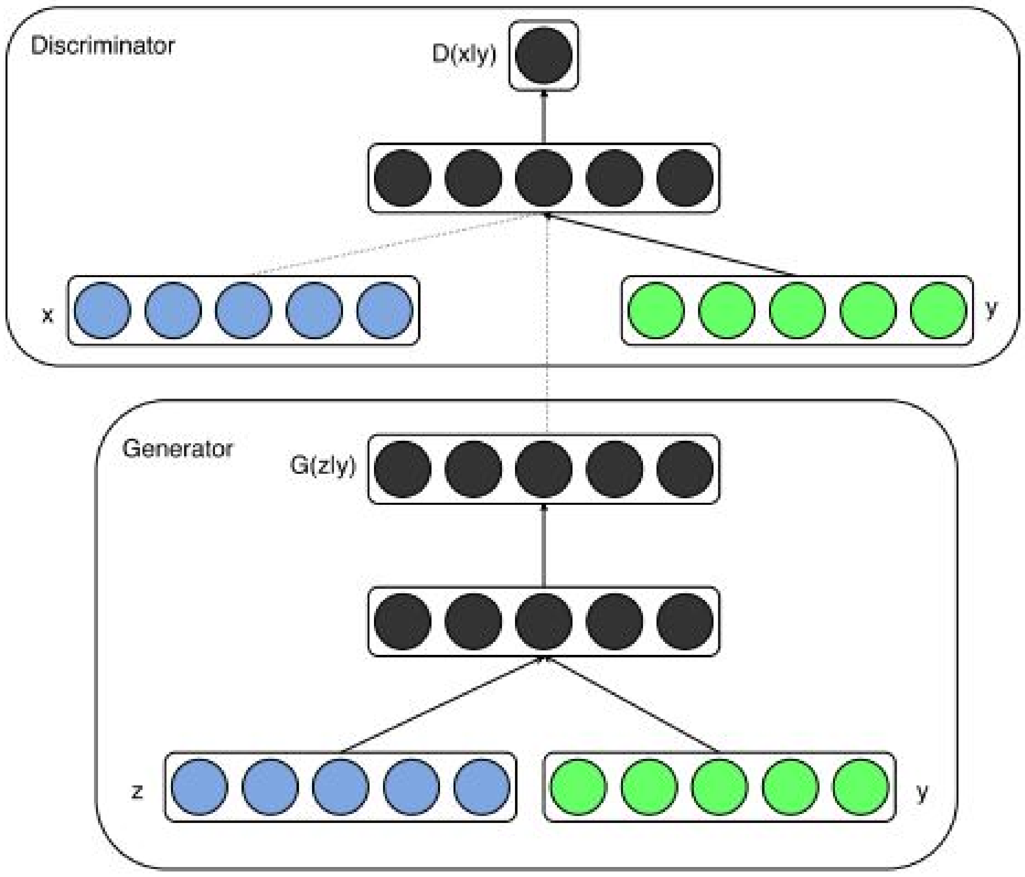


Figure 3.3: Conditional GAN (Source [12])

fake sample with a specific condition or characteristics rather than a generic sample from unknown noise distribution.

Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. y could be any kind of auxiliary information, such as class labels or data from other modalities. Authors perform the conditioning by feeding y into both the discriminator and generator as additional input layer. In the generator the prior input noise $p(z)$ and y are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. In the discriminator x and y are presented as inputs and to a discriminative function (embodied again by a MLP in this case). The objective function of a two-player min-max game would be:

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

3.4 Capsule GAN

In [9], authors proposed CapsuleGAN framework to incorporate capsule-layers instead of convolutional layers in the GAN discriminator, which fundamentally performs a two-class classification task. The final layer of the CapsuleGAN discriminator contains a single capsule, the length of which represents the probability whether the discriminator's input is a real or a generated image. We use margin loss L_M instead of the conventional binary cross-entropy loss for training our CapsuleGAN model because L_M works better for training CapsNets. Therefore, the objective of CapsuleGAN can be formulated as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{data}}[-L_M(D(x), T = 1)] + \mathbb{E}_{z\sim P_z}[-L_M(D(x), T = 0)]$$

Chapter 4

The Proposed Method

Generative Modeling of data is a challenging machine learning problem. Recently [5], introduced Generative Adversial Networks for generating data. But, GANs are notoriously difficult to train and therefore there are less variety of model artitectures known for GANs. We are improving GAN by augmenting them with Neural ODE. In this thesis, we used DCGAN [14] as a benchmark for us due to its popularity and we propose to change the DCGAN architecture with Neural ODE based architecture. We perform experiments on image generation with MNIST data.

4.1 Neural ODE GAN (NGAN)

For NAGAN, the model follow guidelines given in [14] paper by including batch normalization and relu layers in generator and leaky relu in discriminator. Architecture includes Neural ODE block with Convolution blocks defining the derivative in ODE. In [9], only discriminator architecture has been changed without changing generator architecture. We proposed to change both CNN based architectures into a combination of CNN and Neural ODE based architectures. Both generator and discriminator architectures involve 2-D Transpose Convolution and 2-D Convolution layers respectively. The basic idea is to use some Neural ODE Block in these architectures.

Algorithm 3 NGAN algorithm

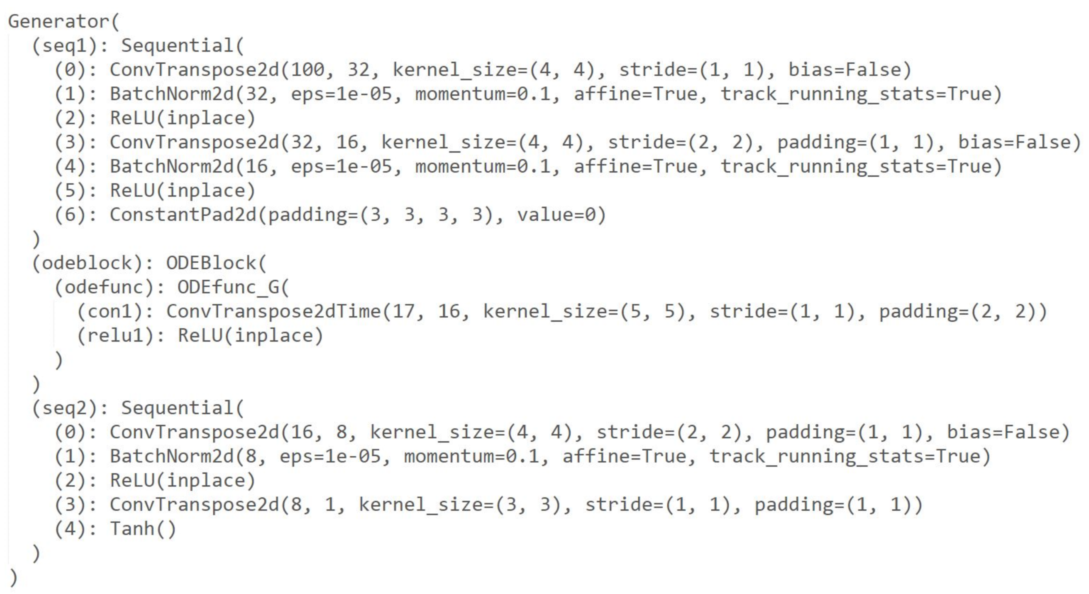
```
Require: NODE based Generator $G$ and Discriminator $D$
         $\eta$ : the learning rate
         $\beta_1$ and $\beta_2$ for Adam Optimizer.
         $m$ : batch size
         $tol$ : tolerance for ode Solver                    ▷ for NODE Block
         $t_0$ : lower limit for ode integration
         $t_1$ : upper limit for ode integration
Require: All parameters in $G$ and $D$ should be initialized
 1: procedure FORWARD($N$, $x$)                              ▷ N is a NODE based neural net
 2:     $L$ : number of layers in $N$
 3:     $z(i)$: output of $ith$ layer in $N$ and $z(0) = x$ (input to $N$)
 4:     for i ← 1 to $L$ do
 5:         if $ith$ layer is a NODE Block then
 6:             $z(i) = ODESolve(z(i-1), f, t_0, t_1, tol)$    ▷ $f$ is the func used in $ith$ layer
 7:         else
 8:             $z(i)$ is the forward propagation as in standard NN layer
 9:         end if
10:     end for
11: end procedure

12: procedure ADVERSARIAL TRAINING($G$,$D$)
13:     for number of training iterations do
14:         for number of minibatchs do
                                                             ▷ $D(x)$= FORWARD($D$, $x$) and
                                                ▷ $D(G(z))$=FORWARD($D$, FORWARD($G$, $z$))
                                                             ▷ Train Discriminator D
15:             Sample minibatch of $m$ noise samples $\mathbf{Z} = \{z^{(i)}\}_{i=1}^{m} \sim p(z)$ (noise prior)
16:             Sample minibatch of $m$ examples $\mathbf{X} = \{x^{(i)}\}_{i=1}^{m} \sim p_{data}(x)$
17:             $grad_{\theta_d} \leftarrow -\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$
18:             $\theta_d \leftarrow \theta_d - \eta *$Adam($\theta_d$, $grad_{\theta_d}$, $\beta_1$, $\beta_2$)
                                    ▷ If $\theta_d$ comes from NODE block use algorithm 2 for update
                                                             ▷ Train Generator G
19:             Sample minibatch of $m$ noise samples $\mathbf{Z} = \{z^{(i)}\}_{i=1}^{m} \sim p(z)$ (noise prior)
20:             $grad_{\theta_g} \leftarrow -\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [log(D(G(z^{(i)})))]$
21:             $\theta_g \leftarrow \theta_g - \eta *$Adam($\theta_g$, $grad_{\theta_g}$, $\beta_1$, $\beta_2$)
                                    ▷ If $\theta_g$ comes from NODE block use algorithm 2 for update
22:         end for
23:     end for
24: end procedure
```

4.2 Experiments and Results

We evaluate the performance of NGAN at MNIST due to its simplicity. And we also compare the results with DCGAN both qualitatively and quantitatively.

4.2.1 MNIST dataset

The MNIST dataset consists of $28X28$ sized grayscale images of handwritten digits. No pre-processing has been done on images. In Neural ODE based generator architecture, we used only a single 2-D Transpose Convolution as ODE function. As suggested in [4], we have augmented neural ode by increasing the dimension of each channel with zero padding.
For generator architecture we used a simple ODE block that consists of only a single 2-D transpose convolution layer whose output also depends on time at which ODE evaluation has been done, to achieve this we have increased a channels of all t values filled, where t is time at which evaluation has been done. As recommended in [14] we have used relu and batch normalization in generator architecture. For discriminator

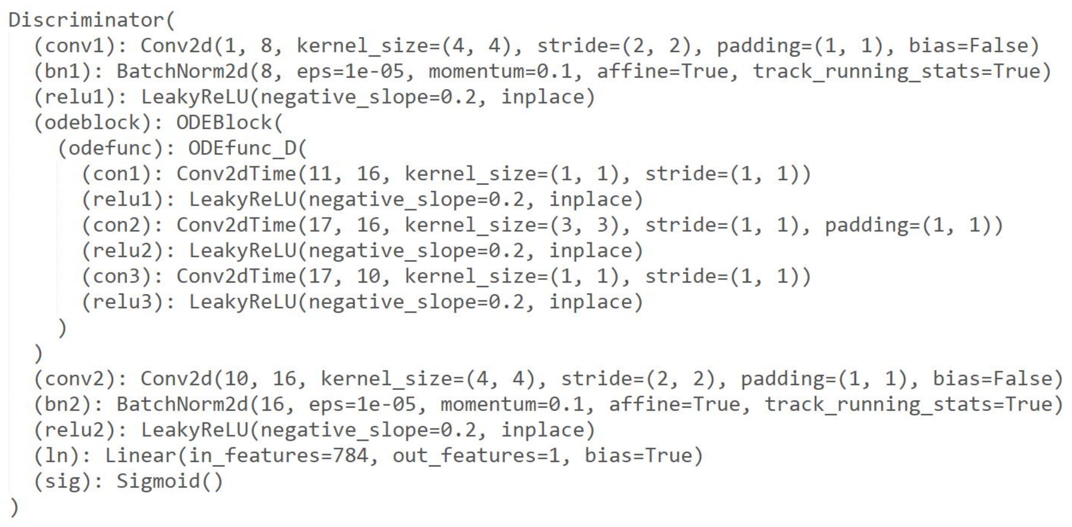


```
Generator(
  (seq1): Sequential(
    (0): ConvTranspose2d(100, 32, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(32, 16, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace)
    (6): ConstantPad2d(padding=(3, 3, 3, 3), value=0)
  )
  (odeblock): ODEBlock(
    (odefunc): ODEfunc_G(
      (con1): ConvTranspose2dTime(17, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (relu1): ReLU(inplace)
    )
  )
  (seq2): Sequential(
    (0): ConvTranspose2d(16, 8, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(8, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): Tanh()
  )
)
```

Figure 4.1: Generator Architecture

architecture we used a ODE block that consists of three 2-D convolution layer, each followed by a leaky relu layer. Also these convolution layers are also time dependent.

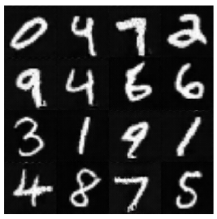
As recommended in [14] we have used leaky relu and batch normalization in discriminator architecture. For experiment, we have used runga-kutta method for solving ODE and back propagate from its operations.



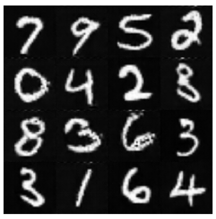
```
Discriminator(
  (conv1): Conv2d(1, 8, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): LeakyReLU(negative_slope=0.2, inplace)
  (odeblock): ODEBlock(
    (odefunc): ODEfunc_D(
      (con1): Conv2dTime(11, 16, kernel_size=(1, 1), stride=(1, 1))
      (relu1): LeakyReLU(negative_slope=0.2, inplace)
      (con2): Conv2dTime(17, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (relu2): LeakyReLU(negative_slope=0.2, inplace)
      (con3): Conv2dTime(17, 10, kernel_size=(1, 1), stride=(1, 1))
      (relu3): LeakyReLU(negative_slope=0.2, inplace)
    )
  )
  (conv2): Conv2d(10, 16, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): LeakyReLU(negative_slope=0.2, inplace)
  (ln): Linear(in_features=784, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

Figure 4.2: Discriminator Architecture

4.2.2 Visual Quality of randomly genearted images



(a) DCGAN Generated Images



(b) NGAN Generated Images

Figure 4.3: Randomly Generated Images

Qualitatively, both dcgan and ngan produce same quality images (even some images are exactly similar). As seen in figure 4.4 and 4.5, the divergence of loss is less in NGAN as compared to DCGAN. And in figure 4.6, we can see the number of forward evaluations in Generator and Discriminator of NGAN in training.

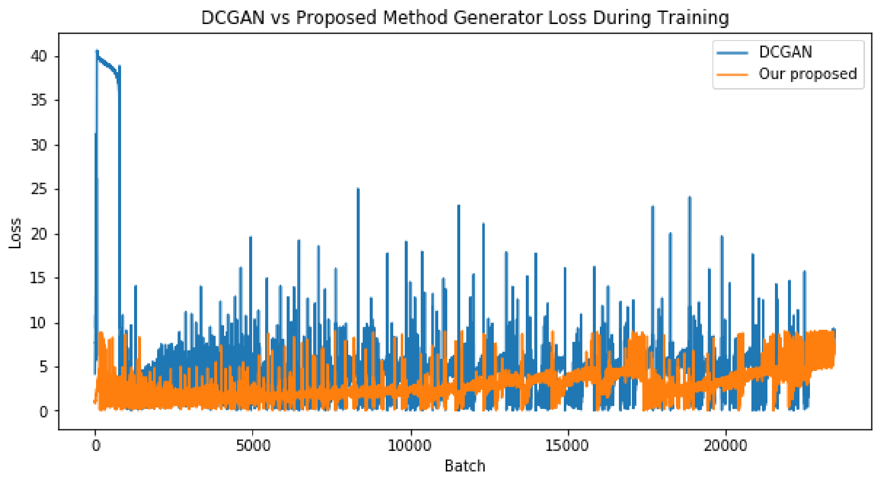


Figure 4.4: Generator Loss Comparison

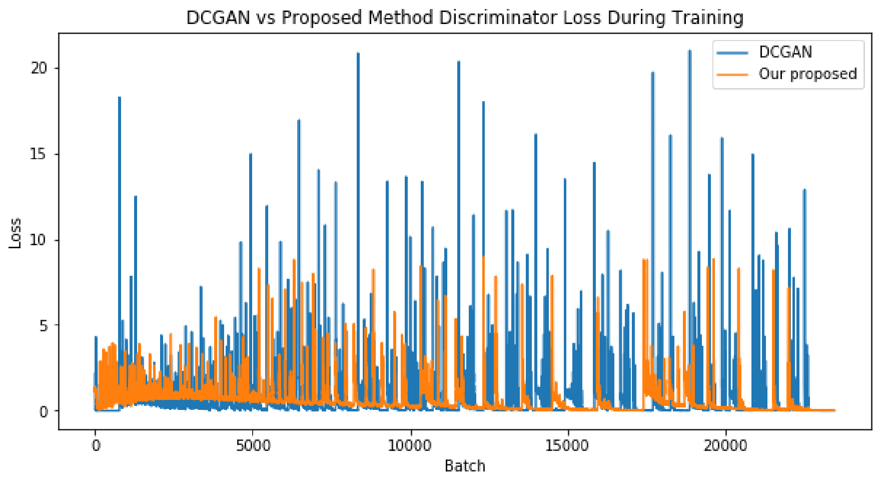


Figure 4.5: Discriminator Loss Comparison

4.2.3 Generative Adversarial Metric

In [8], authors introduced the generative adversarial metric (GAM) as a pairwise comparison metric between GAN models by pitting each generator against the opponent's discriminator, i.e., given two GAN models $M_1 = (G_1, D_1)$ and $M_2 = (G_2, D_2)$, G_1 engages in a battle against D_2 while G_2 against D_1. The ratios of their classification errors on real test dataset and on generated samples are then calculated as r_{test} and $r_{samples}$. Ratios of classification accuracy is considered instead of errors to avoid numerical problems:

$$\boldsymbol{r_{samples}} = \frac{Acc(D_{dcgan}(G_{ngan}))}{Acc(D_{ngan}(G_{dcgan}))}$$

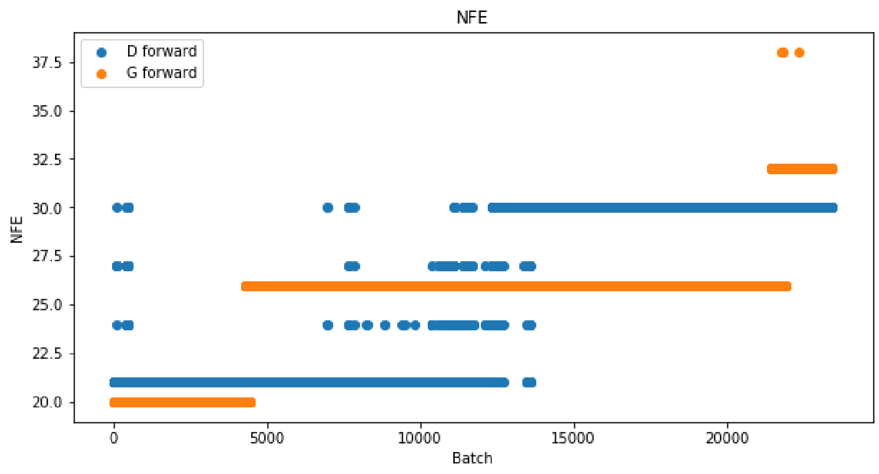


Figure 4.6: Number of forward evaluations (nfe) of G and D in NGAN

Then we take some unseen MNIST data $\boldsymbol{x_{test}}$ and calculated r_{test}:

$$r_{test} = \frac{Acc(D_{dcgan}(x_{test}))}{Acc(D_{ngan}(x_{test}))}$$

Therefore, for NGAN to win against DCGAN, both $\boldsymbol{r_{samples}} < 1$ and $\boldsymbol{r_{test}} \simeq 1$ must be satisfied. In our experiments, we achieve $\boldsymbol{r_{samples}} = 0.86$ and $\boldsymbol{r_{test}} = 1$ on the MNIST dataset. Therefore, NGAN working better than DCGAN on MNIST dataset.

Chapter 5

Conclusion and Future Scope

5.1 Discussion and Conclusion

Generative adversarial networks are extremely powerful tools for generative modeling of complex data distributions. Research is being actively conducted towards further improving them as well as making their training easier and more stable. In this thesis, we present Neural ODE Generative Adversarial Network (NGAN), a framework that uses Neural ODE blocks instead of the standard convolutional neural networks (CNNs) as discriminators and generators within the generative adversarial network (GAN) setting. While modeling image data, we show that NGAN outperforms convolutional-GAN at modeling image data distribution on MNIST dataset, evaluated on the generative adversarial metric. We have seen that NGAN outperform convolution based GAN on MNIST dataset. This indicates that NGAN can be used as a potential alternative to simple convolution based GAN.

5.2 Scope for Future Work

- Theoretically neural ode are more powerful than simple neural network. It would be useful to provide more theoretical analysis for how and why augmentation improves existing GANs.

- We have only used MNIST dataset to show the superiority of NGAN over simple convolutional-GAN, we can replicate experiments on more datasets like cifar etc.

- We can also compare the results of NGAN with more sophisticated versions of GAN
- Since we proposed a architectural change, neural ode based WGAN, MMD GAN can also be designed.

Bibliography

[1] ARJOVSKY, M., CHINTALA, S., AND BOTTOU, L. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning* (International Convention Centre, Sydney, Australia, 06–11 Aug 2017), D. Precup and Y. W. Teh, Eds., vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 214–223.

[2] CHEN, T. Q., RUBANOVA, Y., BETTENCOURT, J., AND DUVENAUD, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 6571–6583.

[3] DENTON, E. L., CHINTALA, S., SZLAM, A., AND FERGUS, R. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 1486–1494.

[4] DUPONT, E., DOUCET, A., AND TEH, Y. W. Augmented neural odes. *ArXiv abs/1904.01681* (2019).

[5] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[6] GOODFELLOW, I. J. NIPS 2016 tutorial: Generative adversarial networks. *CoRR abs/1701.00160* (2017).

[7] HABER, E., AND RUTHOTTO, L. Stable architectures for deep neural networks. *Inverse Problems 34*, 1 (dec 2017), 014004.

[8] IM, D. J., KIM, C. D., JIANG, H., AND MEMISEVIC, R. Generating images with recurrent adversarial networks. *CoRR abs/1602.05110* (2016).

[9] JAISWAL, A., ABDALMAGEED, W., WU, Y., AND NATARAJAN, P. Capsulegan: Generative adversarial capsule network. In *Workshop on Brain-Driven Computer Vision at European Conference on Computer Vision* (2018).

[10] LI, C.-L., CHANG, W.-C., CHENG, Y., YANG, Y., AND POCZOS, B. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 2203–2213.

[11] LU, Y., ZHONG, A., LI, Q., AND DONG, B. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *ArXiv abs/1710.10121* (2018).

[12] MIRZA, M., AND OSINDERO, S. Conditional generative adversarial nets. *CoRR abs/1411.1784* (2014).

[13] ODENA, A., OLAH, C., AND SHLENS, J. Conditional image synthesis with auxiliary classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning* (International Convention Centre, Sydney, Australia, 06–11 Aug 2017), D. Precup and Y. W. Teh, Eds., vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 2642–2651.

[14] RADFORD, A., METZ, L., AND CHINTALA, S. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (2016).

[15] RUTHOTTO, L., AND HABER, E. Deep neural networks motivated by partial differential equations. *ArXiv abs/1804.04272* (2018).

[16] SALIMANS, T., GOODFELLOW, I., ZAREMBA, W., CHEUNG, V., RADFORD, A., CHEN, X., AND CHEN, X. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2234–2242.