Generative Adversarial Networks with Neural Ordinary Differential Equations for Video Generation

A thesis submitted in part fulfilment of the degree of
Master of Engineering (*in Machine Learning and Computer Vision*)

by
Chinh Duc La
U7098799

Supervisor: Prof. Stephen Gould
Examiner: Dr. Liang Zheng

Australian National University

College of Engineering and Computer Science
The Australian National University

October 2021

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university. To the best of the author's knowledge, it contains no material previously published or written by another person, except where due reference is made in the text.

Chinh Duc La
29 October 2021

Acknowledgements

I want to express my greatest gratitude to my supervisor, Professor Stephen Gould, for his guidance and his original idea about Deep Declarative Network that sparked my research interest and for letting me explore my interest.

I would like to thank VinGroup Scholarship Program, without them, the poor boy never got a chance to chance to access such world-class education.

I very much appreciate Dylan Campbell and his course Advanced Topics in Computer Vision for letting me know the joy and depression of doing research. From that, I know how much I love doing research.

I want to thank Tran Anh Tuan, another member from VinGroup Scholarship Program, who helped me through the time I stay in Canberra.

To Nutthadech Banditakkarakul and Mingrui Zhao - two of a few friends that I made in the pandemic, to the couple Linh and Duc, who always try to help an introvert boy to connect with the outside world, thank you for being my friends in Australia.

To Hoang Tien Nhat Anh, Nguyen Tat Dat, Nguyen Dinh Hieu, Tran Minh Quang, Vu Bao Chau and my AhaMove squad for always being the entertainers I need.

Thank you Reddit and Google for always being a reliable sources of information. Thank you 9GAG for all the laugh and memes.

My family, thank you for always trying your best to support my dream and accepting my weirdness. I know I'm might not grown up to be the person that you wanted me to be, I disappointed you from time to time, but no matter who I am, you always be there to support me.

Finally, to my girlfriend, Nguyen Thi Bich Ngoc, thank you for loving me, for not giving up on me and for believing in me. This could be too early but "I couldn't have imagined, how good my life would get from the moment that I met you..."[1], Ngoc. So, **will you marry me?**

[1] "If I didn't have you" by Howard Wolowitz from The Big Bang Theory, Season 7, Episode 6

Abstract

Neural Ordinary Differential Equations (NODEs) (Chen et al., 2018) along with Deep Equilibrium Models (DEQ) (Bai et al., 2019) and Deep Declarative Networks (DDNs) (Gould et al., 2021) are a new class of layer in Deep Learning. Unlike the traditional Neural Network layers which are explicit functions where we can compute output directly from the input, the NODEs, DEQs and DDNs are expressed as an implicit function where it has a specific problem to solve. As a result, implicit layers are more expressive, not like a black box. In this thesis, we will review key concepts of NODEs, one of the subclasses of implicit layers class and investigate why NODEs is not popular in the application domain of Neural Network despite promising results from Chen et al. (2018). In addition, we will take a closer look at some problems relating to continuous dynamics in the context of Generative Adversarial Networks (GANs)(Goodfellow et al., 2014) training and video generation using GANs. We found that computational burden is a problem that prevents NODEs from being popular in Deep Learning and although continuous dynamics looks promising, it could be too simple to achieve what discrete dynamics could do. This result is shown clearly in video generation application. Finally, we found that continuous dynamics in training GANs suggest that mode collapse point and Nash equilibrium have same properties, hence, explain why mode collapse is a common problem in training GANs.

Contents

List of Figures

List of Tables

Nomenclature

ODEs	Ordinary Differential Equations
IVP	Initial Value Problem
NODEs	Neural Ordinary Differential Equations
SDEs	Stochastic Differential Equations
CDEs	Controlled Differential Equations
RK2	Runge-Kutta method with 2 stages or Heun's method
RK4	Runge-Kutta method with 4 stages
ResNet	Residual Network
VAE	Variational AutoEncoder
GANs	Generative Adversarial Networks
WGAN	Wasserstein Generative Adversarial Network
GD	Gradient Descent
ReLU	Rectifier Linear Unit
NN	Neural Networks
Adam	Adaptive momentum estimator
Neural GDEs	Graph Neural Ordinary Differential Equations
RNN	Recurrent Neural Network

Introduction

Machine Learning is one of the most blooming fields in recent years, especially in the subfields of Deep Learning and Neural Networks. Its applications span through many disciplines, from broad areas like Computer Vision or Natural Language Processing to more specific areas like Healthcare, Self Driving Cars or Entertainments. However, unlike many traditional Machine Learning models like Linear Regression, Logistics Regression,... which originally come from statistics and are interpretable, Neural Networks are considered black-box models. Because although neural network can approximate any function (Universal Approximation Theorem), it does not give any insight about the structure of the function being approximated. As a result, given a dataset, there can be two different neural networks with the same result, thus making it hard to analyse as we can only analyse from the empirical point of view, not from the theoretical side.

In this thesis, we consider a new class of neural networks. We call it the implicit neural network layer class. While conventional deep learning models are a stack of multiple explicit layers designed for particular tasks, implicit layers are concerned with solving the mathematical problems.. For example, conventional networks like Convolutional Networks usually consist of convolutional layers(Fukushima, 1980; LeCun et al., 1999), followed by an element-wise nonlinearity like Rectifier Linear Unit (ReLU) and end with additional operations like normalization (Ioffe and Szegedy, 2015; Miyato et al., 2018; Ulyanov et al., 2017) or dropout(Srivastava et al., 2014). These layers are usually designed to extract features from images in computer vision. They could be connected in multiple ways to form things like residual networks (He et al., 2016) or recurrent networks (Cho et al., 2014; Hochreiter and Schmidhuber, 1997; Rumelhart et al., 1986). Implicit layers, on the other hand, are usually designed to solve a mathematical problems. Deep Declarative Networks (DDNs)(Gould et al., 2021) is a class of layers that its output is a solution of an optimization problem. Output of Deep Equilibrium Networks (DEQs)(Bai et al., 2019) is the solution of a fixed point problem. Finally, in attempt to represent continuous dynamics in neural network through solving ordinary differential equations, Neural Ordinary Differential Equations (NODEs) (Chen et al., 2018) was born.

The implicit layer class changes the way we think about designing layers in neural networks. Instead of constructing a new network topology as an explicit forward function, we design layers based on the desired behavior or objective that we want them to have. Moreover, another advantage of implicit layers is that they do not have to store information of the computation graph during the forward pass to calculate gradients in the backward pass for fitting model. Implicit layers calculate the gradients directly using the Implicit Function Theorem (Krantz and Parks, 2002). However, one of the drawbacks is that we can rarely find the exact solution for these layers but low-error solutions based on the algorithm that we use to solve these objectives. Therefore, we always have low-level errors in these layers. But we can always evaluate the solution objectively by measuring how well they satisfy the conditions that the layers are trying to satisfy. However, despite the promising ideas of implicit layers, implicit layers still have not become a popular trend in deep learning research and

application. We will examine NODEs, a subclass of implicit layers, in this thesis to find the answer for its limitations.

Generative models is one class of machine learning models that aims to estimate the distribution p_{data} which generates data using observations from data. One of the earliest methods to estimate p_{data} is Gaussian Mixture Model (GMM). The distribution of data p_{data} is approximated as a mixture of normal distribution with parameters are learned using expectation-maximization (E-M) algorithm. However, GMM assumes that one observation is corresponding to one component of the mixture only. This leads to a problem in higher dimensional data space as GMM cannot easily learn since the size of model increases exponentially[1]. This drawback leads to the development of Boltzmann machine (Hinton and Sejnowski, 1983). Boltzmann machine is one of the first models that learns distributed representations.

The increasing computing power backed by graphical processing units and the amount of available data have led to the rise of deep learning models. As a result, deep generative models[2], such as variational autoencoders (VAEs)(Kingma and Welling, 2014) and generative adversarial networks (GANs)(Goodfellow et al., 2014), become state-of-the-art approaches for distribution representation learning. VAEs and GANs implicitly express the distribution density or we cannot provide an explicit formula for the distribution density they estimate. However, they, especially GANs[3], can generate realistic samples that are similar to real world data.

Yann LeCun, one of Turing Award[4] laureates, once described Generative Adversarial Network (Goodfellow et al., 2014) as "the most interesting idea in machine learning in the last ten years". GANs has been exploited in many applications like creating artwork creation (Zhu et al., 2016) or generating super-resolution images (Ledig et al., 2017). However, one of GANs' biggest disadvantages is that its learning process is unstable. (Qin et al., 2020) hypothesize that the instability comes from discrete training algorithm of deep learning. In this thesis, we will take a look at the continuous dynamics training process using ordinary differential equations solver in GANs context and compare it with the discrete training process.

In this work, we will discuss the original idea of NODEs and its development. Afterwards, we examine NODEs as a replacement for the residual network (He et al., 2016). Next, we investigate empirically continuous dynamics in training GANs. Finally, we try to use GANs and NODEs as an attempt to generate complex videos.

1.1 Main Contributions

The main contribution of this thesis is to provide an introduction to ordinary differential equations and their application in context of deep learning. Contributions of this thesis are the following:

1. We provide an example of how to replace a residual network with a NODEs. We also examine why doing so is not feasible in practice.

[1]For example, for a dataset MNIST (LeCun et al., 1999) (an observation is a 28×28 grayscale image) will require the covariance matrix to have 614656 parameters

[2]Generative models that use Neural Network with many hidden layer as a backbone.

[3]Reader can go to website https://thispersondoesnotexist.com/ that shows a random image generated by GANs every time we refresh it.

[4]Turing Award is generally recognized as the highest distinction in computer science and is often referred as "Nobel Prize of Computing"

2. We investigate the continuous dynamics training in context of GANs and give an explanation on why mode collapse is a common problem of GANs.

3. We combine NODEs with GANs in attempt to generate complex videos and explain NODEs problems in context of video generation.

1.2 Outline

The following is the outline of this thesis:

- In Chapter 2, we provide an overview of a background that is the foundation of the methodology of this thesis. We review ordinary differential equations, what are they and how to solve them numerically. We briefly review Neural Network, Residual Network the Gradient-Based Optimization methods. Then we discuss deep generative models and some of the state-of-the-art deep learning approaches in video generation.

- Neural Ordinary Differential Equations and the continuous dynamics will be the main subject of Chapter 3. We discuss ideas, theoretical aspects as well as some developments of NODEs. Then we provide in detail the experimental settings that we use for this thesis.

- In Chapter 4, we investigate the results of the experiments that we mentioned in Chapter 3 theoretically and empirically.

- We provide a short answer to all of our research questions and future work on Chapter 5.

Literature Review

This chapter provides background knowledge for the following chapters. It will give us a fresh look at ordinary differential equations, which will play a central role in this thesis, in both theoretical and numerical aspects. Next, we will briefly talk about the basics of neural networks and neural networks as generative models. Finally, we will go through some state-of-the-art approaches in video generation.

2.1 Ordinary Differential Equations

2.1.1 Definition and The Existence and Uniqueness Theorem

Ordinary Differential Equations (ODEs) play an important role in many disciplines like physics, economics or engineering as they usually represent the relationship between physical quantities and their rate of changes. One of most famous differential equations is the Navier-Stokes equations[1] that describe the motion of fluids (solving these equations is too important that the Clay Mathematics Institute put an award of 1 million USD to anyone who can solve it[2]).

An ordinary differential equation of order n is an equation of the form

$$\mathbf{z}^{(n)} = f(\mathbf{z}^{(n-1)}, \mathbf{z}^{(n-2)}, \dots, \mathbf{z}', \mathbf{z}, t) \tag{2.1}$$

where $\mathbf{z} : \mathbb{R}^d \to \mathbb{R}^d$ is a function of t and is n-time differential, $\mathbf{z}^{(i)} = \frac{d^i x}{dt^i}$. $t \in [t_0, T]$ usually represents the time domain with the time interval from t_0 to T. Function f is called the dynamics of Equation 2.1

$$f : \underbrace{\mathbb{R}^d \times \cdots \times \mathbb{R}^d}_{\text{n times}} \times [t_0; T] \to \mathbb{R}^d$$

If we given an ODE with an initial value $\mathbf{z}_0$, we refer it to the initial value problem (IVP). The first order IVP has the form

$$\begin{aligned} &\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}, t), \quad t \in [t_0, T] \\ &\mathbf{z}(t_0) = \mathbf{z}_0 \end{aligned} \tag{2.2}$$

When we study about ODEs, we also need to evaluate about their solutions. The following theorem about the solution of IVP is the foundation of ODE and the problem of our approach in Chapter 3.

Theorem 2.1. ***(Existence and Uniqueness Theorem)*** ***(Coddington and Levinson, 1955)*** *Suppose that $f(\boldsymbol{z}, t)$ is piecewise continuous in t and*

$$\|f(\boldsymbol{z}_1, t) - f(\boldsymbol{z}_2, t)\| \leq L\|\boldsymbol{z}_1 - \boldsymbol{z}_2\|$$

for some finite L, for all $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathbb{R}^d$ and for all $t \in [t_0, T]$. Then the IVP 2.2 has a unique solution for $t \in [t_0, T]$.

[1] Reader could refer to https://en.wikipedia.org/wiki/Navier-Stokes_equations that describe the if you want to know more about Navier Stocks equations

[2] It is also called the Millennium Prize Problem

Proof. Due to the scope of this thesis, we refer to Coddington and Levinson (1955) for a proof of theorem 2.1 and more theorem on ODEs. □

What theorem 2.1 means is that given a dynamics function $f(\mathbf{z}, t)$ that is L-Lipschitz continuous, then the solution of IVP 2.2 exists and it is unique.

2.1.2 Numerical Methods to Solve Ordinary Differential Equations

Solving ODEs is an important task but it is complicated. The Navier-Stocks Equations help us understand the fluid flow, but till this day noone actually finds an exact solution of it. As the result, in order to understand its solution, we have to use numerical methods to approximate its solution. In this section, we will review how to approximately solve ODEs using numerical method. For simplicity, we will consider first order ODEs only, but the ideal can be easily translated to n-order system. Given the IVP with $\mathbf{z}$ and t are defined like section 2.1.1:

$$\begin{aligned} \tfrac{d\mathbf{z}(t)}{dt} &= f(\mathbf{z}, t), \quad t \in [t_0, T] \\ \mathbf{z}(t_0) &= \mathbf{z}_0 \end{aligned}$$

We will try to find the approximate solution at any $t \in [t_0, T]$.

Euler's method: Euler's method approximates the solution of the IVP by discretizing the problem with assumption that $dt = h$ is a fixed number, $h = \frac{T - t_0}{N}$, N is the number of step time. We will have the discretised problem

$$\begin{aligned} \frac{\mathbf{z}_n - \mathbf{z}_{n-1}}{h} &= f(\mathbf{z}_{n-1}, t_0 + (n-1)h) \\ \mathbf{z}(t_0) &= \mathbf{z}_0 \end{aligned}$$

where $\mathbf{z}_n = z(t_0 + nh)$.

Hence we have the approximate solution of IVP at time $t_n = t_0 + nh$ as

$$\mathbf{z}_n = z_{n-1} + hf(\mathbf{z}_{n-1}, t_{n-1}), \forall n = 1, \dots, N. \tag{2.3}$$

The predefined h is called the *step size*. The method has local truncation error of $\mathcal{O}(h^2)$ and global truncation error (the accumulation of the local truncation error for multiple steps) of $\mathcal{O}(h)$.

Heun's method: Heun's method may be referred as improved or modified Euler's method by correcting the slope (value of f) by having an addition step. The approximate of the problem with step size given as above is given as

$$\tilde{\mathbf{z}}_{n-1} = \mathbf{z}_n + hf(\mathbf{z}_{n-1}, t_{n-1}) \tag{2.4}$$

$$\mathbf{z}_n = \mathbf{z}_{n-1} + \frac{h}{2}\left(f(\mathbf{z}_{n-1}, t_{n-1}) + f(\tilde{\mathbf{z}}_{n-1}, t_{n-1})\right) \tag{2.5}$$

Heun's method has local truncation error of $\mathcal{O}(h^3)$ and global truncation error of $\mathcal{O}(h^2)$.

Expicit Runge-Kutta methods: The family of explicit Runge-Kutta methods has the form

$$z_{n+1} = z_n + h\sum_{i=1}^{n} b_i k_i \tag{2.6}$$

where

$$k_1 = f(z_n, t_n) \tag{2.7}$$
$$k_2 = f(z_n + h(a_{2,1}k_1, t_n + c_2 h) \tag{2.8}$$
$$\vdots \tag{2.9}$$
$$k_n = f(z_n + h(a_{n,1}k_1 + \cdots + a_{n,n-1}k_{n-1}), t_n + c_n h) \tag{2.10}$$

with predefined n be number of stage, the coefficient a_{ij} $(1 \leq j \leq i \leq n)$. and c_i $(i = 2, 3, \ldots, n)$.

Heun's method is a special case of Runge-Kutta's method with 2 stages (RK2). In this work, we will use 4 stages Runge-Kutta method (RK4). The local truncation error of RK4 is $\mathcal{O}(h^5)$ and the global truncation error is $\mathcal{O}(h^4)$

2.2 Neural Networks

Neural Networks (NNs) are a class of algorithm which was inspired by the biological neural circuits in human brains but NNs has been swayed away from the original path (biological inspired) by mathematics and other disciplines in the last decade. We will cover a class of NNs that was inspired by game theory later in this chapter and another class of NNs which is an extension of ODEs in the Chapter 3. In this section, for the reviewing purpose and simplicity, we will describe NNs using one of simplest subclasses: the two-layer Feed-Forward Neural Networks.

2.2.1 Feed-Forward Neural Networks

Two-layer Neural Network: The class of two-layer Neural Networks with i-dimensional input and o-dimensional output is defined as

$$\mathcal{F}_2 = \{f : \mathbb{R}^i \to \mathbb{R}^o \mid \mathbf{x} \to \mathbf{W}_2 g(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2\} \tag{2.11}$$

Where i, o and h are the dimension of input, output and number of hidden neuron respectively, $\mathbf{W}_1 \in \mathbb{R}^{h \times i}$ and $\mathbf{W}_2 \in \mathbb{R}^{o \times h}$ are called weight matrices, $\mathbf{b}_1 \in \mathbb{R}^h$ and $\mathbf{b}_2 \in \mathbb{R}^o$ are called biases, g are activation functions. An example of a two-layer NNs is shown on figure 2.1 . Some common activation functions are shown in table 2.1.

Name	$g(x)$
Sigmoid	$\frac{1}{1+exp(-x)}$
Hyperbolic tangent	$\tanh(x)$
ReLU	$\max(x, 0)$
Softplus	$\log(1 + e^x)$
Leaky ReLU	$\begin{cases} x \text{ if } x \geq 0 \\ 0.01x, \text{ otherwise} \end{cases}$

Table 2.1: Some of common activation functions

One special theorem about two-layer NNs is that they could approximate any function. This property is also known as the Universal Approximation Theorem

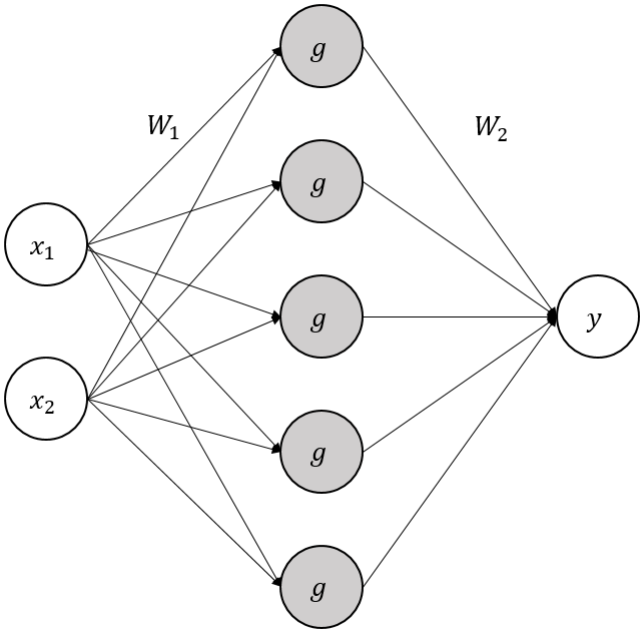

Figure 2.1: A two-layer NNs with input dimension is $i = 2$, output dimension is $o = 1$ and one hidden layer dimension is $h = 5$.

Theorem 2.2. *(Universal Approximation Theorem)(Cybenko, 1989; Hornik, 1991)(Informal) Given an activation function $g : \mathbb{R} \to \mathbb{R}$, for every continuous function $f : \mathbb{R}^i \to \mathbb{R}^o$, there exists a two-layer NNs $f_\varepsilon : \mathbb{R}^i \to \mathbb{R}^o$ $(\varepsilon > 0)$ with the form*

$$f_\varepsilon = \boldsymbol{W}_2 \circ g \circ \boldsymbol{W}_1 \tag{2.12}$$

such that $\|f(x) - f_\varepsilon(x)\| < \varepsilon$ everywhere. (Note that we write the form 2.12 as the compact form of 2.11)

Proof. Due to the scope of this thesis, we refer to Cybenko (1989); Hornik (1991) for a proof of Universal Approximation Theorem. □

The main problem of theorem 2.2 is that we do not know many hidden neurons we need to approximate the given function f. Later in Chapter 3, we will learn that not any class of NNs is universal approximation.

2.2.2 Residual Network

As we mentioned in the previous section, despite two-layer NNs being enough to approximate any function, we do not know how big the hidden layer should be to approximate a given function. Therefore, in practice, it's common to stack more layers instead of keep searching for the suitable number of hidden neurons, hence, increase the "depth" of NNs. However, deep neural networks suffer from degradation problem: "with the network depth increasing, the accuracy gets saturated (which might be unsurprising) and then degrades rapidly" (He et al., 2016, p. 770). The degradation is not caused by overfitting but the training error created by adding more layers. He et al. (2016) proposed a deep residual learning framework to solve this problem by adding identity mapping layer and let the network learn the residual function. They Residual Network (ResNet) block has the form of

$$\mathbf{x}_{out} = f(\mathbf{x}_{in}, \theta) + \mathbf{x}_{in} \tag{2.13}$$

where $\mathbf{x}_{in}$ and $\mathbf{x}_{out}$ are input and output vectors of the layer and $f(\mathbf{x}, \theta)$ is a the residual function with parameters θ that need to be learned.

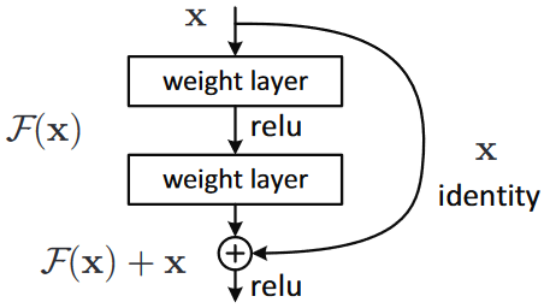

Figure 2.2: Residual Network Structure, $\mathcal{F}(\mathbf{x})$ is $f(x, \theta)$ in equation 2.13. Image from He et al. (2016)

2.3 Gradient Based Optimization

When we learn about training parameters of NNs, Backpropagation is arguably one of most dominant methods to solve this problem. The origins of it go back to the 1960s in context of control theory (Bryson, 1961; Kelley, 1960) but the term *backpropagation* was first used by Rumelhart et al. (1986). We assume that readers have knowledge about backpropagation algorithm and will go straight to gradient based optimizing algorithms which are usually used as a part of backpropagation algorithm.

Given an objective function $J(\theta)$ where θ is the parameters of NNs, we want to find the optimal parameters θ^* that minimize $J(\theta)$. We can write it as an optimization problem

$$\text{minimize}_\theta \quad J(\theta) \tag{2.14}$$

2.3.1 Gradient Descent

Gradient descent (GD) is an effective method to find local minimal point of objective function $J(\theta)$. It is generally a search algorithm guided by the gradient of the function at the current point. Therefore, the parameter are updated iteratively of the form

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta J(\theta_k) \tag{2.15}$$

where $k = 0, 1, \ldots$ and α_k is called the learning rate at time step k. Generally, in deep learning, the objective function at each time step will be calculate using observation data. However, when the number of observed data is too big (and observations might be high dimensional), calculating objective function $J(\theta)$ will be expensive. We often only use a minibatch of observation to calculate objective function $J(\theta)$, hence we have the unbiased[3] estimator $\nabla_\theta J_{minibatch}(\theta)$.

We have the new updated rule for parameters

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta J_{minibatch}(\theta_k) \tag{2.16}$$

The learning algorithm using only a mini-batch of observation like equation 2.16 is called Stochastic Gradient Descent, and the original learning algorithm 2.15 is called Batch Gradient Descent. When the learning rate α_k is a constant through the training process, in most cases, the GD algorithm will not converge. Therefore, we usually have a schedule to update the learning rate so the GD algorithm could converge.

[3]We call it unbiased because we expect its expectation to be the same quantity as the quantity that it estimate

2.3.2 Adam

Adam or adaptive moment estimator (Kingma and Ba, 2015) is one of the most popular optimizing methods in deep learning. Adam has a adaptive learning rate mechanics by estimating the mean and variance (or the first and second moment) of the gradient $g_k = \nabla_\theta J_{minibatch}(\theta_k)$. The update rule for Adam is

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k \tag{2.17}$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^2 \tag{2.18}$$

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k} \tag{2.19}$$

$$\hat{v}_k = \frac{v_k}{1-\beta_2^k} \tag{2.20}$$

$$\theta_{k+1} = \theta_k - \alpha \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \varepsilon} \tag{2.21}$$

2.4 Deep Generative Models

After reviewing about NNs and some algorithms to train NNs, we will give a brief review of generative models using NNs or the deep generative models, their advantages and problems, especially generative adversarial networks. Given an set of data $\mathcal{X} = \{\mathbf{x}^i\}_{i=1}^N$ consists of N data points, the goal of generative model is to learn the underlying structure of the dataset $\mathcal{X}$ in term of probability. So we could sample new data using the learned probability function.

2.4.1 Variational Autoencoder

Variational Autoencoder (VAE) (Kingma and Welling, 2014) is one of state-of-the-art generative models. VAE has the same structure with autoencoder (Kramer, 1991), it consists of 2 NNs, one is called encoder, the second is the decoder. The encoder's role is to learn the latent representation of input data, while the decoder reconstructs it as accurately as possible.

The key idea is that we can approximate any distribution in d dimension by transforming normal distribution with a sufficiently complicated function. Therefore, given a good function approximator, we could estimate any latent variables by mapping from an independent normally distributed z using learned function. Then, map the latent variables to X which is the random variable reprent data $\mathcal{X}$. We could write the distribution fucntion of X given latent variable z as follow

$$P(X|\mathbf{z};\theta) = \mathcal{N}(X|f(\mathbf{z},\theta), \sigma^2 I) \tag{2.22}$$

where $f(z,\theta)$ is a neural network with parameters θ and input $z \sim \mathcal{N}(0, I)$.

The objective of VAE consists of 2 objectives. One is to reconstruct the original input, the other is to make the latent space as close to Gaussian distribution with zero mean and uni variance as possible. The model structure is described in figure 2.3.

Reparameterization trick The left image of figure 2.3 is the original formulation of VAE, however, the stochastic sampling unit (red block) is non-differentiable, so we cannot use backpropagation to train the model. In order to make backpropagation feasible, the reparameterization trick is introduced, which instead of sampling $\mathbf{z}$ directly from $\mathcal{N}(\mu(X), \Sigma(X))$, we, firstly, sample ε from $\mathcal{N}(0, I)$ then use the property of Gaussian distribution to achieve $\mathbf{z}$

$$\mathbf{z} = \mu(X) + \Sigma(X) \odot \varepsilon \sim \mathcal{N}(\mu(X), \Sigma(X)) \tag{2.23}$$

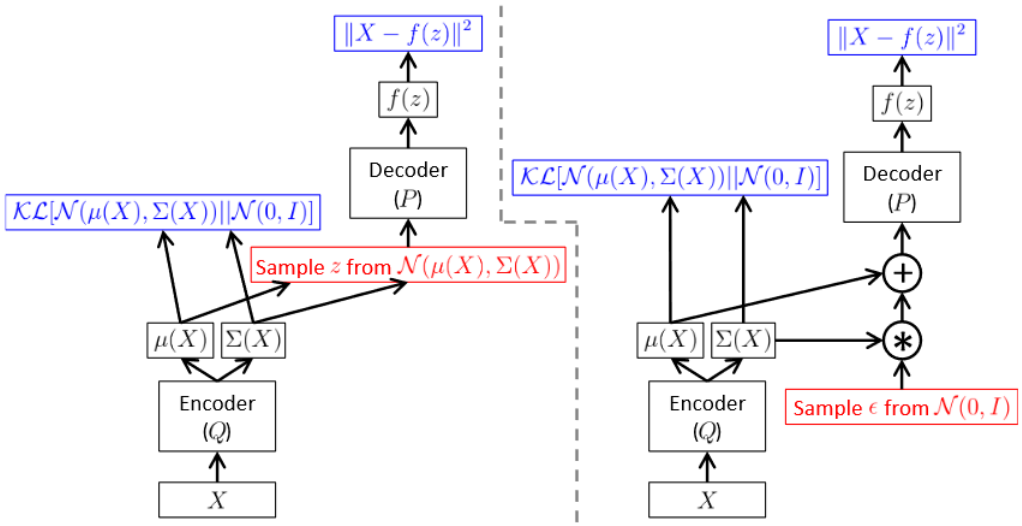

Figure 2.3: Left is without the "reparameterization trick", and right is with it. The random unit (red block) is non-differentiable. Image from Doersch (2021)

One advantage of VAE is that there are clear ways to evaluate the model. However, due to the noise injection and imperfect reconstruction, the generated data are tent to be blurred.

2.4.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) which is claimed as a special case of Artificial Curiosity (Schmidhuber, 2020) are based on the idea of a minimax game where two players try to compete with each other. One player is a generative model G or a neural network that maps input noises, $\mathbf{z} \sim p(\mathbf{z})$, from the latent space to data space. So, instead of learning an explicit probability distribution, G learns the underlying data distribution in an implicit way. The other is another neural network D that try to distinguish the real data and data generated from G. In other word, we want to learn the generator G and the discriminator D simultaneously or G and D play two-player minimax game with value function $V(G, D)$ defined as

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_Z(\mathbf{z})}[\log(1 - D(G(z)))] \tag{2.24}$$

Theorem 2.3. *(Goodfellow et al., 2014) The global minimum of the virtual training criterion* $C(G) = \max_D V(G, D)$ *is achieved if and only if* $p_G = p_{data}$*. At that point,* $C(G)$ *achieves the value* $-\log 4$*.*

Proof. We refer the proof of Theorem 2.3 to Goodfellow et al. (2014). □

Theorem 2.3 shows that there exists a model that could estimate the data distribution. However, like the Universal Approximation Theorem 2.2, it does not show how to find that model.

GANs is famous for creating realistic images (figure 2.4 show some high resolution sample images using a variant of GANs).

Figure 2.4: Some vocano sample images from Nguyen et al. (2017)

2.4.2.1 Vanilla GANs

GANs training using 2.24 is usually refered as Vanilla GANs (we will refer Vanilla GANs as GAN). Despite the success of GANs, training GANs is hard and unstable. It suffers from some problems as follows:

- **Convergence problem**: Goodfellow et al. (2014) has proof the convergence of GANs in minimax game, however, the proof shows that simultaneous gradient descent converges *in function space*. However, in practice, we update our neural nets in parameter space, so the convexity that the proof used does not apply. Goodfellow (2017) observed that one common form of non-convergence problem in GAN game is *mode collapse*. **Mode collapse** is a problem when the generator G choose to generate same outputs regardless of different input z values. Later in Section 4.2.2 we will see that mode collapse is also a convergence point but can only observed the convergence in continuous dynamics.

- **Internal covariance shift** is the state where the input distribution of network activation differs as a consequence of updating parameters in previous layers (Ioffe and Szegedy, 2015). This problem slows down the training process. Santurkar et al. (2018) believes mode collapse is one form of this problem and take a closer look into it to analyse internal covariance shift.

- **Vanishing gradient**: Arjovsky and Bottou (2017) suggested that when the discriminator is too good, the generator will fail to learn since the gradient is vanishing. Hence, optimal discriminator does not help the generator progress.

- **Lack of proper evaluation metrics**: although GANs have many successes in computer vision applications, it is still difficult to evaluate which method is better than others. Because we still don't have a standard procedure to evaluate them. One of most common evaluation metrics is Inception Score (Salimans et al., 2016) but Barratt and Sharma (2018) had pointed out some problems with Inception Score.

2.4.2.2 Modified GANs

Goodfellow et al. (2014) notes that the original GAN game use 2.24 as objective function could cause GAN to get stuck in early stage because the generator G's performance is poor in early stages,

discriminator D could distinguish samples generated by G easily because they are clearly different from the samples from the dataset. Goodfellow et al. (2014) suggest a modify version of generator loss as follows:

$$\begin{aligned} &\max_D \quad \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_Z(\mathbf{z})}[\log(1 - D(G(z)))] \\ &\max_G \quad \mathbb{E}_{\mathbf{z}}[\log D(G(z))] \end{aligned} \tag{2.25}$$

2.4.2.3 Wasserstein GAN

Arjovsky et al. (2017) proposed a modification of GAN where the discriminator does not actually classify input instance but try to make output bigger for real samples than for fake instances. Because the discriminator cannot discriminate, the discriminator of WGAN is actually called *critic*. The proposed loss functions are:

$$\begin{aligned} &\max_D \quad \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_Z(\mathbf{z})}[D(G(z))] \\ &\max_G \quad \mathbb{E}_{\mathbf{z}}[D(G(z))] \end{aligned} \tag{2.26}$$

WGAN has empirically shown that they are more stable in training.

2.4.3 Score-base Generative Model

Score-base generative model is proposed by Song and Ermon (2020a) that is based on denoising score matching with Annealed Langevin sampling. This approach could become a new contender of GANs in terms of image quality without adversarial requirements (as shown in figure 2.5). The authors has, since, improved their score-based model to be more stable and it could generate images with higher resolutions. Although this approach is still slow comparing to GANs in terms of real time generation, it shows promising results. However, this approach is out of the thesis's scope, so we refer readers to Song and Ermon (2020a,b); Song et al. (2021) for more information.

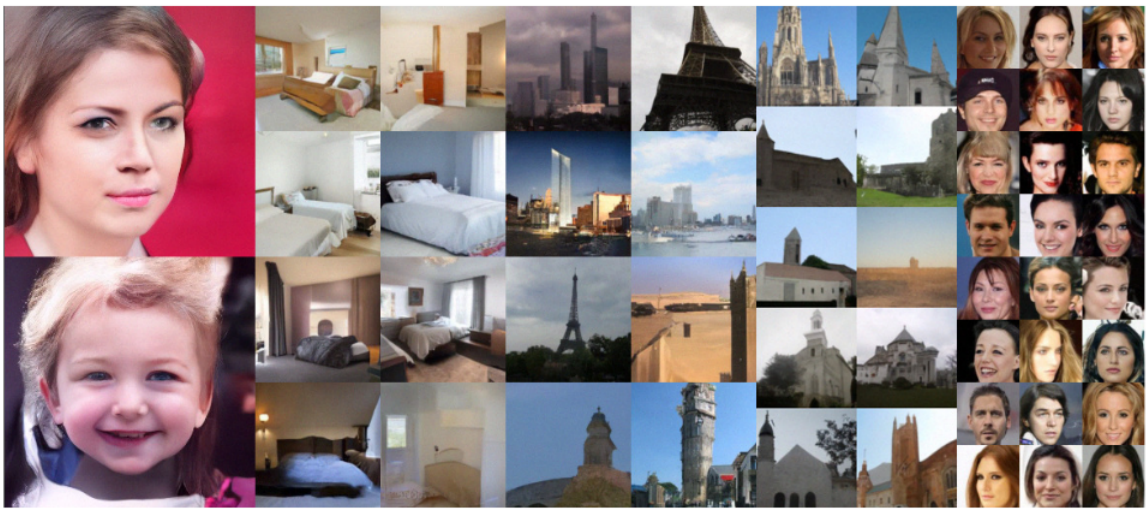

Figure 2.5: Generated samples of score-base model (Song and Ermon, 2020b)

2.5 Deep Learning and Video Generation

This section will discuss about the application of Deep Learning in Video Generation. To be more specific, we will only go into Video Generation GANs.

Video GAN (VGAN)(Vondrick et al., 2016) is the first model that extends GANs to video domain. VGAN used 2-stream generator. One stream is to generate background and the other is to generate moving foreground separately and then combine them into one frame of a video. VGAN uses one discriminator to solve 2 problems at a time: give generator information to learn realistic scenes and to learn realistic motions. Although the result shows that the generated background is relatively sharp, the motion object is lack of resolution.

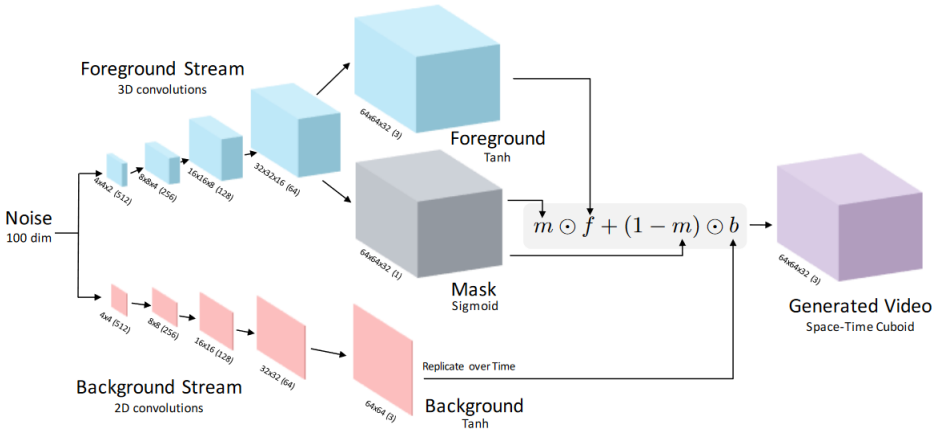

Figure 2.6: VGAN structure. Image from Vondrick et al. (2016)

Temporal Generative Adversarial Nets with singular value Clipping (TGAN-SVC) (Saito et al., 2017) is another early model that tries to generate videos. TGAN uses 2 generators. One is temporal generator, the other is image generator. This could be seen as the first model that tries to separate input latent into content latent and motion latent. However, since the content latent is fed into a temporal generator, with a fixed latent value, we will always have the same motion latent sequences. TGAN suffers from mode collapse when they don't input the same content latent into temporal generator.

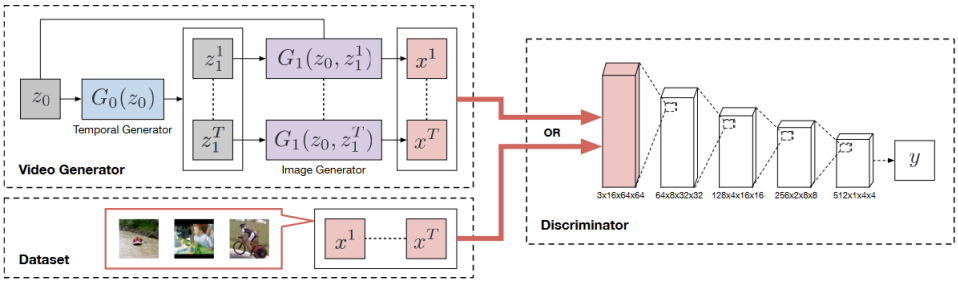

Figure 2.7: TGAN structure. Image from Saito et al. (2017)

Motion and Content GAN or MoCoGAN (Tulyakov et al., 2018) is the first model that *explicitly* try to separate latent input into content latent and motion latent. MoCoGAN is also the first approach that have 2 discriminators, one is for discriminating image and the other is for discriminating video. MoCoGAN uses a RNN to generate series of motion latent values and concatenate with content motion values and feed into an image generator. Yushchenko et al. (2019) tried to improve the original MoCoGAN by modifying video discriminator with Markov Decision Process. Gordon and Parde (2021) tried to replace RNN by ODEs continuous dynamics but they did not show the problem with continuous dynamics or any of their samples. TGANv2 (Saito et al., 2020) and Dual Video Discriminator GAN (DVD-GAN) (Clark et al., 2019) can be regarded as an extension of MoCoGAN

but they can generate high resolution images.

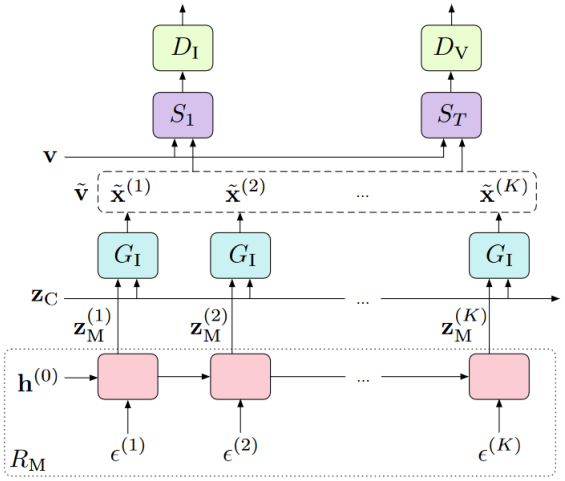

Figure 2.8: MoCoGAN structure. Image from Tulyakov et al. (2018)

ODE second order VAE or ODE^{2}VAE (Yildiz et al., 2019) is the first model that uses continuous dynamics to model data dynamics. ODE^{2}VAE has 2 encoders, one to encode the position latent of the start image frame in the video and the other is to encode the the velocity of the first m frame in the video. Then, use a Bayesian neural network to govern the acceleration and NODEs to find next latent position of next frame and reconstruct the original frame using the decoder. This methods has successfully reconstruct the next frame of simple video like Rotating MNIST.

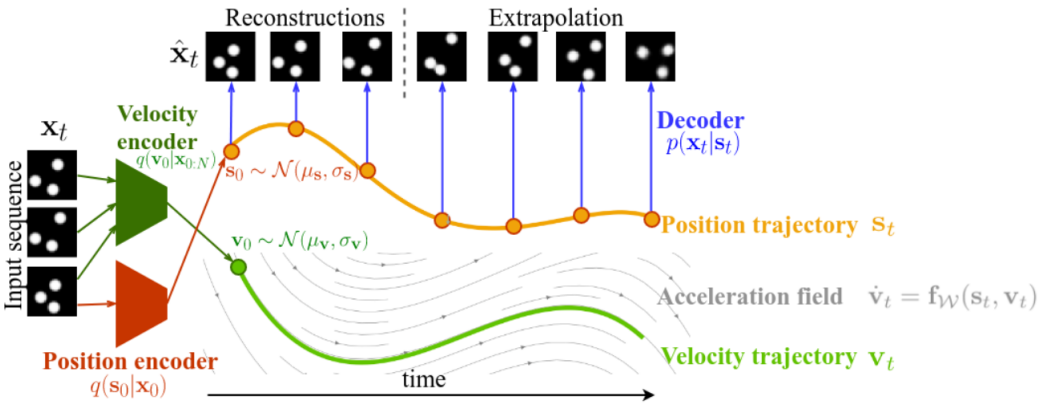

Figure 2.9: ODE^{2}VAE structure. Image from Yildiz et al. (2019)

Neural Ordinary Differential Equations

In this chapter, we will describe Neural Ordinary Differential Equation (NODEs), its variants and problems and we will show how we use it in this thesis in different problems: replacing ResNet, training GANs and generating temporal motion for MoCoGAN.

3.1 Neural Ordinary Differential Equations

3.1.1 Definition

The idea of NODEs (Chen et al., 2018) starts from ResNet. As we describe ResNet in section 2.2.2, the residual network block could be describe as

$$\mathbf{x}_{out} = \mathbf{x}_{in} + f(\mathbf{x}_{in}, \boldsymbol{\theta}) \tag{3.1}$$

When we rewrite equation 3.1 in time domain, we have

$$\mathbf{x}_{t+1} = \mathbf{x}_t + f(x_t, \boldsymbol{\theta}) \times 1 = \mathbf{x}_t + f(\mathbf{x}_t, \boldsymbol{\theta}) \times \Delta t \tag{3.2}$$

where $\Delta t = 1$ is the step size. Let $\Delta t \to 0$, we could express 3.2 as an IVP probem

$$\begin{aligned} \frac{d\mathbf{x}(t)}{dt} &= f(\mathbf{x}(t), t, \boldsymbol{\theta}), \quad t \in [0, T] \\ \mathbf{x}(0) &= \mathbf{x}_0 \end{aligned} \tag{3.3}$$

for some time T. When $f(\mathbf{x}(t), t, \boldsymbol{\theta})$ is a neural network with $\boldsymbol{\theta}$ as parameters, the IVP 3.3 becomes a NODEs problem.

Note that the parameters $\boldsymbol{\theta}$ is independent of t and f depends explicitly on t. ResNet, on the other hand, does not depend on t as it is a discretization version of ODEs.

3.1.2 Learning of Neural Ordinary Differential Equations

Learning of NODEs means we want to estimate the parameter $\boldsymbol{\theta}$ of the dynamics $f(\mathbf{x}(t), t, \boldsymbol{\theta})$ in the IVP 3.3. This can be done using the adjoint sensitivity method (Kaufman, 1964) and treat the ODE solver as a black-box. This method was proposed by Chen et al. (2018), it scale linearly with problem size, has low memory cost and we can control the numerical errors. This section will follows Chen et al. (2018)

Given a loss scalar-valued function L where

$$L(\mathbf{x}(t_1)) = L\left(\mathbf{x}(t_0) + \int_{t_0}^{t_1} f(\mathbf{x}(t), t, \boldsymbol{\theta}) dt\right) = L(\text{ODESolver}(\mathbf{x}(t_0), t_0, t_1, \boldsymbol{\theta})) \tag{3.4}$$

Proposition 3.1. *Define adjoint quantity* $\boldsymbol{a}(t) = \frac{\partial L}{\partial \boldsymbol{x}(t)}$*, we going to prove that*

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)\frac{\partial f(\boldsymbol{x}(t), t, \boldsymbol{\theta}))}{\partial \boldsymbol{z}(t)} \tag{3.5}$$

Proof. We have

$$\mathbf{x}(t+\varepsilon) = \int_t^{t+\varepsilon} f(\mathbf{x}(t), t, \boldsymbol{\theta})dt + \mathbf{x}(t) = T_\varepsilon(\mathbf{x}(t), t) \tag{3.6}$$

Therefore, applying chain rule we have

$$\frac{dL}{d\mathbf{x}(t)} = \frac{dL}{d\mathbf{x}(t+\varepsilon)}\frac{d\mathbf{x}(t+\varepsilon)}{d\mathbf{x}(t)} \tag{3.7}$$

$$\iff \mathbf{a}(t) = \mathbf{a}(t+\varepsilon)\frac{\partial T_\varepsilon(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)} \tag{3.8}$$

We have

$$\frac{d\mathbf{a}(t)}{dt} = \lim_{\varepsilon\to 0}\frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t)}{\varepsilon} \tag{3.9}$$

$$= \lim_{\varepsilon\to 0}\frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\frac{\partial}{\partial \mathbf{x}(t)}T_\varepsilon(\mathbf{x}(t))}{\varepsilon} \tag{3.10}$$

$$= \lim_{\varepsilon\to 0}\frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\frac{\partial}{\partial \mathbf{x}(t)}\left(\mathbf{x}(t) + \varepsilon f(\mathbf{x}(t), t, \boldsymbol{\theta}) + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} \tag{3.11}$$

$$= \lim_{\varepsilon\to 0}\frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\left(I + \varepsilon\frac{\partial f(\mathbf{x}(t), t, \boldsymbol{\theta})}{\partial \mathbf{x}(t)} + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} \tag{3.12}$$

$$= \lim_{\varepsilon\to 0}\frac{-\varepsilon\mathbf{a}(t+\varepsilon)\frac{\partial f(\mathbf{x}(t), t, \boldsymbol{\theta})}{\partial \mathbf{x}(t)} + \mathcal{O}(\varepsilon^2)}{\varepsilon} \tag{3.13}$$

$$= \lim_{\varepsilon\to 0} -\mathbf{a}(t+\varepsilon)\frac{\partial f(\mathbf{x}(t), t, \boldsymbol{\theta})}{\partial \mathbf{x}(t)} + \mathcal{O}(\varepsilon) \tag{3.14}$$

$$= -\mathbf{a}(t)\frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}(t)} \tag{3.15}$$

□

Similarly, when we consider $\boldsymbol{\theta}$ and t are also a function of t, we could write

$$\frac{d\boldsymbol{\theta}}{dt} = 0 \qquad \frac{dt(t)}{dt} = 1 \tag{3.16}$$

We can combine with $\frac{dx(t)}{dt} = f(x(t), t, \boldsymbol{\theta})$ to have an augmented state with corresponding adjoint state

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\theta}\\ t\end{bmatrix}(t) = \begin{bmatrix}f(\mathbf{x}, t, \boldsymbol{\theta})\\ 0\\ 1\end{bmatrix} = f_{aug}(x, t, \theta), \mathbf{a}_{aug} = \begin{bmatrix}\mathbf{a}\\ \mathbf{a}_{\boldsymbol{\theta}}\\ \mathbf{a}_t\end{bmatrix}(t) = \begin{bmatrix}\frac{dL}{d\mathbf{x}(t)}\\ \frac{dL}{d\boldsymbol{\theta}(t)}\\ \frac{dL}{dt(t)}\end{bmatrix} \tag{3.17}$$

So we have

$$\frac{d\mathbf{a}_{aug}(t)}{dt} = -\begin{bmatrix}\mathbf{a}(t) & \mathbf{a}_{\boldsymbol{\theta}}(t) & \mathbf{a}_t(t)\end{bmatrix}\frac{\partial f_{aug}}{\partial[\mathbf{x}, \boldsymbol{\theta}, t]} = -\begin{bmatrix}\mathbf{a}\frac{\partial f}{\partial \mathbf{x}} & \mathbf{a}\frac{\partial f}{\partial \boldsymbol{\theta}} & \mathbf{a}\frac{\partial f}{\partial t}\end{bmatrix}(t) \tag{3.18}$$

where $\frac{\partial f_{aug}}{\partial[\mathbf{x},\boldsymbol{\theta},t]}$ is the Jacobian of f, $\frac{\partial f_{aug}}{\partial[\mathbf{x},\boldsymbol{\theta},t]} = \begin{bmatrix}\frac{\partial f}{\partial \mathbf{x}} & \frac{\partial f}{\partial \boldsymbol{\theta}} & \frac{\partial f}{\partial t}\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$.

So, we have the gradients with respect to $x(t_0), x(t_N)$ from

$$\frac{dL}{d\mathbf{x}(t_0)} = \mathbf{a}(t_0) = \mathbf{a}(t_N) + \int_{t_N}^{t_0}\frac{d\mathbf{a}(t)}{dt}dt = \mathbf{a}(t_N) - \int_{t_N}^{t_0}\mathbf{a}(t)\frac{\partial f(\mathbf{x}(t), t, \boldsymbol{\theta})}{\partial \mathbf{x}(t)}dt \tag{3.19}$$

Similarly, the gradients with respect to θ could be calculate by integrate over the interval and use $a_\theta(t_N) = 0$ as initial state

$$\frac{dL}{d\boldsymbol{\theta}} = -\int_{t_N}^{t_0}\mathbf{a}(t)\frac{\partial f(\mathbf{x}(t), t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}dt \tag{3.20}$$

The gradients with respect to t_0, t_N can be calculate as

$$\frac{dL}{dt_0} = \mathbf{a}_t(t_N) - \int_{t_N}^{t_0} \mathbf{a}(t) \frac{\partial f(\mathbf{x}(t), t, \boldsymbol{\theta})}{\partial t} dt \tag{3.21}$$

We have the Adjoint sensitivity algorithm

Algorithm 1: Reverse-mode derivative of an ODE IVP (Chen et al., 2018)

Input: dynamics parameters $\boldsymbol{\theta}$, start time t_0, stop time t_1, final state $\mathbf{x}(t_1)$, loss gradient $\frac{\partial L}{\partial \mathbf{x}(t_1)}$

`// Compute gradients with regrads to` t_1

$\frac{\partial L}{\partial t_1} = \frac{\partial L}{\partial \mathbf{x}(t_1)} f(\mathbf{x}(t_1), t_1, \boldsymbol{\theta})$

`// Define initial augmented state`

$\mathbf{s}_0 = [\mathbf{x}(t_1), \frac{\partial L}{\partial \mathbf{x}(t_1)}, 0, -\frac{\partial L}{\partial t_1}]$

`// Define dynamics on augmented state`

def *aug_dynamics*$([\boldsymbol{x}(t), \boldsymbol{a}(t), \cdot, \cdot], t, \boldsymbol{\theta})$*:*

`// Compute vector-Jacobian products`

return $[f(\boldsymbol{x}(t), t, \boldsymbol{\theta}), -\boldsymbol{a}(t)\frac{\partial f}{\partial \boldsymbol{x}}, -\boldsymbol{a}(t)\frac{\partial f}{\partial \boldsymbol{\theta}}, -\boldsymbol{a}(t)\frac{\partial f}{\partial t}]$

`// Solve reverse time ODE`

$\left[\mathbf{x}(t_0), \frac{\partial L}{\partial \mathbf{x}(t_0)}, \frac{\partial L}{\partial \boldsymbol{\theta}}, \frac{\partial L}{\partial t_0}\right] = \text{ODESolver}(\mathbf{s}_0, \textit{aug\_dynamics}, t_1, t_0, \boldsymbol{\theta})$

return $\frac{\partial L}{\partial \boldsymbol{x}(t_0)}, \frac{\partial L}{\partial \boldsymbol{\theta}}, \frac{\partial L}{\partial t_0}, \frac{\partial L}{\partial t_1}$

3.1.3 Neural Ordinary Differential Equations Problem

Although NODEs shows a promising direction as continuous dynamics, NODEs has a problem as the following theorem highlights.

Theorem 3.1. *NODEs is not universal approximation. There are classes of function that NODEs cannot represent.*

We will proof Theorem 3.1 by pointing out there are at least one class of function that NODEs cannot represent. Firstly, we will show an important property of ODE

Proposition 3.2. *(Proposition C.6 Younes (2010)) Let $z_1(t)$ and $z_2(t)$ be two solusions of the ODE 2.2 with different initial conditions, i.e. $z_1(0) \neq z_2(0)$. Then for all $t \in (0, T]$, $z_1(t) \neq z_2(t)$ or ODE trajectories cannot intersect.*

Proof. Suppose there exists some $t \in (0, T]$ where $z_1(t) = z_2(t)$. Define a new IVP problem, with initial value as $z_1(t) = z_2(t)$. Due to Existence and Uniqueness Theorem 2.1, we have $z_1(0) = z_2(0)$ which is a contradiction. So $\nexists t \in (0, T] : z_1(t) = z_2(t)$. □

Now, we will show that there's at least one class of functions that NODEs cannot represent

Proposition 3.3. *(Dupont et al., 2019) Let $0 < r_1 < r_2 < r_3$ and $g : \mathbb{R}^d \to \mathbb{R}$ be a function such that*

$$\begin{cases} g(\boldsymbol{x}) = -1 \text{ if } \|\boldsymbol{x}\|_2 \leq r_1 \\ g(\boldsymbol{x}) = 1 \text{ if } r_2 \leq \|\boldsymbol{x}\|_2 \leq r_3 \end{cases}$$

Neural ODEs cannot represent $g(\boldsymbol{x})$.

Proof. Reader could refer to Appendix A.1 for more details on the proof. □

By proving that NODEs cannot represent $g(x)$ as define in proposition 3.3 we have proved Theorem 3.1. We will interpret proposition 3.3 by showing the case in $1d$ or

$$\begin{cases} g(1) = -1 \\ g(-1) = 1 \end{cases} \tag{3.22}$$

The figure 3.1 shows the case function $g(x)$ defined in proposition 3.3 with $d = 1$, we can see that continuous trajectories mapping -1 to 1 (red line) and from 1 to -1 must intersect each other (top left figure). The dash line in top right corner show solution of ResNet (discrete case).

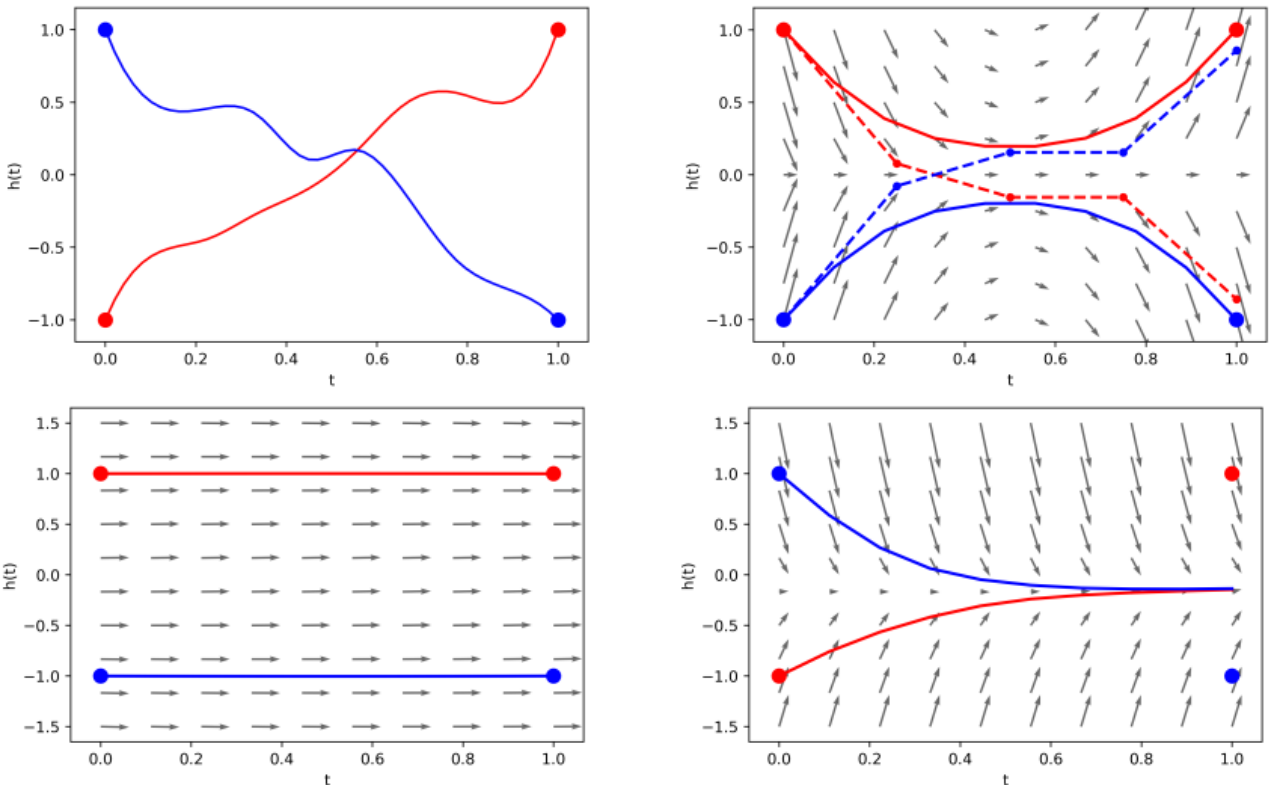

Figure 3.1: $1d$ proposition 3.3. Image from Dupont et al. (2019)

3.1.4 NODEs variants

This section will introduce 2 variants of NODEs, which are Neural Stochastic Differential Equations (Neural SDE) and Neural Controlled Differential Equations (Neural CDEs). There's another variant of NODEs which is Graph Neural Ordinary Differential Equations (Neural GDE), but it is out of scope for this report. Readers could refer to Poli et al. (2019) for more details about Neural GDE. The gray array in the background is the the continuous vector field.

3.1.4.1 Neural Stochastic Differential Equations

Neural Stochastic Differential Equations or Neural SDE (Liu et al., 2019) idea starts with authors trying to find a way to regularise NODEs but regularised techniques like Drop Out (Srivastava et al., 2014), randomly drop residual blocks of ResNet during training time (Huang et al., 2016), adding noise in forward pass (An, 1996; Bishop, 1995) or the Shake-Shake regularisation of ResNet (Gastaldi, 2017) only work for discrete neural networks while NODEs represent a continuous system. Neural

SDE tries to increase the robustness of NODEs by adding a diffusion term (function g) into NODEs and it can be expressed as follow

$$d\mathbf{x}(t) = f(\mathbf{x}(t), t, \boldsymbol{\theta})dt + g(\mathbf{x}(t), t, \boldsymbol{\phi})dB_t \tag{3.23}$$

where B_t is the standard Brownian motion (Øksendal, 2010), which is a continuous time stochastic process such that $B_{t_s} - B_s$ $(0 < s < t_s)$ follows Gaussian with mean 0 and variance t, $g(\mathbf{x}(t), t, \boldsymbol{\phi})$ is an arbitrary neural networks with parameter ϕ.

3.1.4.2 Neural Controlled Differential Equations

Neural Controlled Differential Equations or Neural CDEs (Kidger et al., 2020) is an attempt to create a continous dynamics model of Recurrent Neural Network. Let $X : [t_0, t_n] \to \mathbb{R}^{d+1}$ be the natural cubic spline with knots at $t_0, \dots, t_n$ such that $X_{t_i} = (\mathbf{x}_i, t_i)$, $\mathbf{x}_i \in \mathbb{R}^d$ is the observation at timestamp $t_i \in \mathbb{R}$ and $t_0 < \cdots < t_n$.

Let $\mathbf{z}_{t_0} = g(\mathbf{x}_0, t_0, \boldsymbol{\phi})$ be the initial state where $g(\mathbf{x}, t, \boldsymbol{\phi})$ be any arbitrary neural network with parameter $\boldsymbol{\phi}$. We have the Neural CDE as follow:

$$\mathbf{z}_t = \mathbf{z}_{t_0} + \int_{t_0}^{t} f(\mathbf{z}_t, t, \boldsymbol{\theta})dX_t, \text{ for } t \in (t_0, t_n] \tag{3.24}$$

where $f(z, t, \theta)$ is any neural network with parameter θ.

To evaluate Neural CDE, we can do it by defining $g(z_t, t, \theta) = f(z_t, t, \theta)\frac{dX}{dt}(t)$, then we have the original ODE problem that for all $t \in (t_0, t_n]$

$$z_t = z_{t_0} + \int_{t_0}^{t} f(z_t, t, \theta)dX_t = z_{t_0} + \int_{t_0}^{t} f(z_t, t, \theta)\frac{dX}{dt}dt = z_{t_0} + \int_{t_0}^{t_n} g(z_t, t, \phi)dt \tag{3.25}$$

and we can solve this problem using any methods that we could use for NODEs.

The following two theorems are two important properties of Neural CDEs.

Theorem 3.2. *(Kidger et al., 2020)(Universal Approximation)(Informal): The action of a linear map on terminal value of a Neural CDEs is a universal approximator from {sequences in $\mathbb{R}^d$} to $\mathbb{R}$.*

Proof. Please refer to Theorem B.14 (Kidger et al., 2020) for a formal proof. The main idea is that CDEs may be used to approximate bases of functions on path space. □

Theorem 3.3. *(Kidger et al., 2020)(Informal) Any equation of the form $z_t = z_0 + \int_{t_0}^{t} h(z_0, X(t), \theta)dt$ may be represented exactly by a Neural CDE of the form $z_t = z_0 + \int_{t_0}^{t} f(z_t, \theta)dX_t$. However the converse statement is not true.*

Proof. Please refer to Theorem C.1 (Kidger et al., 2020) for a formal proof. □

3.2 Stage 1: ResNet block as NODEs block

Although we have known that NODEs is the continuous version of ResNet, most of NODEs research still focus on time series dataset rather than on complex dataset like images or natural language. That raises a concern to us. **Why don't we use NODEs on complex dataset like images or natural language?** In this first stage of this research project, we try to find the answer to that question by replacing ResNet block with NODEs block in a GANs model.

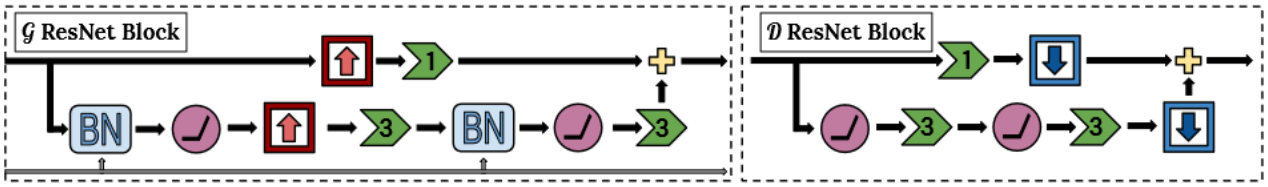

Figure 3.2: ResNet block structures in DVD-GAN, image from Clark et al. (2019)

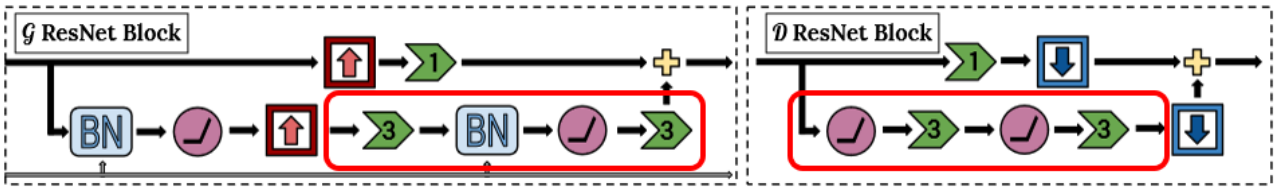

Figure 3.3: The red circle show the part of ResNet Block in DVD-GAN to convert into the NODEs block

We choose DVD-GAN (Clark et al., 2019) as DVD-GAN has ResNet block in both their Generator and Discriminator. We will first try to replace these 2 ResNet block with NODEs blocks. Figure 3.2 shows the detail structure of their ResNet blocks.

The function of ResNet block for generator is as follows

$$\begin{aligned} x_{out} = {} & Conv2d(UpSample(\mathbf{x}_{in}))+ \\ & Conv2d(ReLU(BatchNorm(Conv2d(UpSample(ReLU(BatchNorm(\mathbf{x}_{in}))))))) \end{aligned} \tag{3.26}$$

We convert this ResNet block into NODEs by defining ODE function as follows

$$f_{ode\_gen}(\mathbf{x}) = Conv2d(ReLU(BatchNorm(Conv2d(\mathbf{x})))) \tag{3.27}$$

Hence the generator's ResNet block becomes

$$\begin{aligned} \mathbf{x}_0 &= UpSample(ReLU(BatchNorm(\mathbf{x}_{in}))) \\ \mathbf{x}_{out} &= \text{ODESolver}(f_{ode\_gen}, \mathbf{x}_0, 0, 1) \end{aligned} \tag{3.28}$$

where x_{in} is the input of the blocks and x_{out} is the output of the block.

Similarly, we have the ResNet block of the discriminator

$$\mathbf{x}_{out} = DownSample(Conv2d(x_{in})) + DownSample(Conv2d(ReLU(Conv2d(ReLU(\mathbf{x}_{in}))))) \tag{3.29}$$

We convert this ResNet block for the discriminator

$$f_{ode\_dis} = Conv2d(ReLU(Conv2D(ReLU(\mathbf{x}_{in})))) \tag{3.30}$$

And the discriminator's ResNet block becomes

$$\begin{aligned} \mathbf{x}_0 &= \mathbf{x}_{in} \\ \mathbf{x}_{out} &= DownSample(\text{ODESolver}(\mathbf{x}_{in}, f_{ode\_dis}, 0, 1)) \end{aligned} \tag{3.31}$$

Figure 3.3 show which part of original ResNet block in DVD-GAN to convert into ODEs dynamics.

Convolution Layer and Neural ODEs for normal ODE problem 3.3, x_0 and $x(t)$ has to be in the same space (e.g x_0 cannot be in $\mathbb{R}^3$ while $x(t) \in \mathbb{R}^4$). However, the convolution layer often change the number of channels or convolution layer usually change the input space which make it's not possible to use NODEs. To solve this problem, Dupont et al. (2019) proposed that when the the number of input channels is less than the number of output channels, we will pad input with zeros channels make input and output to be in the same space.

We will give a simple example: let $x_0 \in \mathbb{R}$ and $f(\mathbf{x}(t), t, \theta) : \mathbb{R}^3 \to \mathbb{R}^3$. We could not use x_0 as initial state for 3.3. We change it into $\mathbf{x}_{aug} = \begin{bmatrix} x_0 \\ 0 \\ 0 \end{bmatrix}$ then we have a NODEs

$$\begin{aligned} \frac{d\mathbf{x}(t)}{dt} &= f(\mathbf{x}(t), t, \theta) \\ \mathbf{x}(0) = \mathbf{x}_{aug} &= \begin{bmatrix} x_0 \\ 0 \\ 0 \end{bmatrix} \end{aligned} \tag{3.32}$$

Note: Padding the output in case the the number of output channels is less than the number of input channels will not give us same output in general.

3.3 Stage 2: GANs Training by solving ODEs

In the second stage of this research project, we reinvestigate the same problem that Qin et al. (2020) have done: Training GANs with continuous dynamics. As we review Gradient Descent (GD) in Chapter 2, given a function $f(x) : \mathbb{R}^n \to \mathbb{R}$, we try to find the minimum value of function f by going in descent direction that determined by its gradient.

So for step $k+1$, we have the updated algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k) \tag{3.33}$$

where $\nabla f(\mathbf{x}_k)$ is the gradient of f with input $\mathbf{x}_k$, α is the learning rate. We could rewrite equation 3.33 as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}) \times 1 = \mathbf{x}_k - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}) \Delta t \tag{3.34}$$

where $\Delta t = 1$ is step size.

Let $\Delta t \to 0$, 3.34 become an IVP

$$\begin{aligned} \frac{d\mathbf{x}(t)}{dt} &= f(\mathbf{x}(t)) \\ \mathbf{x}(0) &= \mathbf{x}_0 \end{aligned} \tag{3.35}$$

So we could find minimum of f by solving the IVP 3.35 using any ODE Solver.

Starting from this idea, (Qin et al., 2020) proposed training GAN under continuous dynamics. Consider l_D and l_G are loss functions of discriminator and generator respectively. Let $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ be parameters of discriminator and generator respectively. Using GD, the parameter θ, ϕ at iteration $k+1$ is given by the following

$$\begin{aligned} \boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \alpha \frac{\partial l_D}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k) \Delta t \\ \boldsymbol{\phi}_{k+1} &= \boldsymbol{\phi}_k - \beta \frac{\partial l_G}{\partial \boldsymbol{\phi}}(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k) \Delta t \end{aligned} \tag{3.36}$$

where $\alpha\Delta t$ and $\beta\Delta t$ are learning rates. As $\Delta t \to 0$, we will have the IVP problem

$$\begin{bmatrix} \frac{d\boldsymbol{\theta}}{dt} \\ \frac{d\boldsymbol{\phi}}{dt} \end{bmatrix} = - \begin{bmatrix} \alpha \frac{\partial l_D}{\partial \boldsymbol{\theta}} \\ \beta \frac{\partial l_G}{\boldsymbol{\phi}} \end{bmatrix} \\ \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\phi} \end{bmatrix} (0) = \begin{bmatrix} \boldsymbol{\theta}_0 \\ \boldsymbol{\phi}_0 \end{bmatrix} \tag{3.37}$$

3.3.1 Analysis of Linearised Dynamics for GANs in Vicinity of Nash Equilibria

This section follows closely section 3.2.1 (Qin et al., 2020). Let $v(\boldsymbol{\theta}, \boldsymbol{\phi}) = - \begin{bmatrix} \alpha \frac{\partial l_D}{\partial \boldsymbol{\theta}} \\ \beta \frac{\partial l_G}{\boldsymbol{\phi}} \end{bmatrix}$. Since Nash equilibrium is the the point where both generator and discriminator cannot improve further, we have the gradient of loss functions at Nash equilibrium equal 0. Consider the local Nash equilibrium $(\boldsymbol{\theta}^*, \boldsymbol{\phi}^*)$ where $v(\boldsymbol{\theta}^*, \boldsymbol{\phi}^*) = \mathbf{0}$. Let δ be the different $\boldsymbol{\delta} = [\boldsymbol{\theta}, \boldsymbol{\phi}] - [\boldsymbol{\theta}^*, \boldsymbol{\phi}^*]$. We have $\boldsymbol{\delta}' = -H\boldsymbol{\delta} + \mathcal{O}(|\boldsymbol{\delta}|^2)$ where H is the Jacobian with the following form in GANs:

$$H = - \begin{bmatrix} \frac{\partial v}{\partial \boldsymbol{\theta}}, \frac{\partial v}{\partial \boldsymbol{\phi}} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 l_D}{\partial \boldsymbol{\theta}^2} & \frac{\partial^2 l_D}{\partial \boldsymbol{\theta} \partial \boldsymbol{\phi}} \\ \frac{\partial^2 l_G}{\partial \boldsymbol{\theta} \partial \boldsymbol{\phi}} & \frac{\partial^2 l_G}{\partial \boldsymbol{\phi}^2} \end{bmatrix} \tag{3.38}$$

For cross-entropy loss, the modified loss and Wasserstein loss loss, we have $\frac{\partial^2 l_D}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} = - \frac{\partial^2 l_G}{\partial \boldsymbol{\theta} \partial \boldsymbol{\phi}}^T$. (We will show this in Appendix A.2). As the result we have the linearised dynamics of continuous dynamics

$$\begin{bmatrix} \frac{d\boldsymbol{\theta}}{dt} \\ \frac{d\boldsymbol{\phi}}{dt} \end{bmatrix} = - \begin{bmatrix} A & B^T \\ -B & C \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\phi} \end{bmatrix} \tag{3.39}$$

where A, B, C denote the element of H defined in 3.38.

Theorem 3.4. *(Lemma 3.1 Qin et al. (2020)) Given a linearesd vector field of the form 3.39 were $A \succ 0$ and $C \succeq 0$ or $A \succeq 0$ and $C \succ 0$, and B is full rank. The dynamics always converges to $v = [\mathbf{0}, \mathbf{0}]^T$.*

Proof. Please refer to Lemma 3.1 Qin et al. (2020) for the proof. □

What Theorem 3.4 mean is that the GANs dynamics with cross-entropy loss, modified GANs loss and Wasserstein loss will always converges to a point with gradient equal 0.

3.3.2 Update rule using Numerical ODE solver

With h is the step size or the learning rate, we have the update rule using Euler's method, Heun's method (RK2) and Runge Kutta 4 (RK4)

Euler's Method

$$\begin{bmatrix} \boldsymbol{\theta}_{k+1} \\ \boldsymbol{\phi}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{\phi}_k \end{bmatrix} + hv(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k) \tag{3.40}$$

Heun's Method (RK2)

$$\begin{bmatrix} \tilde{\boldsymbol{\theta}}_k \\ \tilde{\boldsymbol{\phi}}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_{k+1} \\ \boldsymbol{\phi}_{k+1} \end{bmatrix} + hv(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k) \\ \begin{bmatrix} \boldsymbol{\theta}_{k+1} \\ \boldsymbol{\phi}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{\phi}_k \end{bmatrix} + \frac{h}{2} \left(v(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k) + v(\tilde{\boldsymbol{\theta}}_k, \tilde{\boldsymbol{\phi}}_k) \right) \tag{3.41}$$

Runge Kutta 4 Method (RK4)

$$
\begin{aligned}
v_1 &= v(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k) \\
v_2 &= v\left(\boldsymbol{\theta}_k + \frac{h}{2}(v_1)_{\boldsymbol{\theta}}, \boldsymbol{\phi}_k + \frac{h}{2}(v_1)_{\boldsymbol{\phi}}\right) \\
v_3 &= v\left(\boldsymbol{\theta}_k + \frac{h}{2}(v_2)_{\boldsymbol{\theta}}, \boldsymbol{\phi}_k + \frac{h}{2}(v_2)_{\boldsymbol{\phi}}\right) \\
v_4 &= v\left(\boldsymbol{\theta}_k + h(v_3)_{\boldsymbol{\theta}}, \boldsymbol{\phi}_k + h(v_3)_{\boldsymbol{\phi}}\right) \\
\begin{bmatrix}\boldsymbol{\theta}_{k+1} \\ \boldsymbol{\phi}_{k+1}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{\theta}_k \\ \boldsymbol{\phi}_k\end{bmatrix} + \frac{h}{6}(v_1 + 2v_2 + 2v_3 + v_4)
\end{aligned}
\tag{3.42}
$$

We implemented with some modification to save memory in training. Detail of our modified implementation in A.3.

3.4 Stage 3: GAN with NODEs to Generate Video

With the success of modeling the motion using continuous dynamics of ODE^{2}VAE (Yildiz et al., 2019), we try to leverage MoCoGAN (Tulyakov et al., 2018) by using the continuous dynamics NODEs to generate motion sequence instead of using RNN like original MoCoGAN model. This idea is also proposed by Gordon and Parde (2021). While their Inception Score shows somewhat promising on UCF101 (Soomro et al., 2012), they did not show any samples of their model. In this report, we will look more into the this idea using the dataset Rotated MNIST (https://github.com/ChaitanyaBaweja/RotNIST) with some modification to make the dataset more complex.

3.4.1 Generator Modifications

The input to Generator of original MoCoGAN is decomposed into 2 parts. One represents latent value in content space. It will decide the content in the video. The other represents the latent value in motion space. The content latent is generated as a noise like normal GANs. The motion latent sequence is generated by a Recurrent Neural Network (RNN) with input is a sequence of noises and initial hidden state is also a Gaussian noise. The detail structure of generator and discriminator is show in figure 3.4, figure 3.5 and figure 3.6

We will replace the RNN with a NODEs in 3 ways (the original RNN structure is show in figure 3.6):

- Replacing RNN with a normal NODEs.
- Replacing RNN with a Neural SDEs.
- Replacing RNN with a Neural CDEs.

Before going into detail structure of each modification, there's one interesting thing we notice from Gordon and Parde (2021) and Yildiz et al. (2019). That is they did not generate motion directly from noise (in case of MoCoGAN) or from the velocity (in case of ODE^{2}VAE) but feed the noise into another neural network before using it as initial state. We will test both cases in this report.

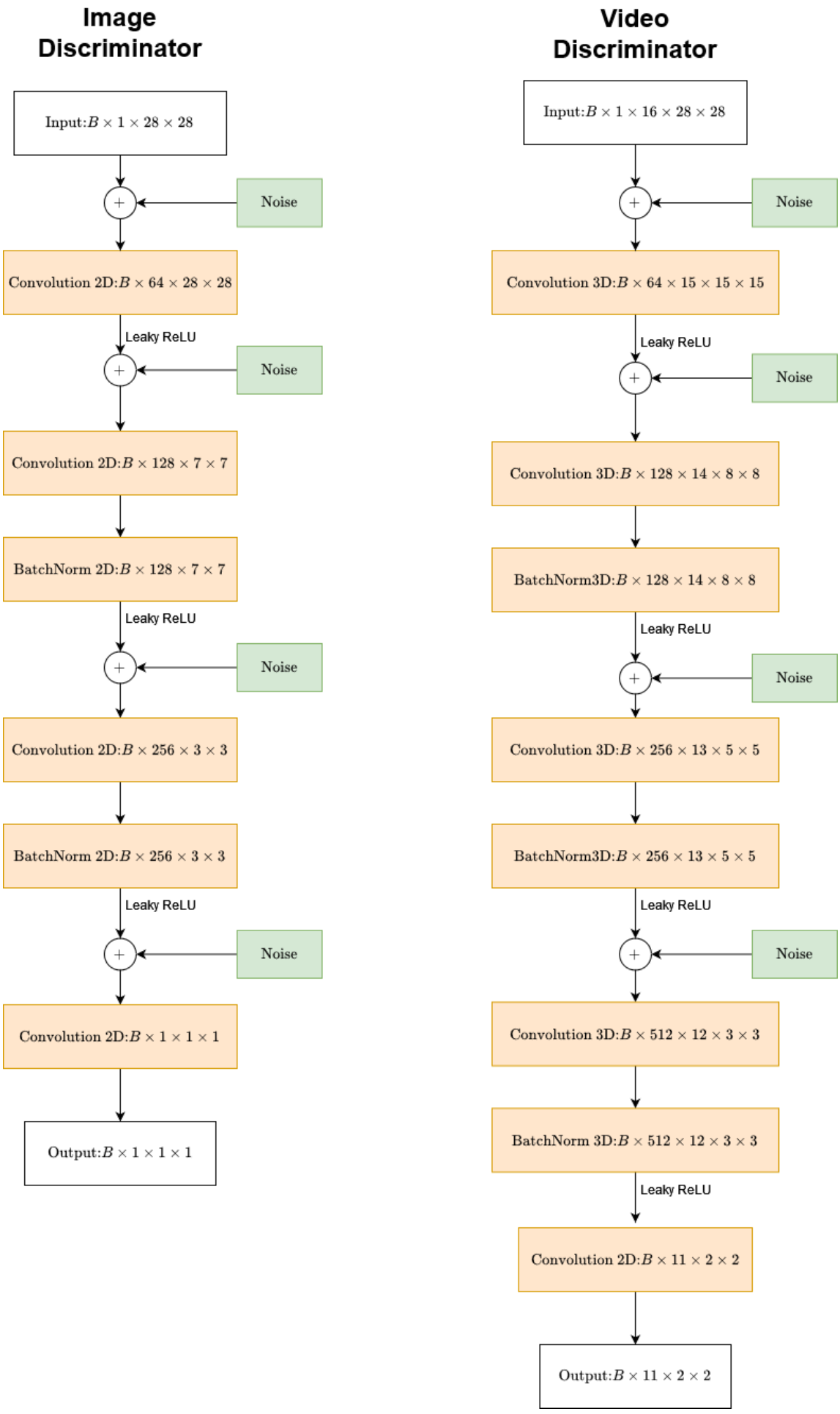

Figure 3.4: The structure of 2 discriminators of MoCoGAN

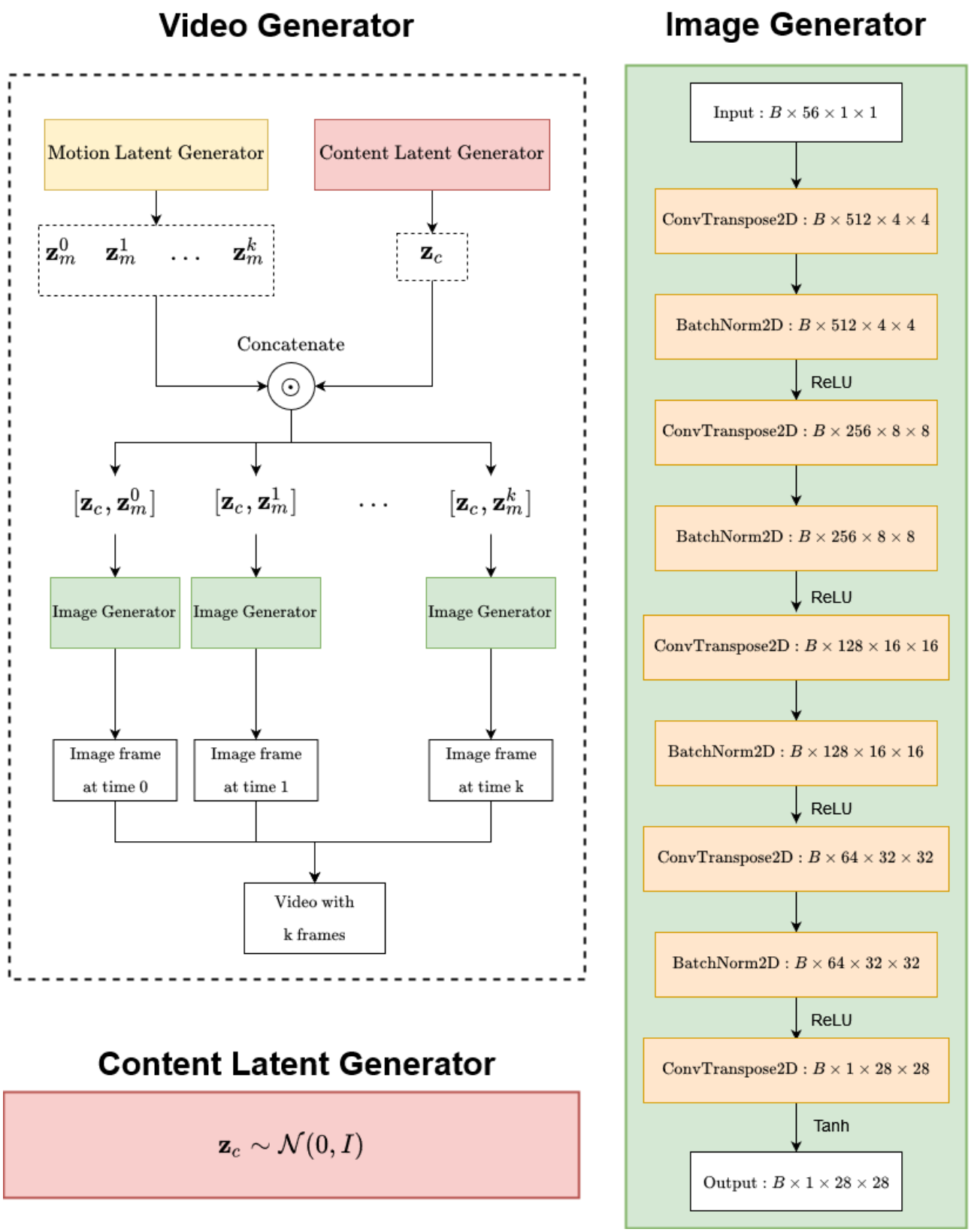

Figure 3.5: The structure of image generator of MoCoGAN

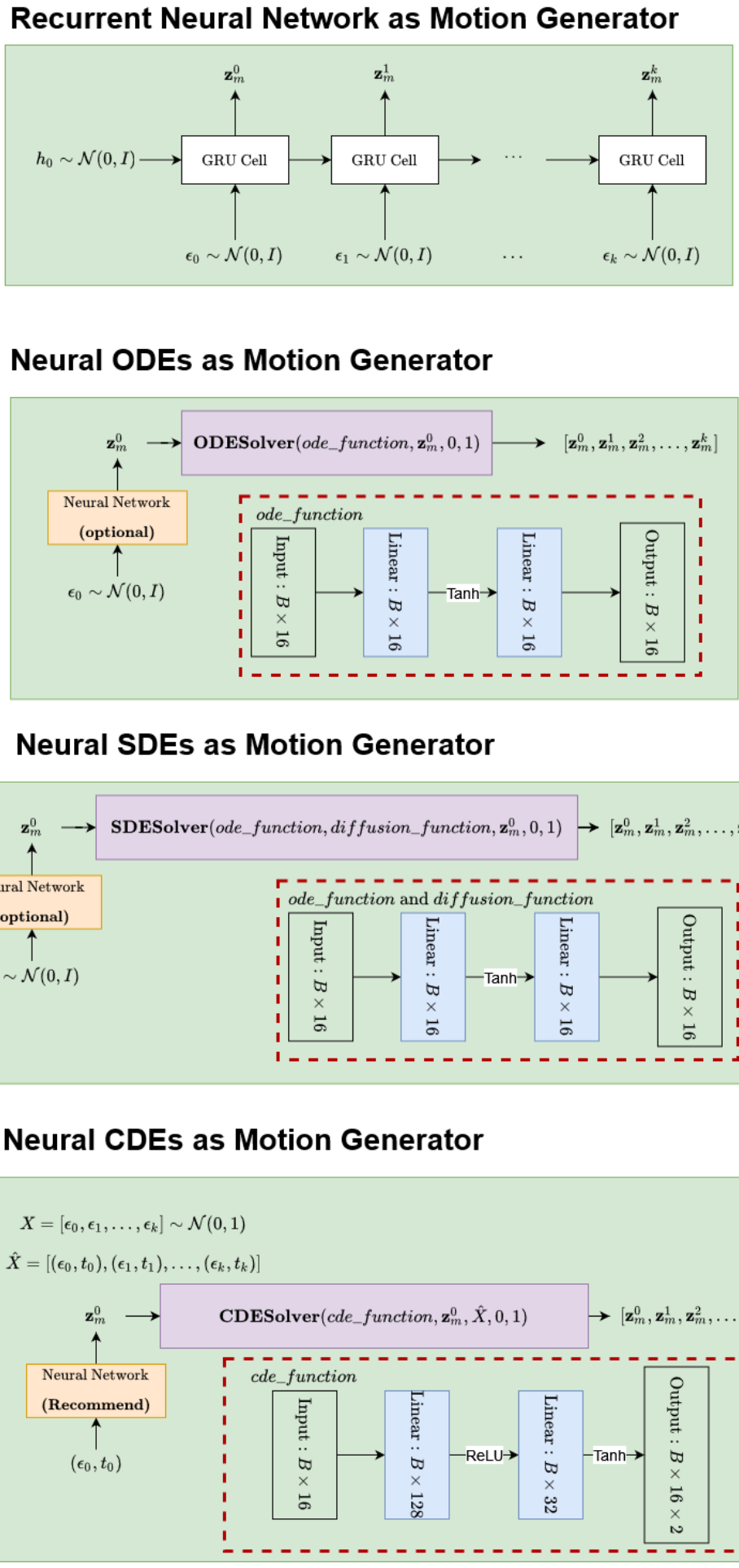

Figure 3.6: The structure of motion generator of MoCoGAN

Generating motion sequence using NODEs: We will use Gordon and Parde (2021) settings. ODE function is a 2-layer neural network with a Tanh activation function in hidden layer (the NODEs structure is show in figure 3.6).

$$f_{ode}(x) = Linear(Tanh(Linear(x))) \tag{3.43}$$

We generate the sequence of motion by the following steps

- Generate intial state as a Gaussian noise
- **(Optional)** Feed the intial state through a neural network to get new intial state
- Generate the sequence of motion by solving the ODE problem.

Generating motion sequence using Neural SDEs: We will use Gordon and Parde (2021) settings. The ODE function and diffusion function are the same (the Neural SDEs structure is show in figure 3.6.

$$\begin{aligned} f_{ode}(x) &= Linear(Tanh(Linear(x))) \\ g_{diff}(x) &= Linear(Tanh(Linear(x))) \end{aligned} \tag{3.44}$$

We generate the sequence of motion by the following steps

- Generate intial state as a Gaussian noise
- **(Optional)** Feed the intial state through a neural network to get new intial state
- Generate the sequence of motion by solving the ODE problem.

Generating motion sequence using Neural CDEs: The CDE function is defined as follows(the Neural CDEs structure is show in figure 3.6:

$$f_{cde}(x) = Tanh(Linear(ReLU(Linear(x)))) \tag{3.45}$$

Since Neural CDEs is considered as RNN with continuous dynamics, we will generate motion latent sequence almost the same way we do using RNN:

- Generate a sequence of Gaussian noise.
- Get initial state by feeding the first noise through another neural network (this step is recommended by Kidger et al. (2020))
- Generate the sequence of motion by solving the ODE problem.

Results and Analysis

In this chapter, we will investigate experiment results of 3 stages from the aforementioned chapter and discuss to answer the three following:

- Why isn't NODEs used more in a more complex model? (equivalent to stage 1 of this project)
- Does training GANs by solving ODEs give us the same result as Qin et al. (2020)? (equivalent to stage 2 of this project)
- Can we combine MoCoGAN with NODEs to generate complex videos? (equivalent to stage 3 of this project)

Software setup: PyTorch (Paszke et al., 2019), an open source deep learning framework, is the main library we use in this project. We also use `torchdiffeq` (Chen et al., 2018)[1], `torchsde`[2] (Liu et al., 2019) and `torchcde`[3] (Kidger et al., 2020). The code we use for stage 1 is based on an unofficial implementation on GitHub[4] and the code we use for stage 3 is based on the work of Gordon and Parde (2021)[5]. We contributed implementation for training using continuous dynamics with 3 methods Euler, Heun and RK4 [6]. The code and script to reproduce our result can be found at https://github.com/chechaohp/gan-ode.

Hardward setup: The experiments for stage 1 were run on a Google Colab Pro High-RAM mode. The rest of our work was executed on Google Colab Pro GPU with 16GB VRAM.

4.1 Stage 1: ResNet block as NODEs block

We tried to run the NODEs block on Google Colab Pro Virtual Machine with High-RAM mode, which has 25.46 GB of RAM. The NODEs has **199600** parameters. Since the number of input channels is 3 which is less than 100 - the number of output channels, we have to pad input with 97 zeros channels (number of parameters increase significantly compared to original ResNet block).

We ran with the same settings that we used to test generator ResNet block. The generator ResNet block could run normally using less than 2 GB of RAM while 25.46 GB of RAM was not enough to run the NODEs (the session crashed whenever we tried to run the test because the computational memory could not be stored in 25.46 GB of RAM).

[1]The library could be found at https://github.com/rtqichen/torchdiffeq
[2]The library could be found at https://github.com/google-research/torchsde
[3]The library could be found at https://github.com/patrick-kidger/torchcde
[4]The code could be found at https://github.com/Harrypotterrrr/DVD-GAN
[5]The code could be found at https://github.com/Zasder3/Latent-Neural-Differential-Equations-for-Video-Generation
[6]The implementation could be found at https://github.com/chechaohp/gan-ode/blob/main/on_dev/ode_training.py

The reasons of this failures are:

- **The computational cost for integral operation is too high** because we want to approximate continuous dynamics by taking small steps. The computational cost increases when we use higher order solver.
- **Padding input with zero channels will significantly increase the number of parameters in our model**. For example, if the number of input channel is 3 and output channel is 100, we need to pad 97 zero channels. Although these zero channels will not affect final results of NODEs and the parameters corresponding to these zeros channels will be useless, the model still have to do computations on these zero channels and learn these parameters. As a result, it is a waste of memory.

Therefore, we can conclude that although NODEs seems to be an advanced version of ResNet, using NODEs on complex dataset (which often requires our model to be big) is not feasible at the moment due to the computational burden.

4.2 Stage 2: GAN Training by solving ODEs

We tried to train a simple GAN on MNIST using Adam, ODE training with Euler's Method, Heun's Method(RK2), and Runge Kutta 4 (RK4) with fixed step size. In this section, we will refer model's name as model result from training using the optimization method (e.g Adam model will do something means model trained by Adam will do something).

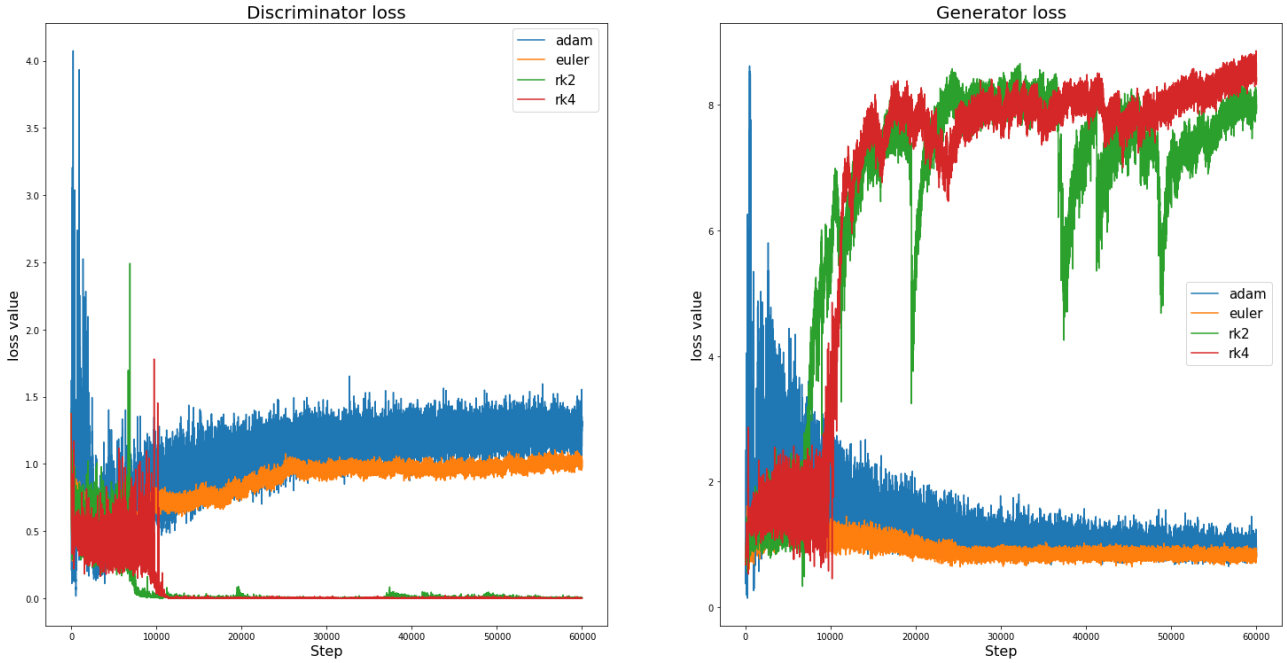

Figure 4.1: Loss value of discriminator (left image) and generator (right image) throughout the training process when using different optimization methods

4.2.1 Convergence:

Figure 4.1 shows the discriminator loss (left figure) and generator loss (right figure) during GAN training using different optimization methods. From the figure, we can clearly see that in this case

Heun's method (RK2 - green line) and RK4 (red line) suffer from vanishing gradient problem as the loss value of the discriminator is almost zero (green line and red line have loss value oscillate about 0) while Euler's method (orange line) and Adam (blue line) slowly converge to local Nash equilibrium.

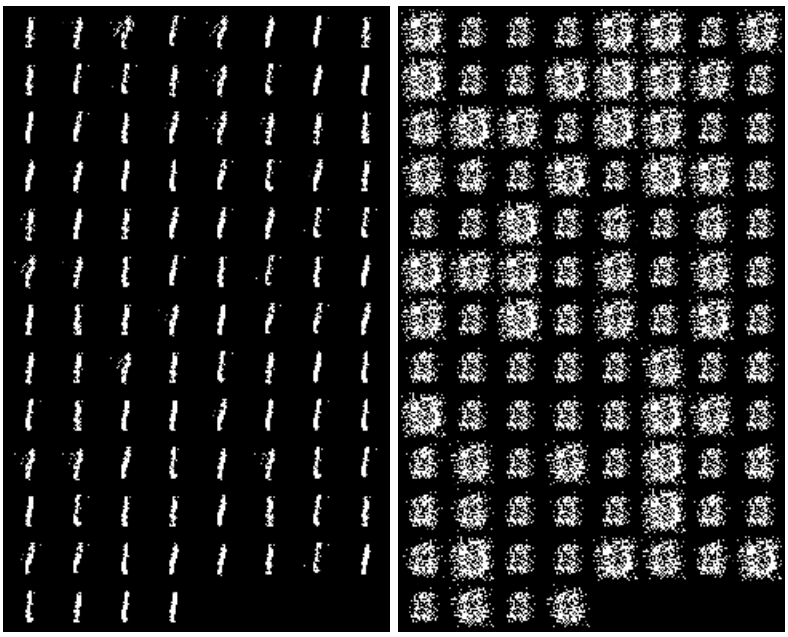

Figure 4.2: Samples from the generator of Heun's method (RK2 - left image) and Adam (right image) when the discriminator becomes too optimal

Whether having vanishing gradient or not, generator loss values of Euler, Heun and RK4 models are all converge to some specific values and oscillate round them (RK2 and RK4 explode at the beginning then slowly fluctuate around a value). We ran many experiments and in no experiment have we observed that the loss value does not converge. Hence, the model failed to learned or not, **when using continuous dynamics loss value will always converge** (or the gradient of loss function will go to zero as we keep training) as long as the preconditions of theorem 3.4 are satisfied.

4.2.2 Mode Collapse

In the case when RK2 and RK4 fail to learn, although the generator suffer from vanishing gradient, it can be seen that they also have mode collapse as portrayed in their output in figure 4.2. When comparing the output quality, the one produced by the Heun model looks sharper than that of Adam model. While Heun model learned to generate 1 only, it seems that Adam model learned to generate 0 only.

We noticed that although the loss value of Heun model converged, the model failed to learn the true distribution of dataset. We hypothesize that **mode collapse is a point that also satisfies at least one of the Nash equilibrium's properties**: the gradient of loss function at mode collapse point is 0. The same property can be seen in Adam case, despite exploding at the beginning (blue line o the left figure of figure 4.3), the generator loss value reach a plateau after a certain number of steps. This explains why mode collapse is a common problem in GAN game.

4.2.3 Weak Algorithm vs Strong Algorithm

In this experiment, we investigate the outcome between Euler's method and RK4.

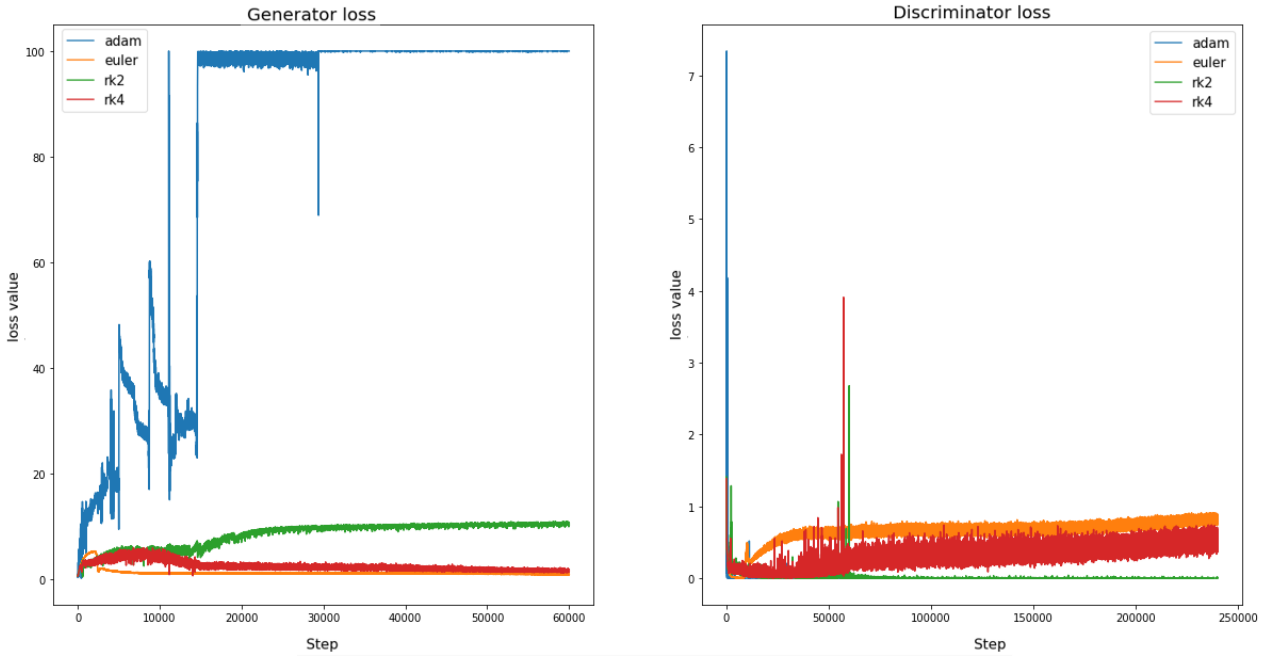

Figure 4.3: Loss value of generator (left image) and discriminator (right image) through out the training process when RK2 and Adam model fails to learn while Euler and RK4 could learn successfully.

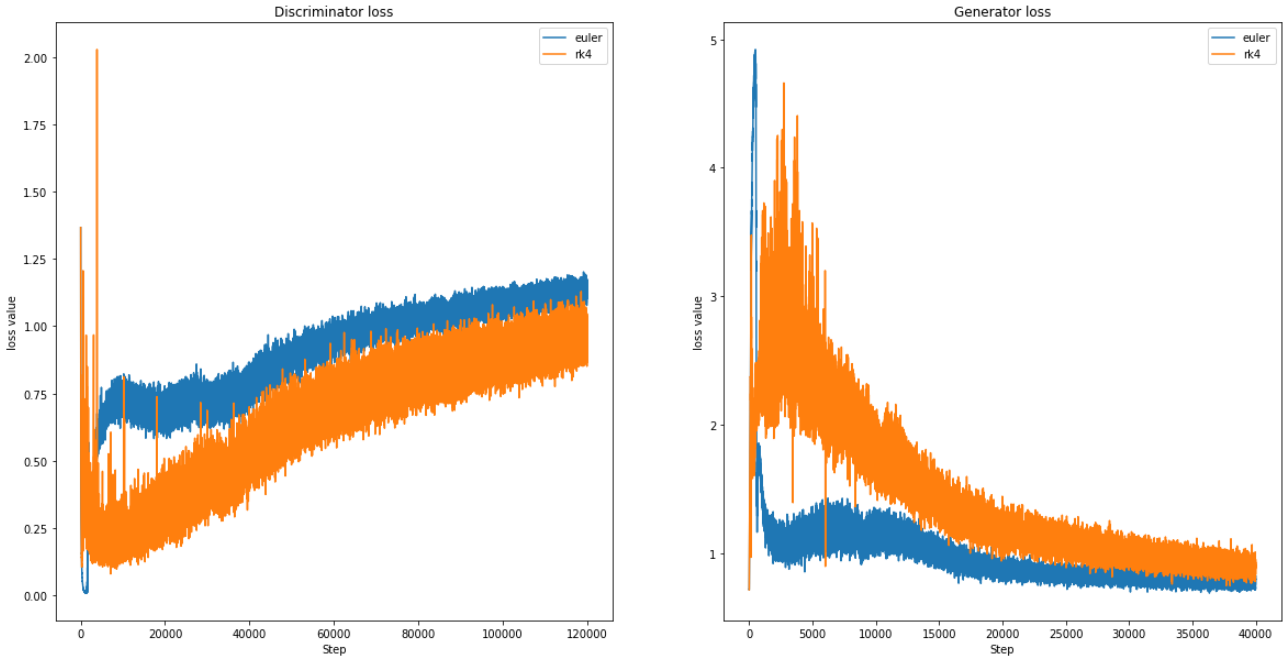

Figure 4.4: Loss value of discriminator (left image) and generator (right image) throughout the training process when using Euler's method and RK4

Figure 4.4 shows the loss value during the training time using Euler's method and RK4. The variance of loss value when we use RK4 is higher than the one using Euler's method. This could also be seen from figure 4.1. This shows that **high order algorithm is more deterministic with samples**, therefore when optimizing, it tends to lead to model to the direction of current samples. This might be a problem in stochastic learning which is common in machine learning.

Moreover, RK4 seems to take longer time to converge compared to Euler's method because of the determination of high order algorithm. We can also see this case in figure 4.3.

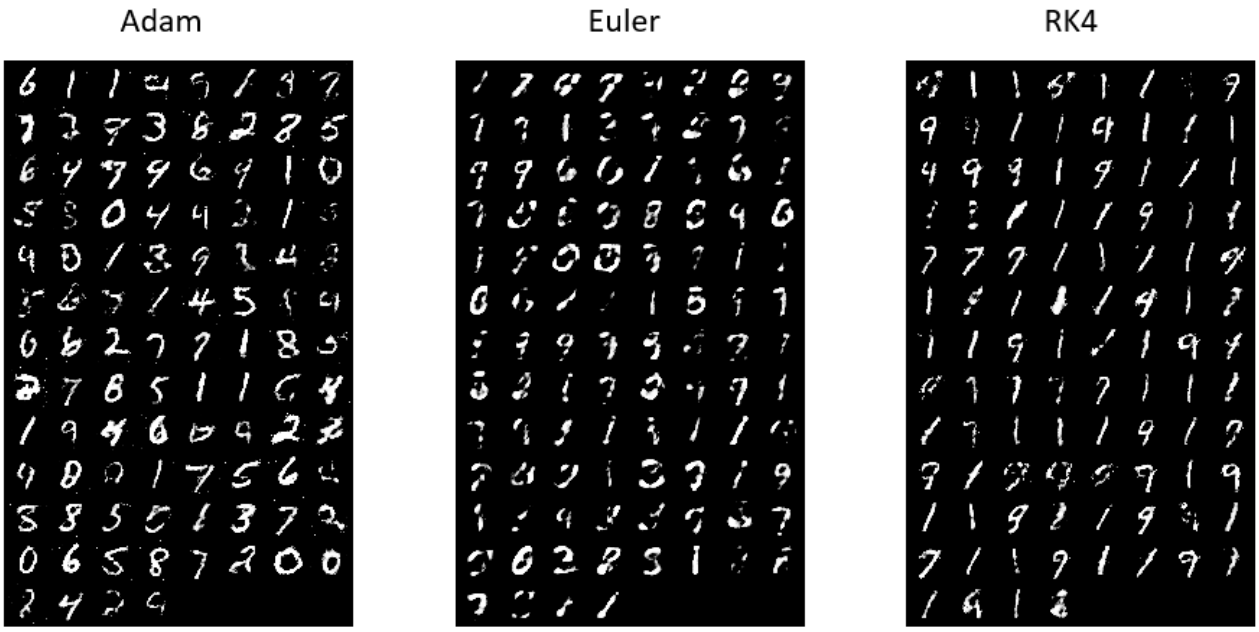

Figure 4.5: Samples generated using Adam model, Euler model and RK4 model when these models converges to Nash equilibrium

Figure 4.5 shows the samples from Adam model, Euler model and RK4 model when they converge to Nash equilibrium. Although the image quality of RK4 model is sharper, RK4 model's samples look less diverse than the other two. This could be due to its slower convergence. This result is the opposite of what Qin et al. (2020) claimed, that high order algorithm leads to faster convergence.

4.3 Stage 3: GAN with NODEs to Generate Video

In this section, we will investigate the performance of MoCoGAN with different NODEs variants as motion sequence generator using modified Rotated MNIST dataset. The original Rotated MNIST dataset is the MNIST dataset and rotate it 360 degree counter clockwise in sequences of 16 frames. We modified this dataset by randomly changing the angle which the images will rotate to increase the diversity of motion in the dataset in hope that the dataset will be more complex and if the result is promissing, we could try to use it on more complicated video dataset like UCF101 (Soomro et al., 2012). We will refer MoCoGAN with original NODEs modification as MoCoGAN-ODE, MoCoGAN with Neural SDEs modification as MoCoGAN-SDE, MoCoGAN with Neural CDE modification as MoCoGAN-CDE.

4.3.1 Final Results

Figure 4.6 4.7, 4.8 shows samples generated from MoCoGAN-ODE, MoCoGAN-SDE and MoCoGAN-CDE when training with class of digit 3. As we tried to run with different seeds, we always get mode collapse (in both content and motion) as you can see from the samples image. We address this as a serious problem.

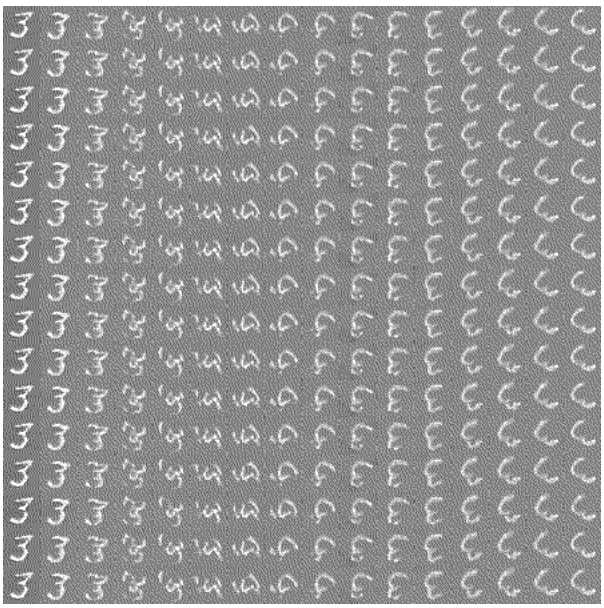

Figure 4.6: Samples from MoCoGAN-ODE

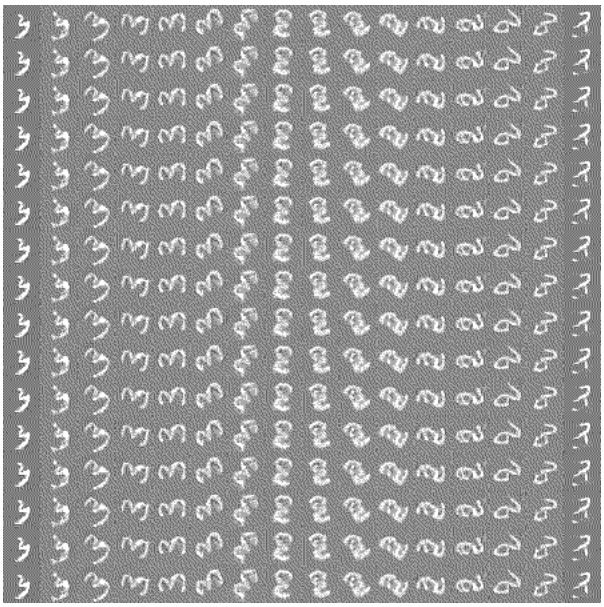

Figure 4.7: Samples from MoCoGAN-SDE

As we observed the training process to find the problem, we have found that:

- The generator learns the motion of the video really fast. But it took longer time for the content to be generated clearly.
- Although the dataset is diverse in rotating angle, the generator only learned one motion only (mode collapse). Moreover, not only the generator learn one motion only, but also one content.
- If we use WGAN loss to avoid mode collapse, the model cannot learn to generate anything, both in content and motion.

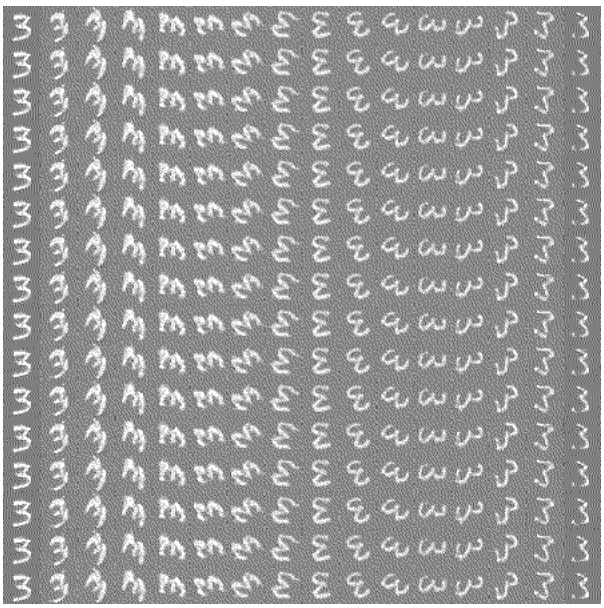

Figure 4.8: Samples from MoCoGAN-CDE

4.3.2 A closer look into NODEs as motion generator

As mentioning in section 3.1.3, we know that there are some class of functions that NODEs cannot represent because the ODE trajectories cannot intersect. Back to the case of the modified Rotated MNIST dataset, since we randomly change the rotating angle, we could have the image rotate clockwise and counter-clockwise. As the result, there are the samples from our dataset rotating from (for example) 90 degrees to -90 degrees and there are samples rotating from (for examples) -90 degrees to 90 degrees. As the result, these trajectories will have to intersect at some point. We believe that this problem confuse our NODEs and it decided to stick to one direction only which is counter-clockwise in this experiment (figure 4.9).

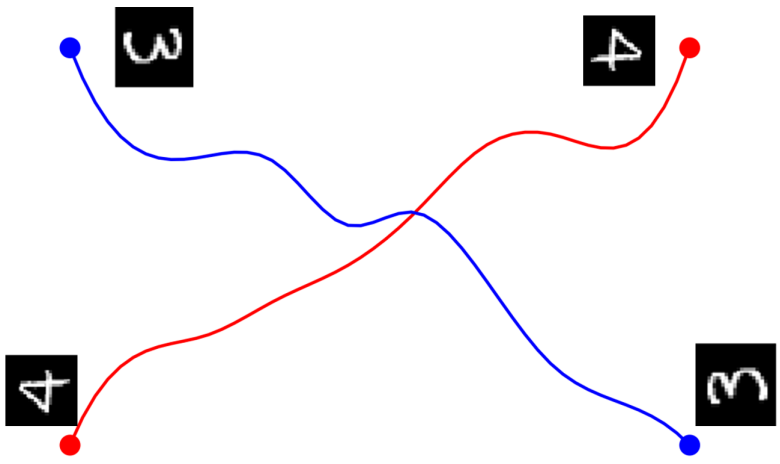

Figure 4.9: Interpretation of 2 trajectories will cross when the Rotated MNIST contains observations that rotates in both direction.

However, not only NODEs choose to learn one direction but also choose one rotating angle. We hypothesize that when NODEs decide to stick to one direction, that point is also a mode collapse point, and model cannot escape that point so it always generate same sequence. This happens to not only NODEs but also its variants Neural SDEs and Neural CDEs (Neural CDEs is universal in path

space, not the function space).

The generator also starts to generate same content after NODEs generating same motion sequence. We believe that the same motion sequence is the reason that make content also mode collapse. We believe that Qin et al. (2020) also suffer from this problem but did not report it in their work. We believe that this problem haven't been addressed before, and the motion generator and image generator should learn at similar pace or one generator will lead the other to mode collapse.

4.3.3 Ablation study

We run another experiments using ODE-RNN(Rubanova et al., 2019) as the motion generator. ODE-RNN adds a NODEs block between each hidden state to find new hidden state as input to next step time (detail how to implement ODE-RNN is on Section A.4). Samples of ODE-RNN is shown in figure 4.10

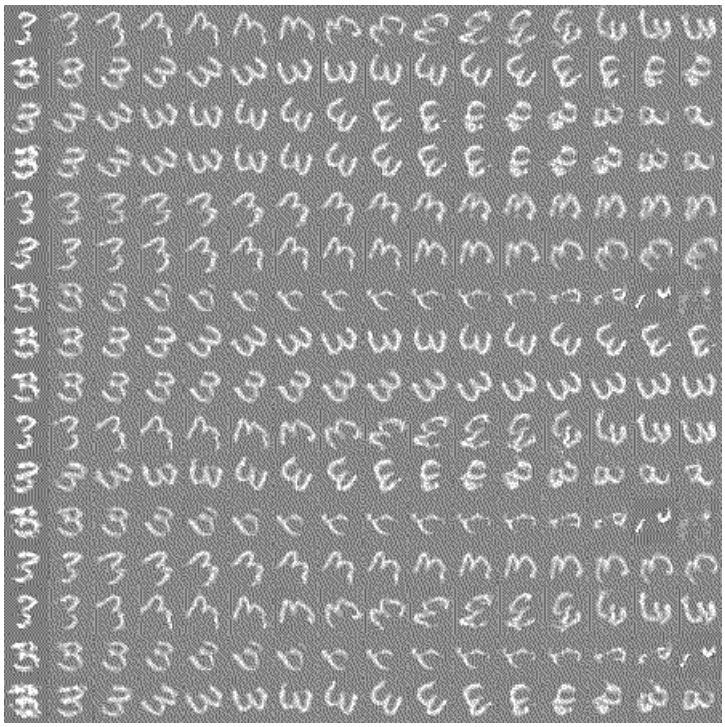

Figure 4.10: Samples from MoCoGAN with ODE-RNN as motion generator

As we can see, the motion in the samples are diverse and the content is also diverse the content is changing through the motion. Since ODE-RNN is not a continuous system like NODEs, the RNN part of ODE-RNN discretises the trajectories. As a result, MoCoGAN with ODE-RNN can learn the complex trajectories space. This strengthen our hypothesis that NODEs and its variants cannot learn the motion space because of continuous dynamics.

Conclusions and Future Development

This thesis is an attempt to apply neural ordinary differential equations to 3 problems: replacing residual block in a state-of-the-art neural network, using ode solver as an optimization algorithm in GANs game and generating motion sequence to generate videos with GANs. Furthermore, we provide a comprehensive and self-contained introduction to NODEs and GANs. In this chapter, we summarise our empirical results and an outlook on potential future work.

5.1 Empirical Results

We have shown that despite being a promising direction for deep learning, NODEs requires high computational cost which is a serious problem as it will require expensive hardware while many other state-of-the-art models can have a similar performance with less computational burden.

From the continuous GAN game point of view, we showed that mode collapse is also a point that satisfies Nash equilibrium's property, that is the gradient of loss function at that point is 0. We also showed that training by solving ODE might not be a good idea in stochastic learning which is a central role of modern machine learning because of the determinant of high order algorithm. However, training model by solving ODE always leads the model to converge.

We attempt to generate complex video using GAN with NODEs. However, our hypothesis failed because the continuous dynamics prevent the model to learn complex motion and, hence, lead to mode collapse. This problem is not only happens to NODEs but other NODEs variants like Neural SDEs and Neural CDEs. Neural CDE has a universal approximate theorem, however, it's only universal in path approximation, not in all kinds of function. On the other hand, we believe that the image generator and the motion generator should have the same pace of learn and one generator mode collapse will lead other mode collapses.

5.2 Future Work

In future work, we plan to work on Neural SDEs and Neural CDEs to find which class of functions that they cannot represent because they also collapse during our training. Furthermore, we plan to develop NODEs to universal function approximation. Moreover, one of the biggest obstacles for NODEs is the computation cost. We believe that this is also an open research question. In addition, since we learn that mode collapse is also a point that satisfies Nash equilibrium's property (so it might be a local Nash equilibrium), we plan to find a better way to represent GAN game to make GAN game avoid these bad local Nash equilibrium.

Apprendix A: Proofs and Additional Implementations

A.1 NODEs is not universal approximation

Theorem A.1. *(Gronwall's Lemma) Let $U \subset \mathcal{R}^d$ be an open set. Let $f : U \times [0,T] \to \mathbb{R}^d$ be a continous function and let $z_1, z_2 : [0,T] \to U$ satisfy the IVPs:*

$$\frac{dz_1(t)}{dt} = f(z_1(t), t), \quad z_1(0) = z_{10}$$
$$\frac{dz_2(t)}{dt} = f(z_2(t), t), \quad z_2(0) = z_{20}$$

Assune there is a constant $C \geq 0$ such that

$$\|f(z_1(t), t) - f(z_2(t), t)\| \leq C\|z_1(t) - z_2(t)\|$$

Then for all $t \in [0,T]$

$$\|z_1(t) - z_2(t)\| \leq e^{Ct}\|z_{10} - z_{20}\|$$

Proof. See Theorem 3.8 in Younes (2010) □

Theorem A.2. *Let $\phi(t)$ be feature function output by the ODE as the flow at the final time T that we achieve by solving the ODE $\phi(x) = \phi_T x$, we have $\phi : \mathbb{R}^d \to \mathbb{R}^d$. For all $t \in [0,T]$, $\phi_t : \mathbb{R}^d \to \mathbb{R}^d$ is a homeomorphism.*

Proof. (Dupont et al., 2019)(Theorem C.7 Younes (2010)) In order to prove that ϕ_t is homemorphism, we need to show that

- ϕ_t is continuous
- ϕ_t is bijection
- ϕ_t^{-1} is continuous

Consider two initial conditions of ODE system, $z_1(0) = x + \delta$ and $z_2(0) = x$ where δ is some perturbation. By Gronwall's Lemma, we have

$$\|z_1(t) - z_2(t)\| \leq e^{Ct}\|z_1(0) - z_2(t)\| = e^{Ct}\|\delta\| \tag{A.1}$$

We can rewrite this as

$$\|\phi_t(x + \delta) - \phi_t\| \leq e^{Ct}\|\delta\| \tag{A.2}$$

Let $\delta \to 0$, this implies that $\phi_t(x)$ is continuous in x for all $t \in [0, T]$.

Suppose there exists initial condition $x_{10} \neq z_{20}$ such that $\phi_t(z_{10}) = \phi_t(z_{20})$. Define new IVP problem with the initial values $\phi(z_{10})$ and solve it back to time 0. Since the solution is unique, we have $z_{10} = z_{20}$ (contradiction). So we have $\phi_t(z_{10}) \neq \phi_t(z_{20})$ whenever $z_{10} \neq z_{20}$. So ϕ_t is bijection.

To prove that inverse ϕ_t^{-1} is continuous, we can set initial condition to $z(t) = \phi_t(x)$ and solve the IVP backward in time and prove like proving $\phi_t(x)$ is continuous.

Therefore, ϕ_t is a homeomorphism. □

Proposition A.1. *(Proposition 3.3) (Dupont et al., 2019) Let $0 < r_1 < r_2 < r_3$ and $g : \mathbb{R}^d \to \mathbb{R}$ be a function such that*

$$\begin{cases} g(\boldsymbol{x}) = -1 \text{ if } \|\boldsymbol{x}\|_2 \leq r_1 \\ g(\boldsymbol{x}) = 1 \text{ if } r_2 \leq \|\boldsymbol{x}\|_2 \leq r_3 \end{cases}$$

Neural ODEs cannot represent $g(\boldsymbol{x})$.

Proof. (Dupont et al., 2019) Let $A = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_2 \leq r_1\}$ and $B = \{\mathbf{x} \in \mathbb{R}^d : r_2 \leq \|\mathbf{x}\|_2 \leq r_3\}$ and denote $\phi(S) = \{\mathbf{y} : \mathbf{y} = \phi(\mathbf{x}), \mathbf{x} \in S\}$ for any set S as a feature transformation. Let $\mathcal{L}$ be a linear map $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.

For a NODE to map points in A to -1 and points in B to 1, the linear map $\mathcal{L}$ must map features in $\phi(A)$ to -1 and the features in $\phi(B)$ to 1, which implies that $\phi(A)$ and $\phi(B)$ must be linearly separable. We will show that this is not possible if ϕ is a homeomorpism.

Define a disk $D \subset \mathbb{R}^d$ by $D = \{x \in \mathbb{R}^d : \|x\| \leq r_2\}$ with boundary $\partial D = \{x \in \mathbb{R}^d : \|x\| = r_2\}$ and the interior $int(D) = \{x \in \mathbb{R}^d : \|x\| < r_2\}$. We have $A \subset int(D)$, $A \cap \partial D = \emptyset$ and $\partial D \subset B$. That is all points in ∂D should be mapped to 1 and a subset of points in $int(D)$ should be mapped to -1 (i.e they are in A). So if $\phi(int(D))$ and $\phi(\partial D)$ are not linearly separable, then neither are $\phi(A)$ or $\phi(B)$.

The feature transformation ϕ is a homeomorphism, so $\phi(int(D)) = int(\phi(D))$ and $\phi(\partial D) = \partial(\phi(D))$, so the points on boundary get mapped to points on the boundary and points on the interior to points in the interior. So it remains to show that $int(\phi(D))$ and $\partial(\phi(D))$ cannot be linearly separated. Let $D' = \phi(D)$.

Suppose all points in $\partial D'$ lie above some hyperplane, i.e. suppose there exists a linear function $\mathcal{L}(x) = w^y x$ ad a constant C such that $\mathcal{L}(x) > C$ for all $x \in \partial D'$. If $int(D')$ were linearly separable from $\partial D'$ then $\mathcal{L}(x) < C$ for all $x \in int(D')$. We now show that this is not the case. Since D' is a connected subset of $\mathbb{R}^d$ (since D is connected and ϕ is a homeomorphism), every point $x \in int(D')$ can be written as a convex combination of points on the boundary $\partial D'$ (to tee this consider a line passing through a point x in the interior and its intersection with the boundary). So if $x \in int(D')$, then

$$x = \lambda x_1 + (1 - \lambda) x_2$$

for some $x_1, x_2 \in \partial D'$ and $0 < \lambda < 1$. Now,

$$\begin{aligned} \mathcal{L}(x) &= w^T x \\ &= w^T(\lambda x_1 + (1 - \lambda) x_2) \\ &= \lambda w^T x_1 + (1 - \lambda) w^T x_2 \\ &\geq \lambda C + (1 - \lambda) C \\ &= C \end{aligned}$$

So all points in the interior are on the same side of the hyperplane as points on the boundary, that is the interior and the boundary are not linearly separable. This imples that the set of features $\phi(A)$ and $\phi(B)$ cannot be linearly separated and so that NODEs cannot represent $g(x)$. □

A.2 Off-Diagonal Elements are Opposites at the Nash Equilibrium

This section is from Qin et al. (2020). We will prove that

Proposition A.2. *In GAN game, when the losses are cross-entropy, non-saturating loss and Wasserstein's loss, we have*

$$\frac{\partial^2 l_D}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} = -\frac{\partial^2 l_G}{\partial \boldsymbol{\theta} \partial \boldsymbol{\phi}}^T \tag{A.3}$$

Before proving the proposition, we need the following Lemma

Lemma A.1. *(Proposition 1 of Goodfellow et al. (2014)) For G fixed, the optimal discriminator D is*

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{A.4}$$

The proof of this Lemma is shown in Goodfellow et al. (2014). Now we will prove the above Proposition.

Proof. For cross-entropy and Wasserstein losses, the property holds because they represent the zero-sum game. We will prove this for non-saturating loss

$$l_D(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\int_x p_d(x) \log(D(x; \boldsymbol{\theta}))dx + \int_z p(z) \log(1 - D(G(z; \boldsymbol{\phi}); \boldsymbol{\theta}))dz \tag{A.5}$$

$$l_G(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\int_z p(z) \log(D(G(z; \boldsymbol{\phi}); \boldsymbol{\theta}))dz \tag{A.6}$$

Let $x = G(z; \boldsymbol{\phi})$ where x is now drawn from the probability distribution $p_g(x; \boldsymbol{\phi})$. We can rewrite this loss as follows

$$l_D(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\int_x p_d(x) \log(D(x; \boldsymbol{\theta})) + p_g(x; \boldsymbol{\phi}) \log(1 - D(x; \boldsymbol{\theta}))dx \tag{A.7}$$

$$l_G(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\int_x p_g(x; \boldsymbol{\phi}) \log(D(x; \boldsymbol{\theta}))dx \tag{A.8}$$

We have

$$\frac{\partial^2 l_D}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} = -\frac{\partial^2 l_G}{\partial \boldsymbol{\theta} \partial \boldsymbol{\phi}}^T \iff \frac{\partial^2 (l_D + l_G)}{\partial \boldsymbol{\theta} \partial \boldsymbol{\phi}} = 0 \tag{A.9}$$

For the non-saturating loss this becomes the following

$$\frac{\partial^2 (l_D + l_G)}{\partial \boldsymbol{\theta} \partial \boldsymbol{\phi}} = -\int_x \frac{\partial D(x; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial p_g(x; \boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \left(\frac{1}{D(x; \boldsymbol{\theta})} - \frac{1}{1 - D(x; \boldsymbol{\theta})} \right) \tag{A.10}$$

At the global Nash, we have $D(x) = p_{data}(x)/(p_{data}(x) + p_g(x; \boldsymbol{\phi}))$ and $p_g(x; \boldsymbol{\phi}) = p_d(x)$. Hence, this is identically zero. □

A.3 Memory Saving Implementation of ODE Solver

The original way of implementation, we have to save a clone of parameters along with gradients ($v(\boldsymbol{\theta}$,
$bm\phi)$). We modified it a little so we don't have to save a clone of parameter, we can keep updating parameters.

Modified Heun's Method (RK2)

$$\begin{aligned}\begin{bmatrix}\tilde{\boldsymbol{\theta}}_k\\ \tilde{\boldsymbol{\phi}}_k\end{bmatrix} &= \begin{bmatrix}\boldsymbol{\theta}_{k+1}\\ \boldsymbol{\phi}_{k+1}\end{bmatrix} + hv(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k)\\ \begin{bmatrix}\boldsymbol{\theta}_{k+1}\\ \boldsymbol{\phi}_{k+1}\end{bmatrix} &= \begin{bmatrix}\tilde{\boldsymbol{\theta}}_k\\ \tilde{\boldsymbol{\phi}}_k\end{bmatrix} + \frac{h}{2}\left(-v(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k) + v(\tilde{\boldsymbol{\theta}}_k, \tilde{\boldsymbol{\phi}}_k)\right)\end{aligned} \tag{A.11}$$

Modified Runge Kutta 4 Method (RK4)

$$\begin{aligned} v_1 &= v(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k)\\ \begin{bmatrix}\boldsymbol{\theta}_{k2}\\ \boldsymbol{\phi}_{k2}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{\theta}_k\\ \boldsymbol{\phi}_k\end{bmatrix} + \frac{h}{2}v_1\\ v_2 &= v(\boldsymbol{\theta}_{k2}, \boldsymbol{\phi}_{k2})\\ \begin{bmatrix}\boldsymbol{\theta}_{k3}\\ \boldsymbol{\phi}_{k3}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{\theta}_{k2}\\ \boldsymbol{\phi}_{k2}\end{bmatrix} + \frac{h}{2}(-v_1 + v_2)\\ v_3 &= v(\boldsymbol{\theta}_{k3}, \boldsymbol{\phi}_{k3})\\ \begin{bmatrix}\boldsymbol{\theta}_{k4}\\ \boldsymbol{\phi}_{k4}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{\theta}_{k3}\\ \boldsymbol{\phi}_{k3}\end{bmatrix} + h(-v_2/2 + v_3)\\ v_4 &= v(\boldsymbol{\theta}_{k4}, \boldsymbol{\phi}_{k4})\\ \begin{bmatrix}\boldsymbol{\theta}_{k+1}\\ \boldsymbol{\phi}_{k+1}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{\theta}_{k4}\\ \boldsymbol{\phi}_{k4}\end{bmatrix} + \frac{h}{6}(v_1 + 2v_2 - 4v_3 + v_4)\end{aligned} \tag{A.12}$$

A.4 ODE-RNN Algorithm

Algorithm 2: ODE-RNN, (Blue line is the only line that's different from original RNN)(Rubanova et al., 2019)

Input: Data points $\{x_1, x_2, \dots x_N\}$, ODE dynamics function f_θ

$h_0 = \mathbf{0}$

for i *in* $1, 2, 3, \dots, N$ **do**

    // Solve ODE to get state at time i

    $h'_i = \text{ODESolver}(f_\theta, h_{i-1}, (t_{i-1}, t_i))$

    // Update hidden state given current observation x_i

    $h_i = \text{RNNCell}(h'_i, x_i)$

end

$o_i = \text{OutputNN}(h_i)$ for all $i = 1, \dots, N$.

return $\{o_i\}_{i=1,\dots,N}, h_N$

Appendix B: More Generated Samples

B.1 ODE Training with CIFAR10

The following figure is samples generate from generator that we train on CIFAR10 with ODE Solver.

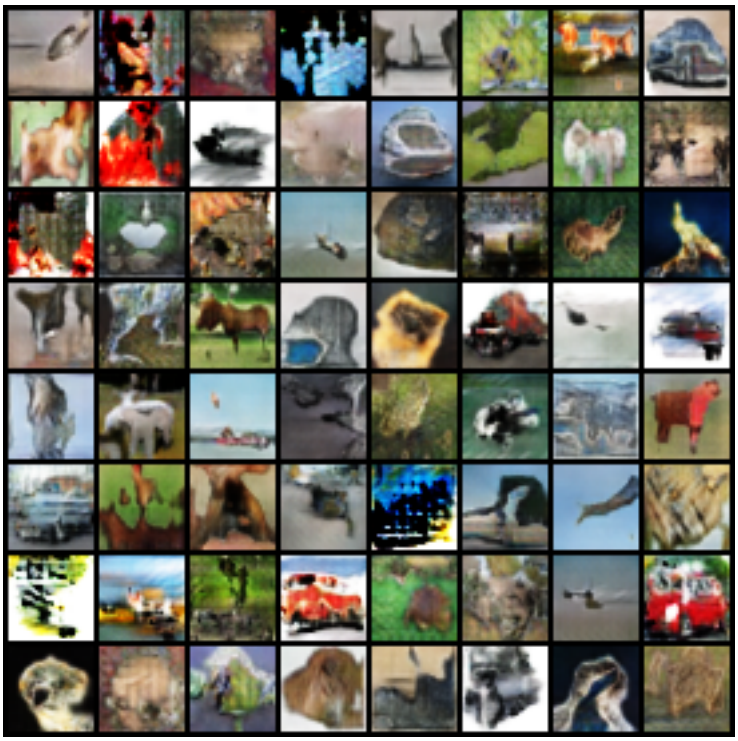

Figure B.1: Samples from CIFAR10

B.2 More samples videos

The following figures are samples when the model learns from all 10 rotating digits . The model fail to learn both in motion and in content. The contents changing overtime like many digits overlap each other.

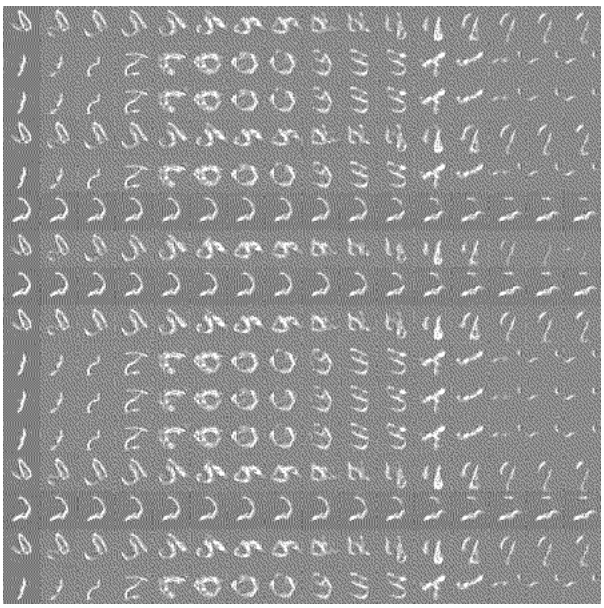

Figure B.2: Samples from MoCoGAN-ODE training from Rotated MNIST

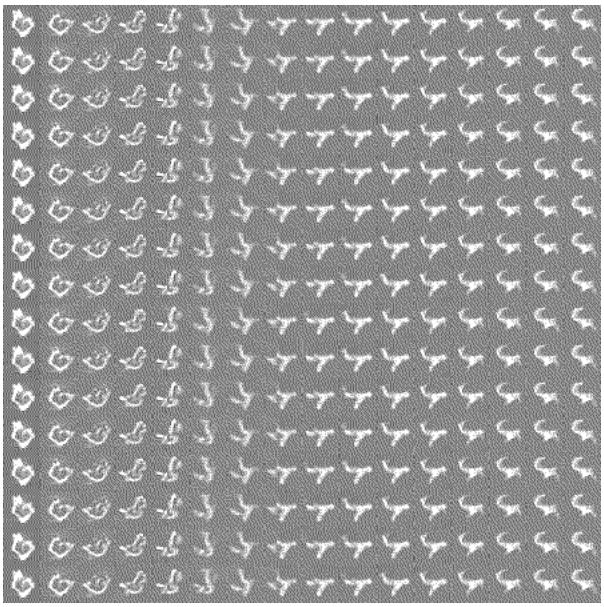

Figure B.3: Samples from MoCoGAN-SDE training from Rotated MNIST

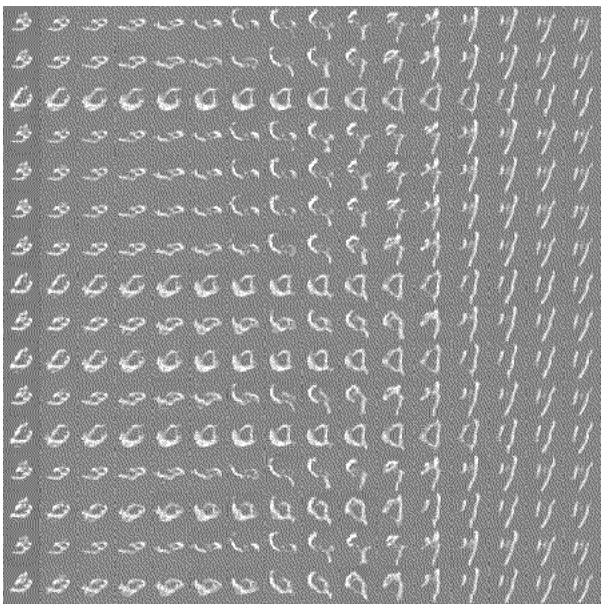

Figure B.4: Samples from MoCoGAN-CDE training from Rotated MNIST

The following sample is generated by generator when we train it on UCF101(Soomro et al., 2012). It's hard to see the motion on static images, but the all image have same motion, this is why we start to think that NODEs is might not be able to learn complex motion space, for the gif version, please refer to https://drive.google.com/drive/folders/1-4C6ppovvXLEVbFG31FgWRMe1X5tswUA?usp=sharing

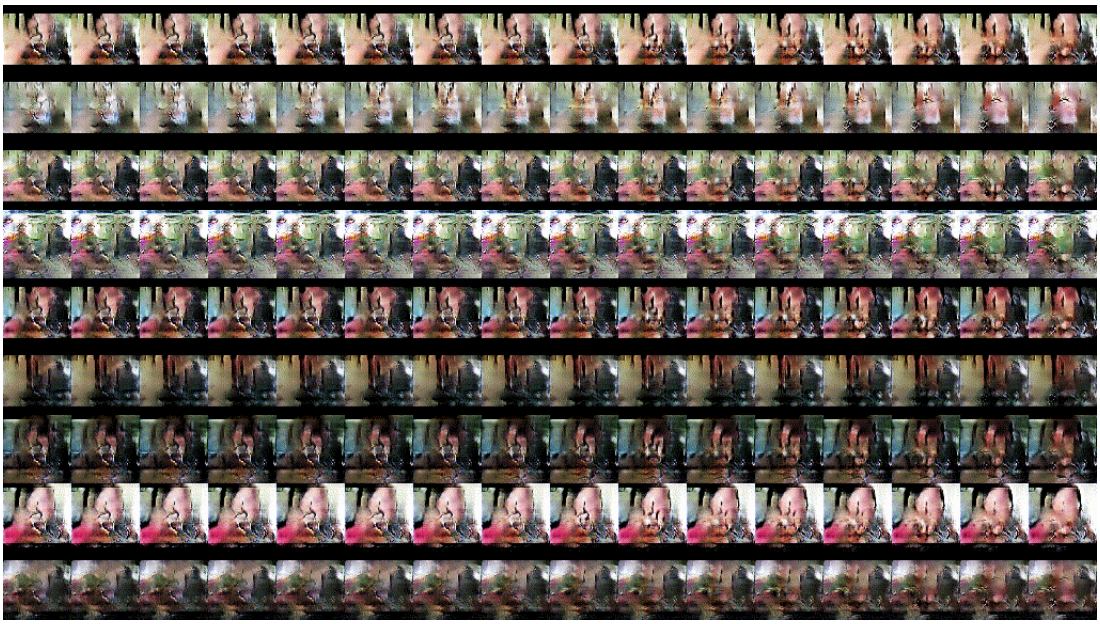

Figure B.5: Samples from MoCoGAN-ODE training on UCF101

Bibliography

An, G., 1996. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8, 3 (1996), 643–674. doi:10.1162/neco.1996.8.3.643.

Arjovsky, M. and Bottou, L., 2017. Towards principled methods for training generative adversarial networks.

Arjovsky, M.; Chintala, S.; and Bottou, L., 2017. Wasserstein gan.

Bai, S.; Kolter, J. Z.; and Koltun, V., 2019. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/01386bd6d8e091c2ab4c7c7de644d37b-Paper.pdf.

Barratt, S. and Sharma, R., 2018. A note on the inception score.

Bishop, C. M., 1995. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7, 1 (1995), 108–116. doi:10.1162/neco.1995.7.1.108.

Bryson, A. E., 1961. Optimal perceptual inference. In *Proceedings of the Harvard Univ. Symposium on digital computers and their applications*. Cambridge: Harvard University Press.

Chen, R. T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K., 2018. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y., 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Clark, A.; Donahue, J.; and Simonyan, K., 2019. Adversarial video generation on complex datasets.

Coddington, A. and Levinson, N., 1955. *Theory of Ordinary Differential Equations*. International series in pure and applied mathematics. McGraw-Hill Companies. ISBN 9780070115422. https://books.google.com.au/books?id=LvNQAAAAMAAJ.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2, 4 (Dec 1989), 303–314. doi:10.1007/BF02551274. https://doi.org/10.1007/BF02551274.

Doersch, C., 2021. Tutorial on variational autoencoders.

Dupont, E.; Doucet, A.; and Teh, Y. W., 2019. Augmented neural odes. In *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf.

Fukushima, K., 1980. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36, 4 (1980), 193–202.

Gastaldi, X., 2017. Shake-shake regularization.

Goodfellow, I., 2017. Nips 2016 tutorial: Generative adversarial networks.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y., 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Gordon, C. and Parde, N., 2021. Latent neural differential equations for video generation. In *NeurIPS 2020 Workshop on Pre-registration in Machine Learning*, vol. 148 of *Proceedings of Machine Learning Research*, 73–86. PMLR. https://proceedings.mlr.press/v148/gordon21a.html.

Gould, S.; Hartley, R.; and Campbell, D. J., 2021. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2021), 1–1. doi:10.1109/tpami.2021.3059462. http://dx.doi.org/10.1109/TPAMI.2021.3059462.

He, K.; Zhang, X.; Ren, S.; and Sun, J., 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. doi:10.1109/CVPR.2016.90.

Hinton, G. E. and Sejnowski, T. J., 1983. Optimal perceptual inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Hochreiter, S. and Schmidhuber, J., 1997. Long Short-Term Memory. *Neural Computation*, 9, 8 (11 1997), 1735–1780. doi:10.1162/neco.1997.9.8.1735. https://doi.org/10.1162/neco.1997.9.8.1735.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4, 2 (1991), 251–257. doi:https://doi.org/10.1016/0893-6080(91)90009-T. https://www.sciencedirect.com/science/article/pii/089360809190009T.

Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; and Weinberger, K. Q., 2016. Deep networks with stochastic depth. In *Computer Vision – ECCV 2016*, 646–661. Springer International Publishing, Cham.

Ioffe, S. and Szegedy, C., 2015. 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 (Lille, France, 2015), 448–456. JMLR.org.

Kaufman, H., 1964. The mathematical theory of optimal processes, by l. s. pontryagin, v. g. boltyanskii, r. v. gamkrelidze, and e. f. mishchenko. authorized translation from the russian. translator: K. n. trirogoff, editor: L.. w. neustadt. interscience publishers (division of john wiley and sons, inc. , new york) 1962. viii 360 pages. *Canadian Mathematical Bulletin*, 7, 3 (1964), 500–500. doi:10.1017/S0008439500032112.

Kelley, H. J., 1960. Gradient theory of optimal flight paths. *ARS Journal*, 30, 10 (1960), 947–954. doi:10.2514/8.5282. https://doi.org/10.2514/8.5282.

Kidger, P.; Morrill, J.; Foster, J.; and Lyons, T. J., 2020. Neural controlled differential equations for irregular time series. In *NeurIPS*. https://proceedings.neurips.cc/paper/2020/hash/4a5876b450b45371f6cfe5047ac8cd45-Abstract.html.

Kingma, D. P. and Ba, J., 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1412.6980.

Kingma, D. P. and Welling, M., 2014. Auto-encoding variational bayes.

Kramer, M. A., 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37, 2 (1991), 233–243. doi:https://doi.org/10.1002/aic.690370209. https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209.

Krantz, S. and Parks, H., 2002. *The Implicit Function Theorem: History, Theory, and Applications*. Modern Birkhäuser classics. Birkhäuser. ISBN 9780817642853. https://books.google.com.au/books?id=ya5yy5EPFD0C.

LeCun, Y.; Haffner, P.; Bottou, L.; and Bengio, Y., 1999. *Object Recognition with Gradient-Based Learning*, 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-46805-9. doi:10.1007/3-540-46805-6_19. https://doi.org/10.1007/3-540-46805-6_19.

Ledig, C.; Theis, L.; Huszar, F.; Caballero, J.; Cunningham, A.; Acosta, A.; Aitken, A.; Tejani, A.; Totz, J.; Wang, Z.; and Shi, W., 2017. Photo-realistic single image super-resolution using a generative adversarial network.

Liu, X.; Xiao, T.; Si, S.; Cao, Q.; Kumar, S.; and Hsieh, C.-J., 2019. Neural sde: Stabilizing neural ode networks with stochastic noise.

Miyato, T.; Kataoka, T.; Koyama, M.; and Yoshida, Y., 2018. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=B1QRgziT-.

Nguyen, A.; Clune, J.; Bengio, Y.; Dosovitskiy, A.; and Yosinski, J., 2017. Plug and play generative networks: Conditional iterative generation of images in latent space.

Øksendal, B., 2010. *Stochastic Differential Equations: An Introduction with Applications*. Universitext. Springer Berlin Heidelberg. ISBN 9783642143946. https://books.google.com.au/books?id=EQZEAAAAQBAJ.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library.

Poli, M.; Massaroli, S.; Park, J.; Yamashita, A.; Asama, H.; and Park, J., 2019. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, (2019).

Qin, C.; Wu, Y.; Springenberg, J. T.; Brock, A.; Donahue, J.; Lillicrap, T.; and Kohli, P., 2020. Training generative adversarial networks by solving ordinary differential equations. In *Advances in Neural Information Processing Systems*, vol. 33,

5599–5609. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/3c8f9a173f749710d6377d3150cf90da-Paper.pdf.

Rubanova, Y.; Chen, R. T. Q.; and Duvenaud, D., 2019. Latent odes for irregularly-sampled time series.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J., 1986. Learning representations by back-propagating errors. *Nature*, 323, 6088 (Oct 1986), 533–536. doi:10.1038/323533a0. https://doi.org/10.1038/323533a0.

Saito, M.; Matsumoto, E.; and Saito, S., 2017. Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Saito, M.; Saito, S.; Koyama, M.; and Kobayashi, S., 2020. Train sparsely, generate densely: Memory-efficient unsupervised training of high-resolution temporal gan. *International Journal of Computer Vision*, 128, 10-11 (May 2020), 2586–2606. doi:10.1007/s11263-020-01333-y. http://dx.doi.org/10.1007/s11263-020-01333-y.

Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; Chen, X.; and Chen, X., 2016. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf.

Santurkar, S.; Schmidt, L.; and Madry, A., 2018. A classification-based study of covariate shift in gan distributions.

Schmidhuber, J., 2020. Generative adversarial networks are special cases of artificial curiosity (1990) and also closely related to predictability minimization (1991).

Song, Y. and Ermon, S., 2020a. Generative modeling by estimating gradients of the data distribution.

Song, Y. and Ermon, S., 2020b. Improved techniques for training score-based generative models.

Song, Y.; Sohl-Dickstein, J.; Kingma, D. P.; Kumar, A.; Ermon, S.; and Poole, B., 2021. Score-based generative modeling through stochastic differential equations.

Soomro, K.; Zamir, A. R.; and Shah, M., 2012. Ucf101: A dataset of 101 human actions classes from videos in the wild.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html.

Tulyakov, S.; Liu, M.-Y.; Yang, X.; and Kautz, J., 2018. Mocogan: Decomposing motion and content for video generation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1526–1535. doi:10.1109/CVPR.2018.00165.

Ulyanov, D.; Vedaldi, A.; and Lempitsky, V., 2017. Instance normalization: The missing ingredient for fast stylization.

Vondrick, C.; Pirsiavash, H.; and Torralba, A., 2016. Generating videos with scene dynamics. In *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/file/04025959b191f8f9de3f924f0940515f-Paper.pdf.

Yildiz, C.; Heinonen, M.; and Lahdesmaki, H., 2019. Ode2vae: Deep generative second order odes with bayesian neural networks. In *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/99a401435dcb65c4008d3ad22c8cdad0-Paper.pdf.

Younes, L., 2010. *Shapes and Diffeomorphisms*. Applied Mathematical Sciences. Springer Berlin Heidelberg. ISBN 9783642120558. https://books.google.com.au/books?id=SdTBtMGgeAUC.

Yushchenko, V.; Araslanov, N.; and Roth, S., 2019. Markov decision process for video generation.

Zhu, J.-Y.; Krähenbühl, P.; Shechtman, E.; and Efros, A. A., 2016. Generative visual manipulation on the natural image manifold. In *Computer Vision – ECCV 2016*, 597–613. Springer International Publishing, Cham.