Department of Informatics
(International Master's Program)

Technische Universität München

Master's Thesis

Identification of stochastic differential equations with Artificial Neural Networks

Asima Azmat

Department of Informatics
(International Master's Program)

Technische Universität München

Master's Thesis

Identification of stochastic differential equations with Artificial Neural Networks

Identifizierung stochastischer Differentialgleichungen mit künstlichen neuronalen Netzen

Author:	Asima Azmat
Examiner:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	Dr. Felix Dietrich
Submission Date:	September 15th, 2021

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, September 15th, 2021 Asima Azmat

Acknowledgments

I would like to thank my supervisor Dr. Felix Dietrich whose novel research and work has inspired this thesis project. His guidance, support and pep talks have kept me motivated and enthusiastic to give my best to this research.

I would also like to thank Prof. Dr. Hans-Joachim Bungartz, who gave me this oppourtunity to work on this amazing project.

"If we knew what it was we were doing, it would not be called research, would it?"

-Albert Einstein

Abstract

Transforming observed data from a system of interest into mathematical models that provide an insight into the dynamics of the system is the paramount of this research. In this project we introduce a machine learning based approach for identifying stochastic dynamical systems from data. We bring together classical tools from three different areas of computer software, namely: agent based modelling and simulation systems, stochastic differential equations and artificial neural networks. Our system of interest is the spread of a contagious viral disease, we study spread of the virus in a group of people who move collectively in a dynamic crowd. Microscopic crowd dynamics are simulated using an open source pedestrians and crowd dynamics simulation software called Vadere. Data of fine-grained particle- or agent-based simulations is obtained from Vadere and from this microscopic simulation data we compute the values of coarse observable dynamics of virus spread in the simulated population over time. The spread of the virus is a stochastic system that can be approximated by stochastic differential equations (SDEs). We use an artificial neural network (ANN) to approximate drift and diffusivity functions that define these SDEs. The loss function of our neural network is inspired by stochastic numerical integration schemes, in this work, we focus on the Euler-Maruyama scheme. SDEs that we learn using ANN serve as surrogate models that can be used to reconstruct the coarse observations of fine-scale dynamics of a stochastic system. We test this method on two experiments carefully crafted to showcase the effectiveness of this approach, how it allows us to accurately learn the stochastic dynamics, and how it predicts future states of the system.

Contents

1 Introduction

This chapter introduces the work, it provides a birds-eye-view of the whole project. It starts with describing the problem that we intend to solve, and then moves on to introduce the goals, objectives and motivations of this work. In section 1.1; we explain the problem at hand that this project aims to solve and outlines different parts of the problem. Section 1.2, sheds a light on some of our motivations behind this work. In section 1.3, we present the goals and objectives that we try to achieve during the lifetime of this project. Finally section 1.4, outlines the structure of the rest of this thesis document.

1.1 Problem Description

Complex dynamical systems are generally modelled using agent based models. Agent based modelling and simulation software are very thorough and capture great detail within the produced simulations. Nevertheless, this same accuracy makes this software very complex, computationally expensive and time consuming. Although all of these details are necessary to get a complete picture of the dynamical system. All of these fine-grained details might not be relevant if we are studying a specific aspect of the system, an underlying system or a subset of a big system.

The result of these complex agent based simulations is a large dataset that contains myriads of information about the underlying stochastic systems. In order to turn this disadvantage into an advantage; we can learn surrogate models from this data, such as stochastic differential equations. If the coarse state contains enough information, these surrogate models can predict futures states of the system without having to know the microscopic fine-grained dynamics. In order to learn surrogate models from data, we need a ”physics-informed” algorithm that is able to identify the dynamical system from data.

System identification and modelling can have many purposes and uses. The principal benefit of system identification and modelling is that it provides a deep and thorough understanding of the system under study. Modelling of complex dynamical systems can be done accurately with stochastic differential equations, while omitting the unnecessary details. One specific dynamical system that we focus on during this project is spread of a viral disease, we call it our system of interest. Understanding this specific system enables researchers to model it adequately and helps in devising disease spread prevention strategies.

A high level overview of the proposed solution to this problem is provided in figure(1.1).

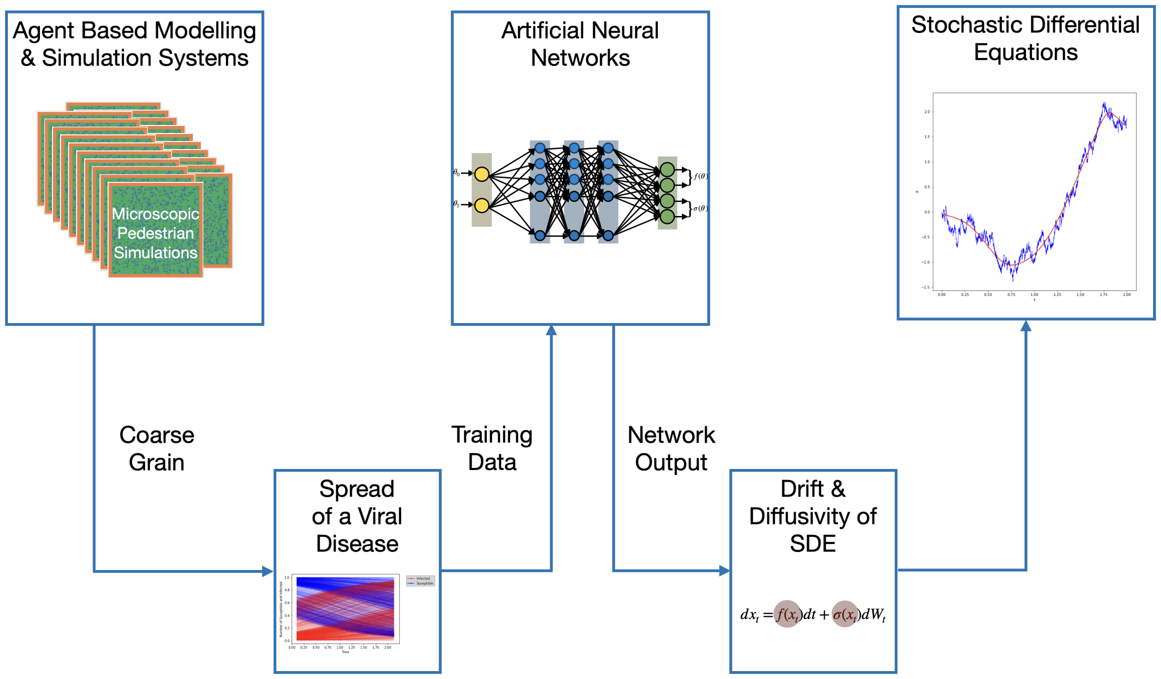

Agent Based Modelling & Simulation Systems
Microscopic Pedestrian Simulations
Artificial Neural Networks
Stochastic Differential Equations
Coarse Grain
Training Data
Network Output
Spread of a Viral Disease
Drift & Diffusivity of SDE
$dx_t = f(x_t)dt + \sigma(x_t)dW_t$

Figure 1.1: A bird's eye view of the project

1.2 Motivations

The problem that we have at hand and the solution that we present to it is three-fold. We have been motivated by three different aspect of this idea as well.

First motivation of this work stems in the amount of data that we have available, as we already discussed the huge amounts of data is produced by agent based simulation systems. The "physics-informed" or "grey-box" identification algorithms enhanced to handle stochastic data are a natural fit for such problems, because they already encode a lot of structure into the algorithm (for example, the SDE structure), so that not as much training data is needed.

Our second motivation originates from the numerical analysis or backward error analysis of stochastic integration schemes. These integration schemes iteratively produce discrete time data that can be exploited by our identification algorithm to perfect its predictions iteratively. Note that this type of analysis is not done in the thesis, but could be part of the future work.

Last but not the least motivation comes from the application side. Macroscopic dynamics of agent-based simulations have long been modelled using effective SDEs. Given fine-grained microscopic simulation data, our work extracts the macroscopic dynamics that can then be modelled using coarse or effective SDEs. One possible application of this work that has been our biggest motivation is to understand the spread of a viral disease on a coarse scale, for example, on the scale of a building or a small town. By understanding the spread of viral diseases, better spread prevention techniques can be employed to contain and eventually eliminate it.

1.3 Objectives

In this section we will define some objectives that serve as a guiding light throughout this project. We also briefly touch upon the strategy we used to achieve each objective. The structure of this research is subsequently based on achieving these objectives one by one and will be explained in detail in the later chapters. Our system of interest, that we want to study is the spread of a viral disease.

- **Generate Fine-grained Microscopic Simulations:** First objective that we have is to simulate the dynamics of our system of interest using a particle- or agent-based modelling and simulation system. For this purpose we use an agent based modelling and simulation framework. This framework simulates microscopic pedestrian and crowd dynamics resulting in very detailed simulations. These simulations result in a dataset containing fine-grained microscopic dynamics of our system of interest.
- **Compute the Values of Coarse Observables:** Our second objective is to use the previously obtained microscopic complex simulations to compute the values of coarse observables. This process results in a dataset that contains only the details that are relevant to our system of interest.
- **Set-up System Identification Algorithm:** Third objective is to setup an algorithm that is able to identify and learn the dynamics of our system of interest. We use a physics informed grey-box neural network for this purpose and pre-process our dataset accordingly. We take inspiration from stochastic numerical integrators such as Euler-Maruyama to setup the training of our neural network.
- **Learn a Surrogate Model:** Lastly, we train the previously setup algorithm with the coarse observable dataset we now have. This results in the algorithm learning underlying stochastic differential equations governing our system of interest. These learned SDEs act as coarse surrogate models that does not need to know the fine-grained dynamics in order to predict future states of our system.

1.4 Document Structure

The rest of this document is organized as follows.

Chapter 2, provides an overview of the areas that lend themselves as building blocks of this project. We discuss foundational concepts of agent based modelling and simulation systems, stochastic different equations and artificial neural networks in this chapter. Explore state of the art work in these areas and how it has led to and inspired this project.

Chapter 3, focuses on the details of the work that has been done during this projects and presents the empirical results. Illustrates the experiments that have been reproduced from state of the art work and then describes experiments designed during this project. Firstly, we explain the architecture of the neural network to be used and its loss function. Secondly, we explore the generation of data for training, validation and testing of the neural network. Then we discuss the evaluation matrices that are subsequently used to measure the success of our experiments.

In **Chapter 4**, we discuss the results of our experiments and future work that can be done in the direction of this research.

2 State of the Art

This chapter discusses the foundational concepts that are necessary to understand our work and state of the art work that has been done in these areas so far. Our work brings three realms of computer science together and proposes a solution that uses these three areas as depicted in figure (1.1). All three of them are already well established and researched:

- Agent Based Modelling and Simulation Systems
- Stochastic Differential Equations
- Artificial Neural Networks

In section 2.1, we will introduce agent based simulation systems, their core concepts that are needed as a foundation of this work and their limitations. We also introduce an agent based modelling and simulation system and a locomotion model we will be using in our work. In section 2.2, we will discuss stochastic systems, stochastic differential equations and stochastic integration schemes; as well as their relevance and importance for our work. Section 2.3 introduces artificial neural networks, the type of neural network we use in our work, its loss function, some other terminology we will be using later on and cutting edge research in this area. Section 2.4 discusses the paper by Dr. Dietrich that inspired this work and forms the bases of this project [20].

2.1 Agent Based Modelling and Simulation Systems

Agent based modelling aims to model the behaviour of a real world entity in an environment that is somewhat based on reality. Agent based modelling simulates a complex system that contains a number of entities that are interacting with each other at a microscopic-level. The entities modelled within an agent based model are a set of autonomous decision-making objects called agents. Generally agent based models are implemented in the form of simulation software that represents an abstraction of reality. Agents in this simulated environment have a set behaviour determined by a set of rules. Agents also interact with other agents within the simulated environment, these interactions also have an influence on their actions. Agents can be given a goal that they try to achieve within the simulation time. Each agent navigates the simulated environment step by step, at each step accessing their situation with respect to the environment and other agents [49].

2.1.1 Example Agent based model

Microscopic traffic simulation is widely used for traffic research and prediction [46]. A good example of an agent-based model can be derived from such traffic simulation systems, let us suppose that there is a group of cars moving on a highway. All the vehicles present on the highway constitute a large system with highway as its environment. Each vehicle with a driver can be considered an agent. Each agent within this system can have a certain behaviour independent of other agents. Each driver interacts with other drivers and can perform certain actions; such as move forwards, stay in place, sound a horn, change lanes etc. Next action of each driver depends on the environment, situation and their interaction with other drivers.

Actions of individual agents in an ABM have an impact on other agents they come in contact with and can even change overall state of the system as well. For instance if the driver in front slows down, the driver behind him will also slow down. This depicts impact of an action of one agent on another agent they come in contact with. In another situation if one driver crashes or start moving in a wrong direction, it might cause a traffic jam. Such an action changes the overall state of the system that was previously moving forward and puts it into a traffic jam. Systems that react to a certain amount of external noise are called ”stochastic systems”. In the case of car traffic, such external noise may be introduced by not modeling individual drivers intentions, but randomly brake in certain random intervals.

2.1.2 Limitations of agent based models

In order to further underline biggest motivations of our work, we would like to highlight some limitations of agent based modelling.

1. Agent based models are not general purpose, they have to have right level of abstraction with just the right amount of details to be useful [10]. ABMs that specialize in simulating one system usually can not simulate other similar systems.

2. Agent based modelling is multi-faceted and contain unnecessary details if the system of interest is rather specific. For instance, we can use crowd simulation software to model spread of a disease in the crowd. However, if our system of interest is ”spread of disease”; we still have to model each pedestrian at microscopic level [41].

3. Agent based modelling and simulation software tool-kits have performance limitations; they are not designed for extensive simulations and use huge amounts of computation power [4].

4. ABMs take much longer time to simulate complex scenarios, making them highly time consuming [14].

2.1.3 Vadere

In our work we use an open source simulation software called Vadere. Vadere is a framework that simulates the microscopic pedestrians and crowd dynamics [40]. It is developed by the research group of Gerta Köster at the Department of Computer Science and Mathematics at the Munich University of Applied Sciences. Many other researchers and institutes also contribute to it and use it for their research. There are many other microscopic pedestrian simulation softwares as well, some of them are: Brahms, HLA_Agent, HLA_RePast, PDES-MAS, PedSim, Repast-J/Repast-3, Repast HPC, Repast Simphony (2D/3D), SeSAm, Simio (2D/3D) [4]. We use Vadere because it is free, open-source and thoroughly tested, and it already contained a model for the spread of a viral disease.

Vadere focuses on locomotion level models to simulate realistic pedestrian behaviours. This provides a strong basis for more complex models that can be built on top of it. Each Vadere simulation corresponds to a scenario which is hypothetical or is an abstraction of reality.

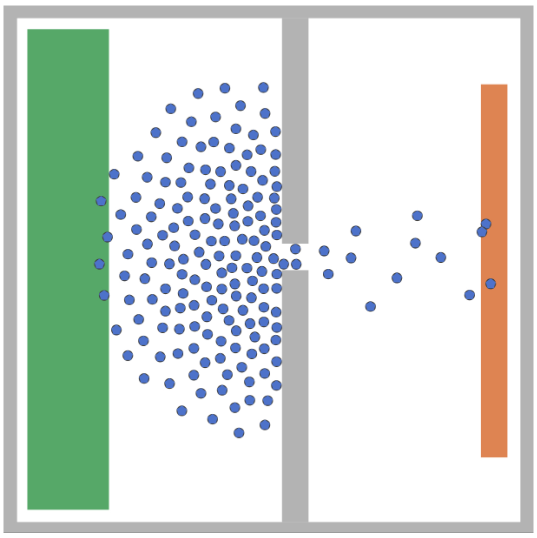


Figure 2.1: The four basic elements of simulations: agents (blue) who move from a source (green) to a destination (orange) while avoiding obstacles (grey)

As shown in figure (2.1), each scenario contains four basic topographic elements:

1. Agent: that are simulated pedestrian (blue)
2. Source: is a designated area within simulated space, where agents are created (green)
3. Target: is a designated area where agents aim to reach (orange)
4. Obstacles: are parts of simulation that agents are unable to pass through (grey)

Vadere provides an easy to use graphical user interface (GUI). Vadere GUI can be used to define different aspects of simulation such as simulation properties, locomotion model it will use, topographical elements etc. Each simulation runs for a set amount of time and has a set time step length. After each time step, new positions of all the agents in the simulation are changed and recorded.

2.1.4 Optimal Steps Model

There are multiple locomotion models available in Vadere such as Gradient navigation model (GNM) [21], Behavioral heuristics model (BHM) [65]. We use Optimal Steps Model (OSM) developed by Seitz and Köster in [64]. OSM have been adjusted and extended to fit the findings from interdisciplinary experiments and studies. For example, it was extended based on the (inter-) personal space theory [73][72] and the social identity theory [71]. OSM describes every step that and agent will take during a simulation. Every agent takes a step to its next position within a circle around its current position. The radius of the circle is equal to the agent's step length. Next optimal step is based on a utility function which is a combination of smaller utility functions based on the goals that it has. In OSM every agent has three goals:

1. Reaching the destination based on the current position and destination, represented by u_t
2. Avoiding obstacles, represented by $u_{o,j}$
3. Keeping a certain minimum distance from all other agents, represented by $u_{p,j}$

The overall utility u is calculated in equation(2.1) as the sum of the target utility u_t, the smallest of the m (negative) obstacle utilities $u_{o,j}$, and the sum of all n pedestrian utilities $u_{p,j}$ [63]:

$$u = u_t + \min_{j \in 1 \ldots m} u_{o,j} + \sum_{i=1}^{n} u_{p,j}. \tag{2.1}$$

2.1.5 Spread of a Viral Disease

One of the threats facing human societies has been spread of different types of diseases. Diseases spread in many different ways, specifically viral disease can spread through a variety of ways. Some viruses can spread through blood transplant, insects, touch, saliva, contaminated surfaces or sometimes through air as well. Some very deadly diseases caused by viruses include measles, influenza, COVID-19 [48], HIV, smallpox. Viruses can be divided in different categories based on many different factors such as spread-ability, transmissiblity, chemical and physical properties etc [2].

If a virus starts spreading rapidly in a large number of people in a population within a short amount of time, it is called an epidemic[McNamara20181P]. Epidemiology is a vast field in itself, where scientists study and analyze spread of different diseases. This study is done to identify the patterns in disease spreads, which help epidemiologists devise strategies to contain the epidemic. Several mathematical models are developed to enhance the data available to epidemiologists [17]. Mathematical models combined with statistical analysis gives epidemiologists powerful tools to make meaning analysis and devise effective strategies.

In order to mathematically model infectious diseases, compartmental models are often used in epidemiology. The origin of such models is the early 20th century, with important works being that of Ross [61], Hudson [62], Kermack and McKendrick [38] and Kendall [37]. Compartmental models are a general modelling techniques, several specific models have been developed for specific cases. Most basic compartmental model is SIR model, SEIR model is a variation of SIR model that we will be using in our work.

2.1.6 SEIR Model

A locomotion model can have attributes that add another layer of information about the agents. One such attribute in the optimal steps model is based on SEIR model. We use SEIR model implemented by Marc Marot-Lassauzaie and Victoria Dahmen at TUM. In SEIR model, agents are assigned to one of four groups:

1. **S**usceptible: agents that are likely to be infected once they come in contact with an infected agent
2. **E**xposed: agents that have already come in contact with an infected agent
3. **I**nfectious: agents that already infected and can pass it onto other agents
4. **R**ecovered: agents that have recovered from the infection

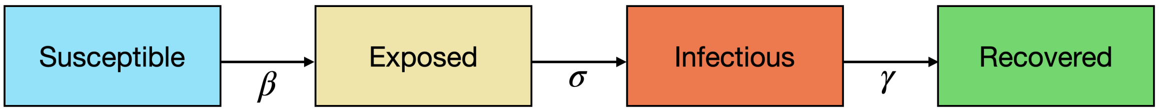


Figure 2.2: SEIR Model with transition rates

Figure (2.2) depicts how SEIR model works along with the concept of transition rates. Transition rate is the rate with which agents go from one state to another. Susceptible agents can only go to Exposed state when they come in contact with an infected agent and β represents the infection rate. β represents the probability of disease spreads from infected agent to a susceptible agent. σ is the incubation rate which represents the rate at which the exposed agent becomes infectious. γ represents the recovery rate at which infected agents recover from the disease [32].

In a constant populations with no new births or deaths, if all agents are mixing infinitely fast (no short range effects), and if there is an effectively infinite amount of agents (no finite-size effects), the SEIR model can be represented using following equations(2.2, 2.3, 2.4, 2.5) [1]:

$$\frac{dS}{dt} = -\frac{\beta SI}{N} \tag{2.2}$$

$$\frac{dE}{dt} = \frac{\beta SI}{N} - \sigma E \tag{2.3}$$

$$\frac{dI}{dt} = \sigma E - \gamma I \tag{2.4}$$

$$\frac{dR}{dt} = \gamma I \tag{2.5}$$

where $N = S + E + I + R$ is the number of total population

Figure(2.3) shows how number of agents in each S,I,R group increase or decreases over time in a classical SIR model. The number of infected agents increases as the disease spreads over time and then decreases. This decrease happens due to agents being recovered over time. Whereas, number of susceptible agents only decreases over time. Contrarily, number of recovered agents only increases over time given the assumption that they do not reinfect. In SIR as well as SEIR model do not take into account the deceased population, which can be modelled using other compartmental models.

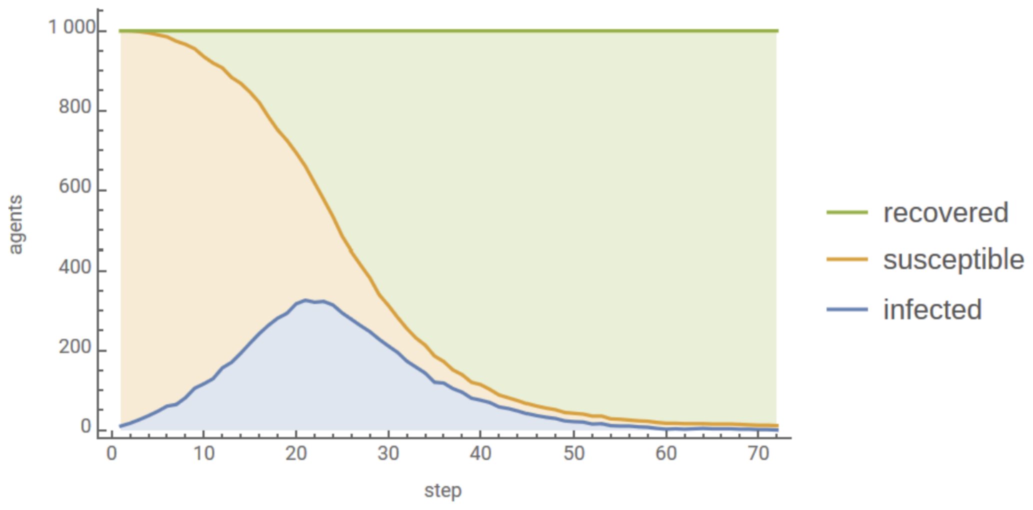


Figure 2.3: Number of Susceptible, Infected and Recovered agents in SIR Model over time

2.2 Stochastic Differential Equations

Stochastic differential equations serve as a coarse-surrogate model of a stochastic system. In order to understand Stochastic Differential Equations (SDEs), we will start by developing an understanding of Stochastic systems first. Then we will explore what stochastic differential equations are, integration schemes used to approximately solves SDEs and state of the art work in this area.

2.2.1 Stochastic Systems

A stochastic system is a system that contains at least one stochastic process. The theory of stochastic processes started with Einstein's work on the theory of Brownian motion and a series of papers published afterwards. However, the development of theory of stochastic process was done afterwards by scientists including Smoluchowksi, Planck, Kramers, Chandrasekhar, Wiener, Kolmogorov, Itô, Doob [54]. A stochastic process can be defined as a process describing a random phenomenon with an element of uncertainty that evolves over time. In probability theory, a stochastic process is defined as a collection of random variables. The mathematical definition of stochastic process is given below, quoted from [54].

"Let T be an ordered set, (Ω, F, P) a probability space and (E, G) a measurable space. A stochastic process is a collection of random variables $X = X_t; t \in T$ where, for each fixed $t \in T, Xt$ is a random variable from (Ω, F, P) to (E, G). Ω is called the sample space and E is the state space of the stochastic process X_t."

Given a set of inputs, a stochastic process can produce different output at different times. An example of stochastic process can be the movement of a gas molecule [70]. Stochastic processes find their applications in all areas of natural sciences such as mathematics, physics, finance, biology, engineering and even in quantum mechanics. Familiar examples of stochastic processs include stock market [66], climate modelling [29] and crowd dynamics [55]. Different types of stochastic processes are used for modelling in different contexts. Some basic types of stochastic processes include but are not limited to:

1. Markov process
2. Poisson process
3. Wiener process
4. Bernoulli process

Stochastic processes are used to model random phenomena using Stochastic differential equations (SDEs), which we well explore in later sections.

2.2.2 Example Stochastic system

An example of a stochastic system can be number of people in an Oktoberfest tent, let's denote it with X. As t which is a representation of time changes, X also changes as people enter the tent or leave. X depends on a lot of factors such as the time of the day, weather, quality of the beer etc. These external factors add an element of uncertainty in the system that makes it stochastic. If we start with a certain number of people X_t in the tent and take a step h in time, the output of the system at time $t+h$ denoted by X_{t+h} can not be predicted with 100% certainty. Output of the system also depends on the choice of time step h. On one hand, if we take smaller value of h, the system might change very little. On the other hand, if h is considerably large, the number of people in the tent might have changed by a few hundreds.

Another example of stochastic system, where stochastic modelling is often applied is stock market. In figure (2.4), two simulations of stochastic modeling of stock price behavior in Ghanaian Stock Exchange can be seen [6].

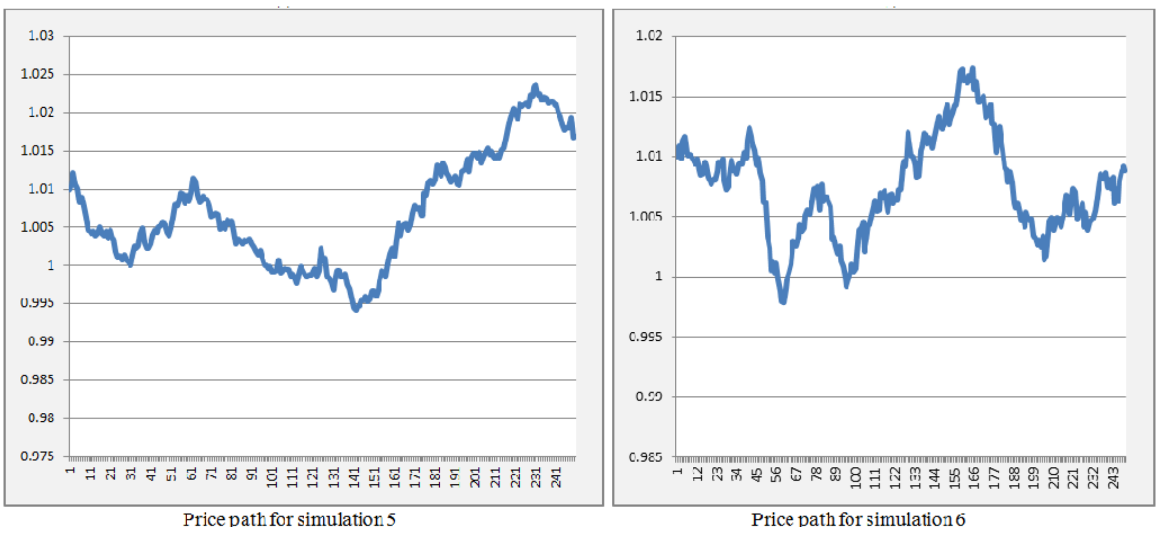


Figure 2.4: Stochastic modelling of stock prices in the Ghanaian Stock Exchange [6]

2.2.3 Stochastic Differential Equations

Stochastic system is either a representation of a physical process that occurs in nature such as movement of gases or a social process such as crowd dynamics. In order to model physical processes deterministically, ordinary differential equations (ODEs) are used. ODEs describes the relationship between a physical process and its rate of change. Just like ordinary differential equations, SDEs also define a relationship between a function and its rate of change. Additionally, they provide a better representation of the function, because they include the possibility of random effects disturbing the system. A stochastic differential equation (SDE) is a differential equation containing at least one term that is a stochastic process. SDEs contain a Gaussian white noise variable, calculated as the derivative of a stochastic process [23].

Theoretical foundations of SDEs are well-established, we consult the standard literature on stochastic processes [54], stochastic calculus [36], and stochastic algorithms [56]. SDEs are used to solve many different problems in machine learning and find their use in many different areas. SDEs are widely used for generative modelling, a great example of which are generative-adversarial networks [26, 34]. Long-Short-Term-Memory networks are used in a stochastic setting to generate hand-written text [35]. A framework has been developed that learns an SDE for training and sampling of score-based models, this framework allows sample generation [67].

2.2.4 SDEs vs ODEs

A depiction of a stochastic Wiener process' evolution over time can be seen in figure(2.5). As it can be seen in the figure that the same process can be modelled in two different ways. We modelled using a stochastic differential equation (SDE) and displayed it as a blue line graph. There is also a red line graph that shows the ordinary differential equation (ODE) model of the same system. On one hand the ordinary differential equation model does not take into account the random effects in the environment and thus predicts a pretty smooth evolution over time. While on the other hand, stochastic differential equations also include the random effects in the environment interfering with the system. Hence, an SDE provides a good model of uncertainties in the system, while an ODE ignores uncertainties and only predicts the evolution of the overall system as the time passes. SDE can be imagined as an ODE with an added element the quantifies uncertainty.

2.2.5 Mathematical Definition of an SDE

Let W_t denote a multidimensional Wiener process, with zero initial state $W_0 = 0$. If x_t is the value of x at time t, $f(x_t)$ is a smooth vector-valued function and $\sigma(x_t)$ a smooth matrix valued function. Mathematically Stochastic Differential Equations are defined as follows:

$$dx_t = f(x_t)dt + \sigma(x_t)dW_t \tag{2.6}$$

where $f : R^n \to R^n$ and $\sigma : R^n \to R^{n\times n}$ are smooth, possibly nonlinear functions; every $\sigma(x)$ is positive and bounded away from zero (and a positive-definite matrix if $n > 1$), and W_t a Wiener process such that for $t > s$, $W_t - W_s \sim N(0, t - s)$. SDEs act as a surrogate model of the stochastic process that can predict its future state without having to know fine-grained microscopic states of the system.

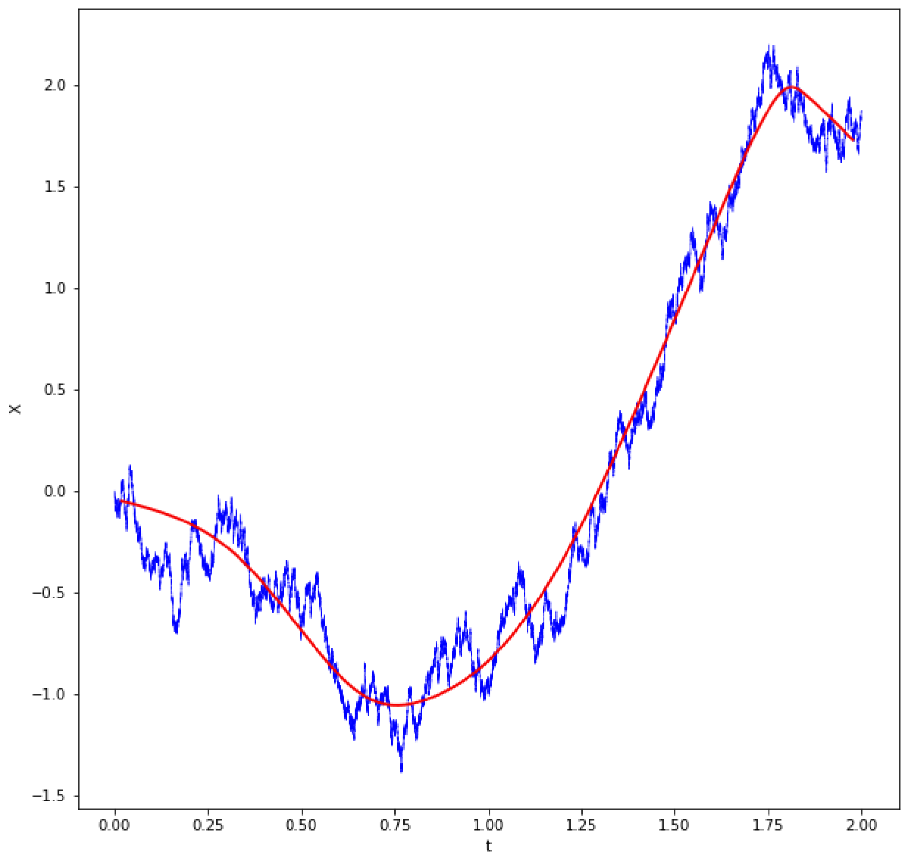


Figure 2.5: Depiction of SDE (blue) and ODE (red) paths of a Process

2.2.6 Drift and Diffusivity

In equation (2.6) the coefficient $f(x_t)$ and the coefficient $\sigma(x_t)$ are a vector valued function and a matrix valued function, respectively. These two coefficients are respectively called drift and diffusivity of the stochastic system. Drift and diffusivity in an SDE are generally unknown and need to be approximated in order to find an effective solution of the SDE.

In order to understand what drift and diffusivity are in the physical world, let's take the example of a boat going in the water. Consider that X_t is the state of X at time t. W_t is the noise in the environment that makes this system stochastic, examples of this noise can be the waves in the water as well as the wind that blows around the boat. Both the waves and the wind randomly affect the speed of the boat. $f(x_t)$ which is called as drift is the speed of the boat. $\sigma(x_t)$ called as diffusivity is the measure of influence of the noise W_t on the speed of the boat. In this example, if we know the drift and diffusivity of this stochastic system we can use them to predict the future states of the system.

Diffusivity Matrix valued function that we want to learn can have three different types:

- Diagonal: in a diagonal diffusivity matrix only the n diagonal entries of the $n \times n$ matrix can be learned
- Triangular: in a triangular diffusivity matrix, entries on and below the diagonal can be learned
- Symmetric Positive Definite (SPD): SPD diffusivity matrix is similar to the triangular diffusivity, however, the learned lower triangular matrix is multiplied with its transpose to obtain an SPD matrix(D): $D = L.L^T$

2.2.7 Euler-Maruyama Scheme

Generally SDEs are impossible to solve in closed form and must be tackled approximately using numerical methods [19]. For the simulation of solutions to SDE models, a numerical method can be used. The Euler-Maruyama(EM) method is one of the simplest methods that can be used for this purpose [33]. This approach of using forward-Euler integrator in neural network is not new. It has been successfully employed to create recurrent neural networks that are based on simple ODE integrator schemes [59]. It has also been used to develop very deep neural network architectures in the past such a residual networks [30]. This approach also forms the basis of deep Hamiltonian networks based on symplectic integrators [27, 12, 77].

The use of integration schemes, particularly stochastic integrators, has been proposed by Dietrich et al. in [20] which we explore and experiment with in this project. Integration schemes such as Euler-Maruyama iteratively produce discrete time data, we mathematically inform the neural network architecture the structure of the numerical integrator. This enables us to take advantage of ”backward error analysis” of these integration schemes [78].

Given an SDE of the form presented in equation (2.6), Euler-Maruyama is used to predict the state of the system x_1 after a small time step h. Mathematically, Euler-Maruyama is described as follows:

$$x_1 = x_0 + hf(x_0) + \sigma(x_0)\delta W_0 \tag{2.7}$$

x_1 in equation(2.7) is modelled using a Gaussian distribution. It is a point drawn from a multivariate normal distribution, conditioned on x_0 and h as shown in equation(2.8).

$$x_1 \sim \mathcal{N}(x_0 + hf(x_0), h\sigma(x_0)^2) \tag{2.8}$$

Other numerical integration schemes are also used for approximate numerical solution of SDEs such as Milstein method and Runge-Kutta method which is a generalization of Runge-Kutta method for ODEs. Numerical integration schemes such as Euler-Maruyama provide a pretty accurate solution of an SDE, as it is evident from a comparison done by Bayram et al. in figure(2.6) [9].

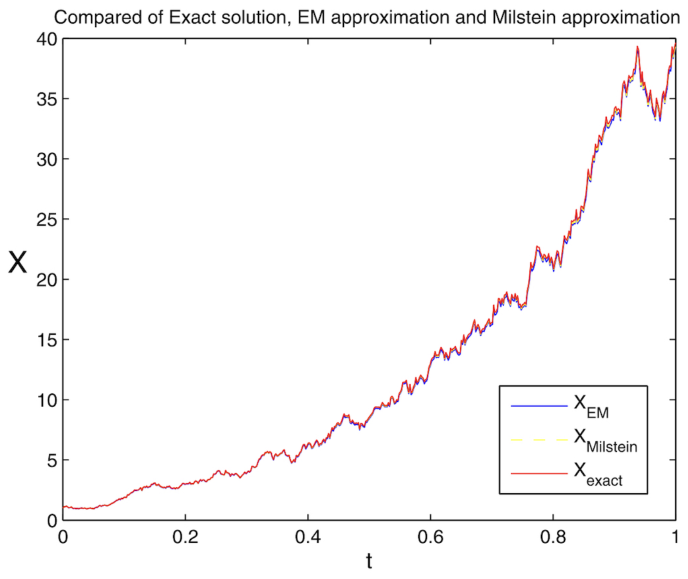
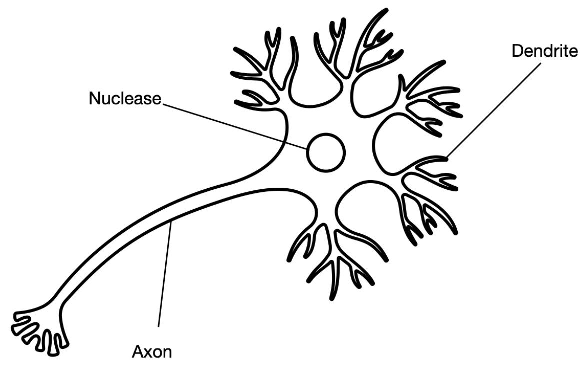


Figure 2.6: Comparison of SDE paths generated by Euler-Maruyama and Milstein method with exact solution [9]

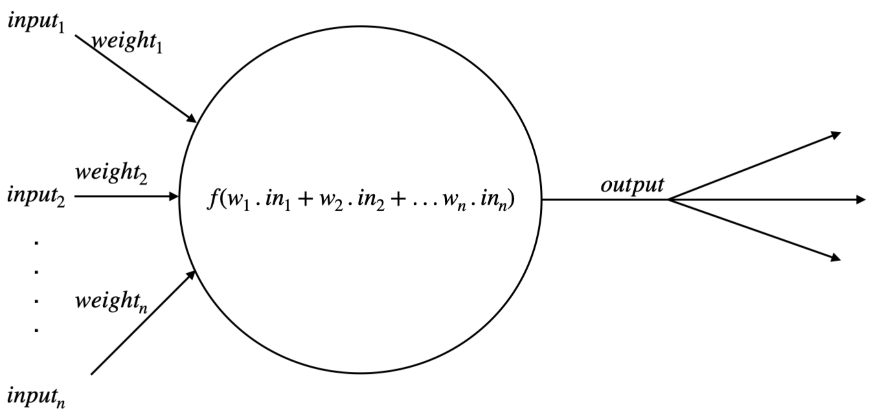
2.3 Artificial Neural Networks

Artificial neural networks (ANNs) are computing software inspired by biological neurons' simplified structure as depicted in figure(3.4a). ANNs' communication mechanism is loosely based on the structure of the so called biological neural networks. An artificial neural network consists of a network of computing units that are called nodes or neurons. Every neuron has a small computing unit that receives an input, performs some simple computations according to its predefined function and produces an output. Neurons are connected to other neurons through connections or edges. Each connection carries the output signal of one neuron to another neuron, the output signal is multiplied by the weight of the connection which increase of reduces the effect of the output [22]. In an artificial neuron shown in figure(2.7b), inputs are based on dendrites of biological neuron, computing unit shown in a circle is based on nucleus and output is based on axon.

Use of artificial neural networks to learn from large amounts of data dates back to 1950s, the first neural network was in the form of the a simple computing unit called Perceptron [60]. However, modern neural networks entail deep and complex architecture that require large computation power to produce an acceptable output. Recently, research in this area has boomed due to an increase in available computation power because it is now fairly easy to program and train a neural network. Very deep networks have been developed that are able to learn complex underlying systems such as stochastic systems. Modern neural networks can be used as a very powerful system identification and ap-



(a) Simplified biological neuron



(b) Artificial neuron structure

proximation algorithm. Our proposed methodology is inspired by the natural connection between neural networks and dynamical systems. From a stochastic system's perspective; the forward passes in neural network training can be seen as state evolution of the dynamical system [43].

2.3.1 Related Concepts

Study of artificial neural networks is a very wide field of study that constitute many subfields such as deep learning etc. Covering all the fundamental concepts is beyond the scope of our work. Only the concepts and ideas that we will use to explain our methodology and results will be briefly introduced in the subsequent part of this section. For a deeper and thorough understanding of all the related concepts we suggest following resources [69, 25, 5, 53, 13].

In this section we will be bringing up a few questions that the reader should be able to answer before diving into the rest of this work. We will be using these terms in the later chapters when we explain how we setup and train our neural network.

- How neural networks work as a function mapping a vector of inputs x to a vector of outputs y
- What are the non-linear activation functions, why activation functions are needed to be applied to the output of a neuron to allow it to learn complex patterns in the data
- What are the training and validation data sets
- What is a loss function, how training and validation loss is calculated
- How neural networks are trained using an optimizer and its types, what is back propagation
- What is batch training, epochs, iterations etc
- How is hyperparameter tuning done, which techniques are used
- What are network over-fitting and under-fitting
- Which methods and techniques are used for neural network weight initialization

2.3.2 Existing System Identification Algorithms

Identification of dynamical systems from observed data has been studied extensively and several methods and algorithms have been introduced [15, 16, 42, 51, 57, 58]. Runge-Kutta ResNet architecture was devised for deterministic ODE identification and extended to implicit ResNets. Some examples of this are DeepONets [47], ResNets [30], Neural ODEs [18, 59], neural PDEs [24]. In order to identify a dynamical system, if SDE is the desired outcome of this identification, different methods are required. Gaussian processes have been used for identifying underlying SDEs from observed data without gradient matching [76]. Euler-Maruyama scheme has also been used to calculate target likelihood in this approach however it is not explicitly mentioned by the authors. Mani-fold constrained Gaussian processes are also used to identify ODEs from noisy and sparse observations data [75]. Non-Gaussian statistics are also used for generative stochastic modelling of strongly non-linear flows [7]. Density and ensemble approximation methods are also used to study stochastic systems with SDEs using discrete particle ensemble observations [74]. Fokker-Plank operator is another method used to study stochastic dynamical systems, it uses a non-parametric modeling approach for forecasting stochastic dynamical systems on low-dimensional manifolds [11].

Learning SDEs from observed data is a natural next step in this direction. The local drift and diffusivity estimation can now be done using a (more or less) surrogate model such as a neural network. In our work we will be needing two neural networks: f_θ to approximate drift of an SDE and σ_θ to approximate the diffusivity of an SDE. Drift net f_θ parameterizes a differential equation to fit the predictive function, while the diffusion net parameterizes the Wiener process and encourages high diffusion for data. The drift net controls the system to achieve good predictive accuracy, while the diffusion net characterizes model uncertainty in a stochastic environment [43].

2.3.3 Training Data

In order to train a neural network to make predictions, we need a suitable dataset that will train the neural network well and will enable it to make accurate predictions. What we need to train our particular neural network is a discrete spatio-temporal data set. In our training data we a set of N snapshots $D = \{(x_0^{(k)}, x_1^{(k)}, h^{(k)})\}_{k=1}^N$, where $x_0^{(k)}$ are the data points scattered in the state space of our SDE(2.6) and the value of $x_1^{(k)}$ is obtained by evolution of the SDE(2.6) starting at $x_0^{(k)}$ after a small time step $h^{(k)} > 0$. The snapshots are sampled from a distribution $x_0 \sim p_0$ and transition densities $x_1 \sim p_1(.|x_0, h)$ are associated with SDE(2.6) for a given time-step $h > 0$, which is taken from some distribution p_h. The joint data-generating distribution is given by $x_1 \sim p_1(.|x_0, h)p_o(x_0)p_h(h)$.

The data can also be recorded along a long trajectory $\{x_{t_i}\}$ of SDE(2.6) with sample frequency $h_i > 0$, such that $t_{i+1} = t_i + h_i$. We assume that the snapshots in D are sampled sufficiently close to each other in region of interest. Also the step size $h^{(k)}$ is defined per snapshot, which means that it can have different values in different snapshots.

2.3.4 Loss Function

The loss function is at the heart of any neural network. A neural network has to predict a set of outputs, given a set of inputs and a loss functions or measures the error of neural network in that prediction. Based on the amount of error in prediction, weights of the neural network are updated to minimize its loss. This process of updating the weights iteratively using backpropagation, is called neural network optimization. If a neural network is learning well, the loss converges after training as shown in figure(2.8). Different types of loss functions are used in different types of neural networks and they can also be inspired by different numerical methods. Loss functions derived from numerical integration schemes were devised in the 90s [24, 59]. Recently, they have been increasingly used in deep learning architectures such as ResNets [30], Hamiltonian networks [12, 27, 77] and Poisson networks [35]. In order to allow backpropagation through a long series of SDE iterations, scalable gradient approaches are used [45]. This type of scalability is not an immediate training issue if only a single time step is available for each data point, as it is in our work.

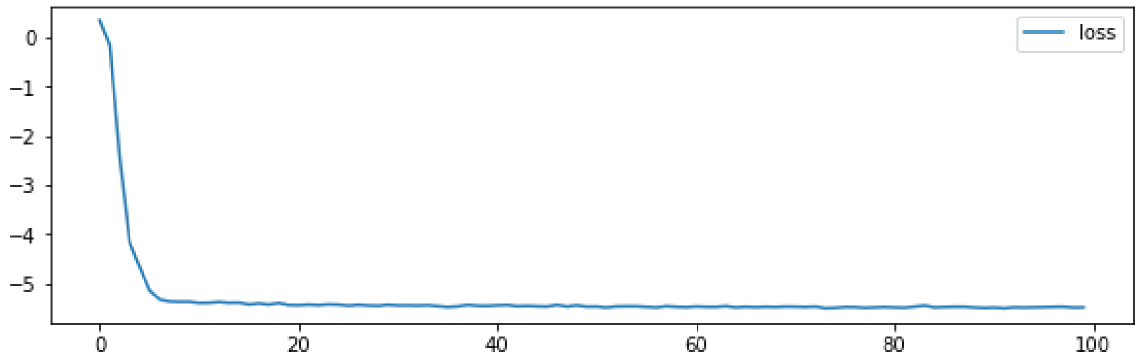


Figure 2.8: Loss function curve that converges during training

2.3.5 Euler-Maruyama inspired Loss Function

Artificial neural networks have done very well in a wide variety of tasks, such as machine translation [8], image classification [44] and reinforcement learning [52]. However, ANNs are still not good at modelling stochastic systems where uncertainty is the basic element of the system. In order to learn the dynamics of a stochastic system, we need a loss function that is derived from a stochastic integration scheme. Loss function derived from one integration scheme will be different than the one derived from another integrator.

Stochastic integrators can vary from each other in many different respects; for example they can have higher or lower rate of convergence, offer strong or weak convergence and can even be based on different type of stochastic calculus such as Itô or Stratonovich. We use a loss function that is inspired by and embodies an Itô-calculus based integrator Euler-Maruyama scheme in our work. EM provides us the state of the system after a time step in equation(2.7). The convergence of equation(2.7) has been studied extensively when $h \to 0$ [54]. This idea enables us to construct a loss function that optimizes both drift f_θ and diffusivity σ_θ neural networks at the same time. State of the stochastic system after a small time step h, denoted by x_1 is from a normal multivariate distribution as shown in equation(2.8).

As we discussed in our training data set D we have triplets $(x_0^{(k)}, x_1^{(k)}, h^{(k)})$ but not the drift and diffusivity of our system of interest. In order to approximate f and σ, we look at the probability density function p_θ of the normal distribution of x_1 defined in equation(2.8). This type of likelihood estimation with normal distribution has been used in developing several other generative neural networks [26, 39, 45, 67, 75, 76]. Given the neural networks f_θ and σ_θ, we ask that the log-likelihood of the data D under the assumption in (2.8) is high. Probability density function of a variable x is given in equation(2.9):

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{(x-\mu}{\sigma})^2} \tag{2.9}$$

We plug in our variables and to simplify it we take a logarithm of the equation(2.9). We also remove the constant terms because they do not influence the minimization [54].

$$\theta := arg \max_{\hat{\theta}} \mathbb{E}[log p_{\hat{\theta}}(x_1|x_0,h)] \approx arg \max_{\hat{\theta}} \frac{1}{N}\sum_{k=1}^{N} log p_{\hat{\theta}}(x_0^{(k)}|x_1^{(k)},h^{(k)}) \tag{2.10}$$

Now we can work out our loss function that will be minimized to find optimal weights θ. The logarithm of the well-known probability density function of the normal distribution in previous step, together with the mean and variance from (2.9), results in minimization of loss during training:

$$\mathcal{L}(\theta|x_0,x_1,h) := \frac{(x_1 - x_0 - hf_\theta(x_0))^2}{h\sigma_\theta(x_0)^2} + log|h\sigma_\theta(x_0)^2| + log(2\pi) \tag{2.11}$$

We can think of p_θ as being induced by an SDE(2.6) with modified drift f and diffusion σ. Such kind of modified equations have been derived for a variety of numerical methods [78].

2.4 Learning Effective Stochastic Differential Equations from Microscopic Simulations

This section will provide a brief overview of a theory presented by Dr. Dietrich et al. in ”Learning effective stochastic differential equations from microscopic simulations: combining stochastic numerics and deep learning” [20], that forms the basis of our work during this project. This work puts forth a machine learning based approach that enables stochastic system identification from scattered snapshot data. Fine-grained particle- or agent-based simulations are taken as an input and values of coarse observables are calculated. These coarse observables are governed by underlying effective stochastic differential equations (SDEs), that can act as surrogate models.

In order to approximate the underlying SDEs, the drift and diffusivity functions of SDEs has to be approximated. The drift and diffusivity of SDEs are approximated by using an identification algorithm in the form of a neural network developed by Dr. Dietrich. Loss function used in the training of this neural network is inspired by stochastic integration schemes such as Euler-Maruyama scheme and Milstein method. Backward error analysis of these methods fits into the neural network loss function perfectly. The neural network and loss function are implemented using TensorFlow. Training data used to train this neural network does not require to have long trajectories, it works on scattered snapshot data as well.

Paper also illustrates experiments that are done using this neural network, system of interest in these experiments is spread of a viral disease. First of all coarse-grained particle / lattice simulations of an SIR epidemiological model are obtained. They are used as an input to the implemented neural network to approximate the drift and diffusivity of a 2D mean field SDE. In order to obtain a ground truth, a kinetic Monte-Carlo lattice simulation of the same SIR model are obtained using Gillespie algorithm. Ground truth model is compared with the surrogate approximated SDE model to measure the performance of the neural network.

What has not been done in the paper, and what is the content of this thesis (see next chapter), is to study coarse-graining of more complex scenarios with agent-based simulations. In Vadere, very complex scenarios can be constructed, analyzed and coarse-grained, while the paper only discussed a box scenario with a cellular automaton-type microscopic model.

3 Main Part

This chapter outlines the details of the work we have done during this thesis project. We start with describing in detail the task of system identification and then step by step explain the process that we go through to accomplish this task. Our task of system identification from data is a regression task that can be sufficiently modelled using a neural network architecture.

Before we dive deep into the neural network that we will be using during this project, we define what type of task we have at hand its details and methodology we will be using to solve this problem in section 3.1. In section 3.2, we sketch out the details of the neural network that we will be using for our defined task. Section 3.3, introduces the dataset we will be using for training out neural network, how we obtain and process it before we use it as an input. In order to test how our neural network is doing we compare networks results with a test dataset; we describe generation and preparation of our test dataset in 3.5 section. In section 3.6, we outline experiments that we perform using our neural network, their setup and results. Loss function that is used by our neural network is a new type of loss function, in section 3.7, we compare with other evaluation metrics.

3.1 Task Description

We have already established in the previous chapters that learning stochastic differential equations from microscopic simulation data is an important problem. Stochastic differential equations can be used for modelling purposes in many different disciplines of natural sciences. Hence, there can be many different possible applications of our work in some totally different areas. However, during this project we have been inspired and motivated by a certain stochastic system. We call this our system of interest, it is the spread of a contagious viral disease in a closed population.

3.1.1 Motivation

We discussed some motivations of our work in section 1.2, in this chapter we will focus in detail on a motivation we have on the application side. Improving quality of human life through possible disease elimination or at least disease spread prevention has always been a huge area of research. When a contagious viral disease starts spreading very fast, its called an epidemic. Epidemics affect many different areas of our lives. From economies to medical centers, all aspects of human life get affected by an epidemic. Hospitals start receiving a high number of patients and eventually start running out of capacity. People can not move freely which affects businesses, eventually economy of regions and countries. In order to stop the spread of a virus, many different measures are taken such as social distancing, lock-downs, travel bans, special hygiene care etc. However, in order to understand the spread of a virus the most important aspect is to understand how it is spreading.

3.1.2 Problem Description

Spread of a contagious viral disease turns into an epidemic almost every year in some part of the world. Epidemics affect health of a large portion of the population, it becomes very difficult to provide medical care to the sick and weak, hospitals and medical centers get chocked up due to a high numbers of patients, livelihood of masses get affected and economy of the country takes a hit. If an epidemic spreads in a wider geographical area, it becomes a pandemic; such as Spanish flu pandemic of the last century or the COVID-19 pandemic that we are seeing today. Pandemics not only affect local economies and health systems, but a global trade sedation is inevitable; it affects global economy, affects mental and physical health of masses and has a negative impact on almost every area of our lives. According to a research done by US department of disease prevention, 20 million people had died globally in influenza pandemic of 1918 [50]. In today's world despite the availability of advanced medical technologies, according to world health organization, 4.5 million people have died in past 2 years due to COVID-19 pandemic.

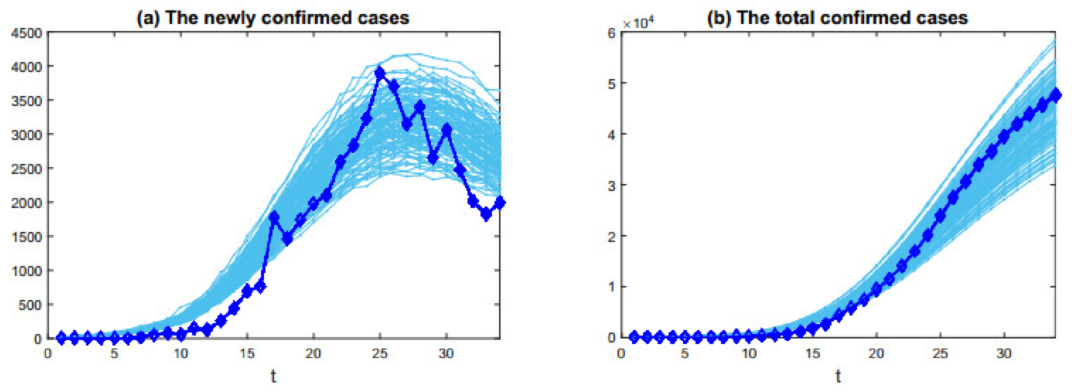


Figure 3.1: The data of newly reported confirmed cases and total number of confirmed cases of COVID-19 disease from January 11, 2020 to February 13, 2020. Stochastic fit was performed 100 times, the data and the fitted curves are represented by the deep blue and light blue curves, respectively [31]

Spread of a virus such as COVID-19 can be modelled using a stochastic process [31], as it is evident from the figure(3.1) where actual data is represented with dark blue graph and predicted stochastic process is shown with light blue graph. This graph shows the number of new cases where total population is not really considered. However, the percentage of total closed population that lies in susceptible or infected group is predicted later. If we look at figure(3.1b), it can be seen that the virus outbreak started with small number of cases in the beginning and then spread to a larger population. The overall graph looks smooth as if its an ordinary differential equation graph. However, if we zoom in and see a smaller portion of the graph; we can see that the virus spread graph is not very smooth and it ”looks stochastic” as it is evident from figure(3.1a).

To contain an epidemic or a pandemic; several measures are taken by the governments and health recommendations are issued by the WHO. In order to plan effective measures, issue useful recommendations or implement successful measures; institutes need deeper insights into this process which is a stochastic dynamical process. These insights come from adequate modelling of the spread of virus using statistical and mathematical tools. Statistical tools alone are not enough for effective modelling because often data gathered is limited and virus spread process is only partially observed.

If we think of humans who are carrying, spreading and getting infected by the virus as agents, we can employ a very powerful modelling tool used by the researchers; agent based modelling and simulation software. These software and frameworks produce simulations with great amount of detail about every agent, its movements over time, its interactions with other agents and impact it has on agent’s behaviour and future movements. Some of the information is also available implicitly and can be calculated using observed information. For instance if we have a group of agents exiting a room in the simulation, we can calculate their evacuation time. On top of all this information, we can incorporate an epidemic model to simulate spread of the virus.

Even though agent based simulation software provide a pretty good approximation of the reality, they have many limitations that we have discussed in section 2.1.2. In order to adequately model a stochastic dynamical system such as spread of a contagious viral disease without having to simulate time consuming and irrelevant details present in agent based simulations; we need a stochastic system identification algorithm that can learn from coarse observables that we calculate from microscopic simulation data.

3.1.3 Methodology

So far we have understood the problem that we have at hand and explored biggest motivation behind our research. Now we will dive deep into the strategy and methodology used to achieve the objectives defined in section 1.3.

Setup Stochastic system Identification algorithm

First of all we will explore and setup the stochastic system identification algorithm that we will be using during this project. This algorithm is in the form of a neural network developed by Dr. Felix Dietrich at TU Munich. In order to use this algorithm for system identification, we need a data set that contains scattered snapshots of our system of interest after different time steps.

Input Data Generation

In the second step we will use an open source agent based simulation framework Vadere to generate fine-grained microscopic dynamics of a closed population in a closed environment. We use Vadere to simulate the spread of a contagious viral disease in the population.

Data Pre-processing

On one hand we have from the previous step in our methodology is microscopic dynamics of the stochastic system; on the other hand we need a data set with scattered snapshot data. We take our original dataset through a number of steps to achieve the dataset that we need, details of this are discussed in 3.3 section.

Neural Network Training

Input dataset obtained in the previous step is used to train the neural network that we had setup. Network training goes on until we get acceptable values of training and validation loss. We go through multiple iterations of hyper-parameter tuning while looking for acceptable loss values.

Test Data Generation

Test data is also generated using Vadere and coarse observable values are calculated after different time steps. The process followed for text data generation is similar to the training data generation, specific details of this are discussed in section 3.5.

Computational Experiments

All the above steps are applied in context of a specific experiment that we setup. We reproduce one experiment by Dr. Felix Dietrich and setup two more experiments and record their results. Finally we record and report our findings during these experiments.

3.1.4 Implementation Details

In the later parts of this chapter we will be discussing in great detail all the implementation details of different parts of our solution. This section is aimed at providing a high level overview of the implementation phase of this project, software tools that were used and the results that they provided.

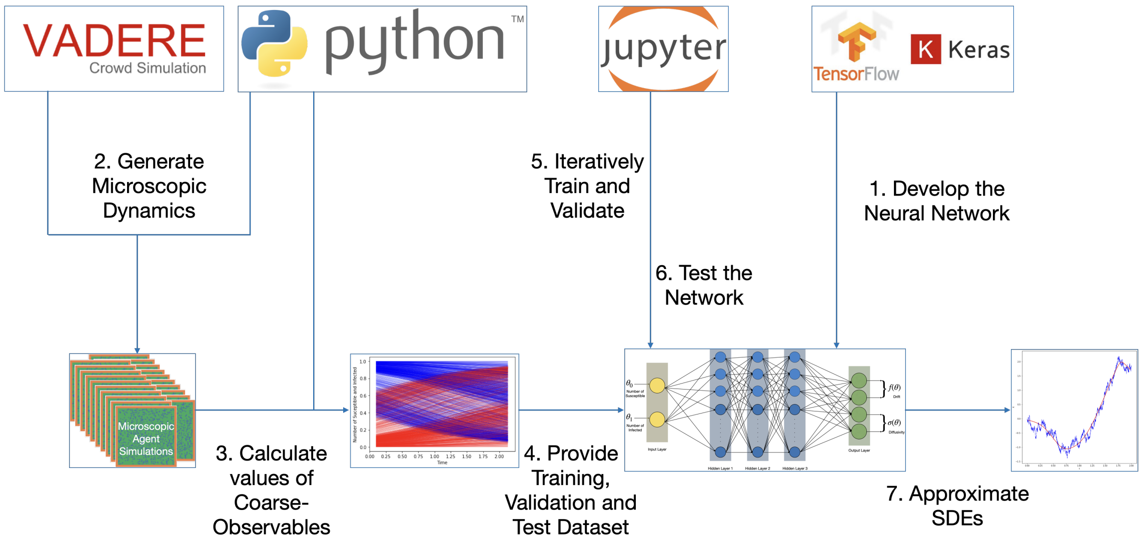

VADERE
Crowd Simulation
python
jupyter
TensorFlow
Keras
2. Generate Microscopic Dynamics
5. Iteratively Train and Validate
1. Develop the Neural Network
6. Test the Network
Microscopic Agent Simulations
3. Calculate values of Coarse-Observables
4. Provide Training, Validation and Test Dataset
7. Approximate SDEs

Figure 3.2: High level implementation details of the project

We describe in figure(3.2) how different parts of our solution fit together and which software tools we used to develop this solution. The neural network and its loss function were defined in `TensorFlow V2.4.1 [3] (Apache 2.0 license)` and `Keras` library by Dr. Felix Dietrich at TU Munich [20]. Then we generated the microscopic simulation data by an automation software written in `Python` generates simulations and their output `csv` files by invoking the `Vadere` console. We use different scripts that use this technique to generate two different data sets. We use another script to calculate values to of coarse-observables of the system in both data sets. We take the first dataset and split it into training and validation data sets. We use a `Jupyter Notebook` with a `Python 3.0` kernel to iteratively train and validate our neural network until we have the desired value of loss function. Then we use the same `Notebook` to test the neural network and its performance. Finally we generate the approximated SDE paths using our `Notebook`.

One important thing to note here is that we did not download the compiled version of Vadere, because the compiled version that is available on Vadere website does not contain the specific SEIR model that we want to use. We downloaded the source code from Vadere git repository, incorporated the SEIR model and then compiled the whole project.

3.2 Neural Network Architecture

We have already introduced that the systems identification algorithm we will be using is an artificial neural network with a loss function that minimizes drift and diffusivity nets at the same time. For the sake of simplicity, we will be referring to the drift neural network and diffusivity neural network as an SDE neural network (SDE-NN) in this section. We will be exploring the implementation details of SDE-NN in this section. Network topology is visualized in figure(3.3) which is a simplified view of the network; input and hidden layers are used twice, separately for drift and diffusivity. Also, here the diffusivity is diagonal, hence the same number of output neurons are visualized as for the drift. In case we use a different type of diffusivity such as triangular or SPD, number of neurons in output layer will increase.

A neural network consists of a specific number of neurons, SDE-NN that we setup for our experiments contains 156 neurons. Each neuron is connected with other neurons through edges and the weight of one edge is called one network parameter. In SDE-NN we have $2,854$ parameters; a set of all the network parameters together is denoted by θ. Neurons are organized within layers, every neuron belongs to a specific layer depending on its function. The SDE-NN has 5 layers; an input layer, three hidden layers and an output layer which are implemented using `keras` library. `keras` is an open source framework that contains many building blocks of a neural network.

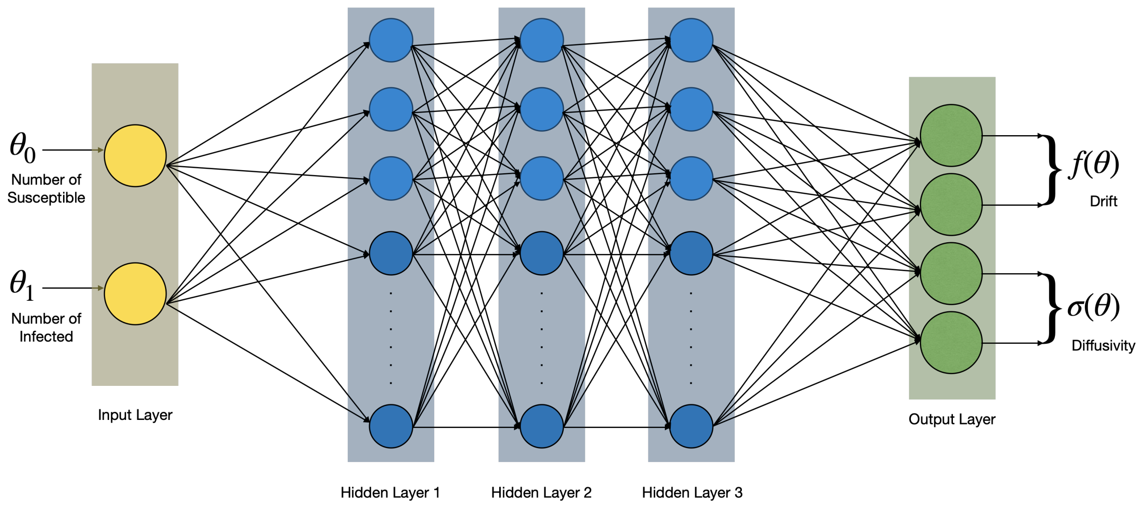


Figure 3.3: Architecture of the SDE neural network (SDE-NN) used for stochastic system identification, here the diffusivity is diagonal, hence the same number of output neurons are visualized as for the drift

3.2.1 Drift Net and Diffusivity Net

On one hand we can think of the drift net as the one that tries to fit the predictive underlying system by using a parameterized SDE. On the other hand we can think of the diffusivity net that tries to predict how much the actual dynamics will deviate from the function predicted by drift net. The drift net enables the SDE-NN to achieve good predictive accuracy, while the diffusion net characterizes the uncertainty in a stochastic environment. Theoretically, when both drift and diffusivity of the stochastic system are low:

neural network should make confident predictions with low variance. When drift is high and diffusivity is low: the system can make confident predictions but with large variance. When diffusivity is high and drift is low: The predictions are scattered in a highly diffused fashion over the parameter space [43].

3.2.2 Input Layer

The input layer is the very beginning of the flow for a neural network, it is the entry point for training data into the neural network. It consists of artificial input neurons that are not performing any computations on the data, they only just bring the data into the network for further processing by subsequent layers of the network. We implement our input layer using `tf.keras.layers.Input` function.

The input layer of SDE-NN contains only two neurons; number of susceptible population denoted by θ_0 and number of infected population denoted by θ_1. These are not to be confused with the network weight parameters that are denoted by θ. The input layer can contain more input neurons if we have to study a bigger system; for example if we are to study an SIR system we can have three inputs for respective S, I and R values. However, during this thesis we study a simplified version of SIR system that contains only S and I values.

3.2.3 Hidden Layers

All the layers in a neural network except for input and output layers are called hidden layers because the model developers have no control over these layers. Only number of layers is defined in the beginning when neural network architecture is being setup and when the network starts learning everything within these layers is controlled by the network itself. We use TensorFlow `tf.keras.layers.Dense` function to implement all the hidden layers as described in table(3.1). All the hidden layers that contain `GP.mean.hidden` in their name (first column) can be considered a part of drift net and all the layers with `GP.std.hidden` in their name are a part of diffusion net. First hidden layer in drift and diffusion net have 25 neurons and 75 network parameters each. Second and third hidden layers in both networks have 25 neurons and 650 parameters.

3.2.4 Output Layer

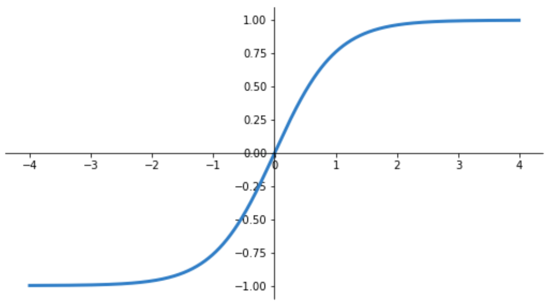
In the output layer we have four neurons, two of them provide us drift function and the other two diffusivity. Output layer is also implemented using `tf.keras.layers.Dense` function. All the layers of neural network are created and passed to `tf.keras.Model` function to build the model.

Network Layer Name (Type)	Output Shape	No. of Parameters	Connected to
GP_inputs (Input Layer)	(None, 2)	0	
GP_mean_hidden_0 (Dense)	(None, 25)	75	GP_inputs
GP_std_hidden_0 (Dense)	(None, 25)	75	GP_inputs
GP_mean_hidden_1 (Dense)	(None, 25)	650	GP_mean_hidden_0
GP_std_hidden_1 (Dense)	(None, 25)	650	GP_std_hidden_0
GP_mean_hidden_2 (Dense)	(None, 25)	650	GP_mean_hidden_1
GP_std_hidden_2 (Dense)	(None, 25)	650	GP_std_hidden_1
GP_output_mean (Dense)	(None, 2)	52	GP_mean_hidden_2
GP_output_std (Dense)	(None, 2)	52	GP_std_hidden_2

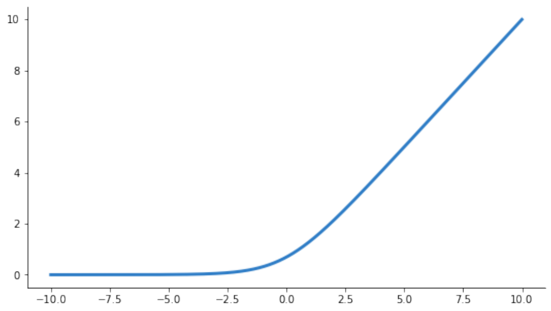
Table 3.1: Neural Network Layers

3.2.5 Activation Function

Each neuron in the hidden and output layer contains an activation function. Activation function `tanh` we are using in hidden layers is non-linear that allows the network to learn non-linearities within the data.



(a) `tanh` activation function



(b) `softplus` activation function

`tanh` is a variation of sigmoid activation function but works better than the sigmoid. Sigmoid is one of the most commonly used activation function in the neural network community. For an given input, a `tanh` function returns a value between -1 and 1 as shown in figure(3.4a). This works very well with most models as it gives a probabilistic measure as an output which is calculated using equation(3.1).

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{3.1}$$

Output layer of the diffusivity net uses the activation `softplus+STD_MIN_VALUE`; where `STD_MIN_VALUE` is the minimal number that the diffusivity models can have which is added for numerical stability, and `softplus` is an activation function shown in figure(3.4b). `softplus` is used to calculate diffusivity because it can not be a negative value.

The equation(3.2) for calculating `softplus` is as follows:

$$f(x) = log(1 + e^x) \tag{3.2}$$

3.2.6 Loss Function

Loss function is a mathematical way of measuring how wrong the predictions of the model are. Loss function is a special kind of error function that takes as an input the ground truth values and the values predicted by the model and return the amount of error that there is. The parameters of the above described neural network are learned by minimizing the equation (2.11).

3.2.7 Optimizer and its Learning Rate

In order for a neural network to make good predictions, it needs to optimize its weights during neural network training. During optimization process, multiple algorithms are used such as backpropagation and gradient descent. Back propagation is an iterative process; each iteration consists of a forward pass and a backward pass. In a forward pass, input propagates through all the layers of the neural network such that it is multiplied by the weight of one layer and then passed on to the next until it reaches the output nodes. Back propagation is a more complicated process based on gradient descent with the ultimate goal of loss minimization. The general idea is to calculate the gradient of the loss function with respect of the weights. In each iteration, the model weights are updated using the gradient of the loss function which guides the loss function towards the minimum. Another important concept in this process is the learning rate which tells the algorithm how fast or slow it should move towards the minimum point. This process of updates is repeated several times until the model accuracy reaches an acceptable value.

Optimizer that we use is Adamax with a standard `learning rate` 0.001; Adamax is defined using `tf.keras.optimizers.Adamax` function. Adamax is based on the infinity norm variation of the very popular Adam optimizer, which uses gradient descent with the combination of the Momentum and RMSProp algorithms. These algorithms compute a weighted average of the squares of the gradient, in order to identify the lowest overall loss of the network.

3.3 Input Data

Input data is a very important element for making a neural network good at its predictions, quality of the data is directly proportional to the accuracy of the neural network's predictions. We obtain our input data by following these steps below, details of each step will be discussed in the later parts of this section:

1. Define an agent based scenario using Vadere GUI or a JSON file; we use Vadere GUI

2. Run Vadere from console to simulate the scenario, collect the simulation output data and SIR values at the end of the simulation which we will consider to be a day
3. Simulate the scenario for next days (specified per scenario) starting at the SIR values from the previous step
4. Repeat step 2 and 3 until we have the amount of data points that we need
5. Calculate coarse observable values of our system of interest from the collected microscopic data
6. Convert the macroscopic data to time snapshot pairs (x_t, x_{t+h})
7. Time reduction factor of 10 is used to scale the Vadere time step and make it small enough for the network to handle effectively
8. Normalize the data such that all the system SIR values lie between $[0, 1]$ interval
9. Select only Susceptible and Infected values as θ_0 and θ_1, since the neural network we have setup takes two input values
10. Make the input dataset sparse

We generate two input data sets for two different experiments we will be performing. Specifications of these data sets are described below in table(3.2), along with a cubic 1D dataset that we have taken from Dr. felix's work [20]:

Input Dataset	Time Step (h)	Number of Data points
cubic, 1D	0.01	10,000
Box	0.01	1,000
Bottleneck	0.01	1,000

Table 3.2: Input Data Specifications

3.3.1 Input Microscopic Data

Microscopic data generation for most of the experiments was quick and took about 3 hours. At one point during our experiments; we generated $10,000$ training data points for an experiment which took about 24 hours. However, upon comparison and analysis we learned that it was not really necessary to increase the size of data set as it did not improve the accuracy very much. In order to generate data for our neural network, the first step that we need to take is to create an agent based simulation using Vadere. Every simulation in Vadere is based on a scenario. A scenario defines every aspect of the simulation: the environment where the scenario takes place, the agents and their attributes, locomotion model, additional models such as SIR model, data output specifications etc. In order to define the scenario, we use Vadere GUI. Vadere GUI consists of multiple tab that are used to define different aspects of the system. Every scenario is basically a JSON file, which can either be written by hand or created using the GUI.

Simulation

We use simulation tab in Vadere and sketch out the details of the simulation. In order to set the length of a single simulation, we set the value of `"finishTime"` attributes. A simulation always starts at time 0 and ends at `"finishTime"`. `"simTimeStepLength"` is used to define the amount of time after which Vadere should produce new state of the system. Value of attribute `"visualizationEnabled"` is set to `false` which makes the process of our data generation faster. `"useFixedSeed"` is also an important attribute, we set its value to `true` which makes sure that the seed used to create the simulation is controlled by `"fixedSeed"` attribute and not randomly generated by Vadere. `"fixedSeed"` attribute is later controlled by our code, use of fixed seed is done to ensure

Model

Model tab in Vadere is used to define which locomotion model we will be using. We use the ″Load template″ option at the top of the model tab to select optimal steps model[2.1.4] using option `"org.vadere.simulator.models.osm.OptimalStepsModel"` with default values

SEIRG Attribute: We insert one more attribute by using ″Add Attribute″ option and selecting `"org.vadere.state.attributes.models.AttributesSEIRG"`. SEIRG is a model attribute set that contains many different attributes within. Since we only want to use S and I from the model; values of `"exposedRate"`, `"recoveredRate"`, `"numberOfExposed"`, `"numberOfRecovered"` and `"recoveryLikelihood"` are set to 0. Value of `"infectedRate"` is set to 0.001, `"infectionRadius"` is set to 1.5, `"infectionLikelihood"` is set to 0.01 and rest of the default values are used.

Topography

We use Topography creator tab in Vadere and sketch out the details of the scenario.

Floor Field: First thing we create is the floor field upon which the agents will be moving. The boundaries of this floor field defined by its attributes `"height"` and `"width"`, determine how much roam agents have to move about. We also create some obstacles in some scenarios and they define the areas where the agents are not able to go.

Source: Then we create a source which actually creates the agents and spawns them onto the floor field. Source has multiple attributes that we will be specifying: `"spawnNumber"` is used to defined number of agents, `"spawnAtRandomPositions"` is set to `true` so that an agent is spawned at a new position in a new simulation, `"useFreeSpaceOnly"` is set to `false` so that agents are created close together and the spread of virus happens, `"targetIds"` is set according to the ID of the target we will be creating later. For rest of the attributes we use default values created by Vadere.

Target: We define a target that is the target for all the agents that are created by a source that points to this target, which makes `"id"` as the most important attribute that we have to set. There is only one more attribute that we define `"absorbing"`, this attribute's value is set to `true` or `false` according to the requirements of the scenario.

Data output

Vadere can produce data in different formats in a csv file, it depends on the user what output they really need. Output specifications can be defined in "Data output" tab, by using data processors. Every processor contains different type of data, we use five different processors most important one is `"EventtimePedestrianIdOutputFile"`. It captures group ID of every agent along with agent ID at every time step, which results in a very long csv file which has all the fine-grained microscopic details of the system. All the information obtained in this step is not directly relevant for us; we do not need to know exactly which agent was in which specific group at a certain time. What we do need is the total number of agents in each group at every certain point in time.

3.3.2 Calculate Values of Coarse Observables

First of all we use the output csv file from the previous step to calculate number of susceptible and infected population at every Vadere time step. Vadere time step has a corrosponds to the real time in the real world, however, we do not need this corrospondence. The time step in our input data set can be arbitrary; we can choose to zoom out and instead of recording group counts after every Vadere time step we can record it after each simulation.

3.3.3 Normalization and Processing of Data

In order to normalize our dataset we divide each column by the total number of agents in the system which ensures that all the values in individual groups lie between $[0, 1]$, now we can call these values θ_0 and θ_1. For instance, if total number of agents in a scenario is 100 and there are 85 susceptible agents and 15 infected agents; value of θ_0 will be $\frac{85}{100} = 0.85$ and value of θ_1 will be $\frac{15}{100} = 0.15$. Another part of input data pre-processing is that we skip some of the simulated time snapshots, so that the individual points are further apart in time.

3.3.4 Training and Validation Data Split

Training dataset is used to train the network on a specific set of data. In order to make sure that the network has not over-fitted i.e. it has not simply memorized the data but has learned the underlying patterns; we use validation set. Validation dataset is separated from the training dataset and is kept aside and network never sees the data before validation phase. Once the network has completed training on the training dataset, we evaluate its performance on the validation dataset. If the network performs reasonably well on the validation dataset as well, it is considered well trained. However, if the network does not perform well a number of steps can be taken, we might reconfigure different hyperparameters of the network or normalize or process input data in a different way or totally change the dataset. This list of steps is not exhaustive, many different types of measures can be taken to improve model performance. After making these changes we start second iteration of training; retrain the network and evaluate its performance again on the validation dataset until the network has reached desired performance. In all our experiments 90% of the training data stays in the training dataset, while arbitrarily chose 10% is set aside in the validation dataset.

3.4 Training the Neural Network

Training time, setup and hyperparameters are usually positively related to a network architectures potential. Without being over-trained, a deeper network has the stronger expressive ability but also more parameters to learn thus causing more time to train. Without being under-trained, with enough amount of data, network's learning capability also increases. In this section we will be exploring the setup that we found to be perfect for our neural network and yielded the best results during our experiments.

3.4.1 Hyperparameter Tuning

Hyperparameters are the non-learn-able parameters of the network that are set before training the neural network; and the accuracy the network depends heavily on these. Depending on the size of our dataset and computing power available to us; we need to choose a number of hyperparameters that are best suited for our network. For instance choosing a good learning rate is a very important; if learning rate is too low network will be very slow and can be stuck in a local minima, if it is too high the network might miss the minima and might not converge. If the network is not performing well, developers can tweak the hyperparameters and train the network again. We went through a number of iterations to reach the hyperparameters that lets our network yield high accuracy.

3.4.2 Training Setup of Neural Network

In order to specify what happens during training and optimization of the neural network; first of all we implement and plug in our custom Euler-Maruyama scheme based loss function. Then, we need to customize what happens in the the `tf.keras.Model.fit()` function by following the steps below:

- Create a new class that sub-classes the `keras.models.Model`
- Create our own `train_model` function
- Pass Numpy arrays, by calling `fit(x,y, ...)`, this enables data to be passed as tuples of form `(x,y)` which are time snapshots of our system

All the neural networks were trained on a single laptop computer with an Apple M1 8-Core Processor with a built-in GPU and a 16GB RAM. To train our neural network we use Stochastic Gradient descent based optimizer `Adamax` with an initial `learning_rate` of $1e^{-2}$ that decays at the rate of 0.9 and a momentum of 0.999 and has a small constant for numerical stability $\epsilon = 10^7$. We compile it using Euler-Maruyama loss function.

We specify a batch size of 32, allowing 32 data samples to be processed before loss computation and weight updates in our network. We specify the number of epochs to be 100 to allow multiple passes through the entire training dataset.

3.4.3 Development Life Cycle

Researchers would identify a problem where stochastic differential equations can serve as a surrogate model. They would, then, see the tools that they have available to simulate microscopic dynamics of the system. Microscopic data will be used to calculate the coarse-observable values of the system. They would choose a suitable algorithm to model the learning problem, select a proxy dataset to train and tune the hyperparameters of the model. Once the model hyperparameters have been carefully tuned, the engineers can test out the model using a simulated test dataset. Once the model has reached acceptable accuracy, it can be applied to model the real world problems.

3.5 Test Data

Test data is the dataset that we use to evaluate the final performance of our network after the development phase. In order to make sure that it provides an unbiased performance evaluation, it is only to be used after the network completes training and is final and stable. This can not be guaranteed using the validation dataset, although it is not used during training, we have still reconfigured our neural network repeatedly to make its performance good on the validation dataset. Hence, it is absolutely important to not touch the test dataset until the very end of the network development workflow.

Our test data is generated using Vadere and coarse-graining technique similar to the training dataset, using following steps:

1. Use the agent based simulation scenario defined for the training dataset
2. Run Vadere from console to simulate the scenario for a day and next number of days (depending on the scenario), collect the simulation output data
3. Repeat step 2 until we have desired number of test data points we need
4. Calculate coarse observable values of our system of interest from the collected microscopic data
5. Convert the macroscopic data to time snapshot pairs (x_t, x_{t+h})
6. Time reduction factor of 10 is used to scale the Vadere time step
7. Normalize the data such that all the system SIR values lie between $[0, 1]$ interval
8. Select only Susceptible and Infected values as θ_0 and θ_1, since the neural network we have setup predicts only these two values

Further specific details (if any) of data sets in each experiment will be discussed in their respective sections.

3.6 Computational Experiments

First of all we reproduce one of the toy examples presented in the paper [20], to illustrate our neural network identification technique. After that we design two experiments inspired from real world scenarios, in order to show an application of learning underlying SDEs from coarse observables.

3.6.1 1D Cubic SDE

In this first experiment we reproduce the results from a 1D cubic experiment in the paper [20], with drift f and diffusivity σ defined through $f(x_t) = -2x_t^3 - 4x_t + 1.5, \sigma(x_t) = 0.05x_t + 0.5$. We trained the neural network with Euler-Maruyama loss function; training and validation loss curves that we got are shown in figure(3.5):

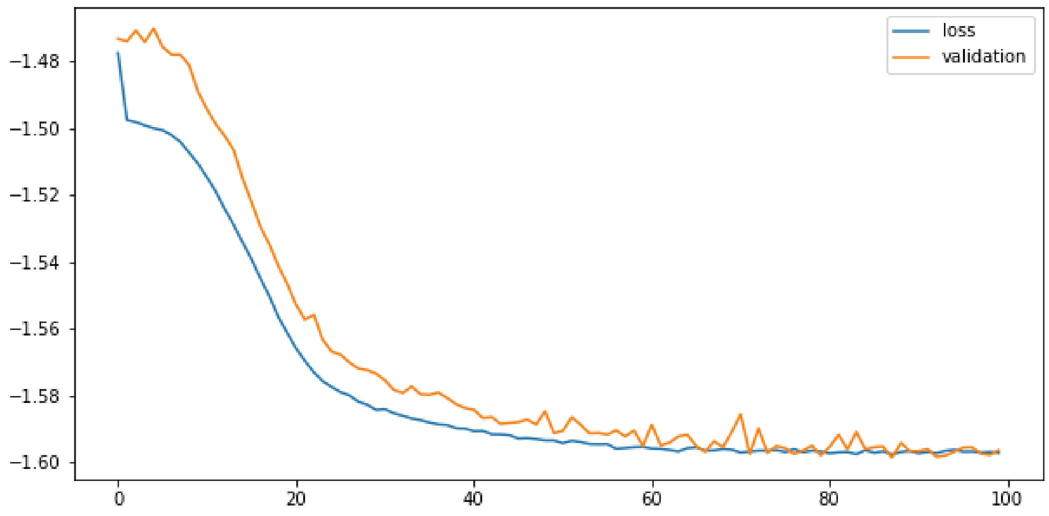


Figure 3.5: Neural network loss while learning 1D cubic SDE

True and approximated drift and diffusivity functions are shown in figure(3.6), as you can see that the approximated functions are quite good as compared to the original functions, specially the drift function predicted by the drift net.

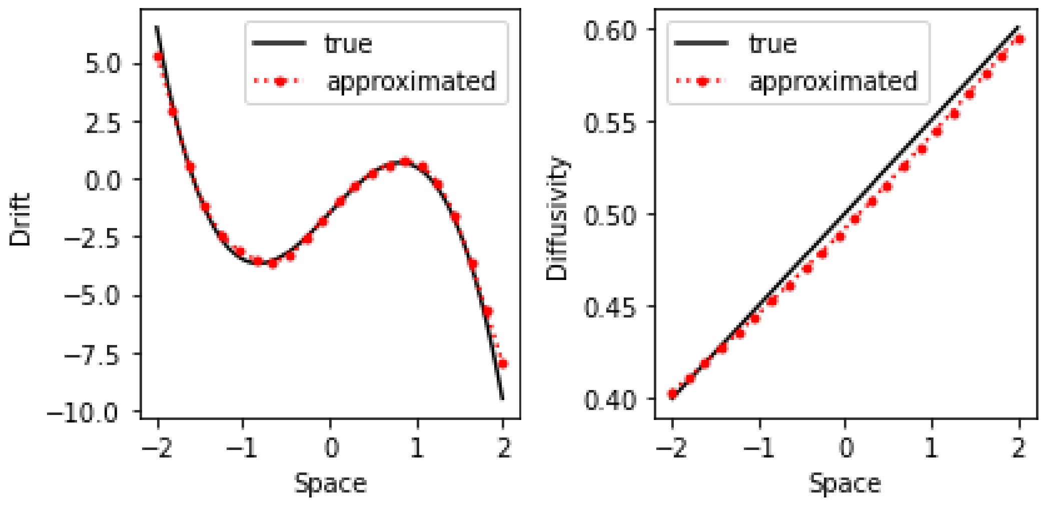


Figure 3.6: Neural network loss while learning 1D cubic SDE

3.6.2 Concert Scenario

As the name of this section suggests, this experiment is inspired by a real world scenario where people are confined in a closed space and do not move around too much. We will open this section by specifying details of the scenario and its attributes, the attributes not mentioned in this chapter are either default or described in input data section[3.3] of this chapter previously.

Simulation

The value of `"finishTime"` is set to 25.0 for this experiment, it means that the Vadere simulation runs from 0 to 25 Vadere seconds. `"simTimeStepLength"` is set to 0.999999, we have selected a bigger time step because we assume that the agents are not really moving within the box. Values of `"usefixedseed"` attribute are controlled by our code and go from $0 - 1000$ for training data and $0 - 100$ for the test dataset.

Topography

We build this scenario in Vadere, its graphical topography is shown in figure(3.7). Due to the topographical look of this scenario and for the sake of simplicity; we also call it ″Box scenario″ later on.

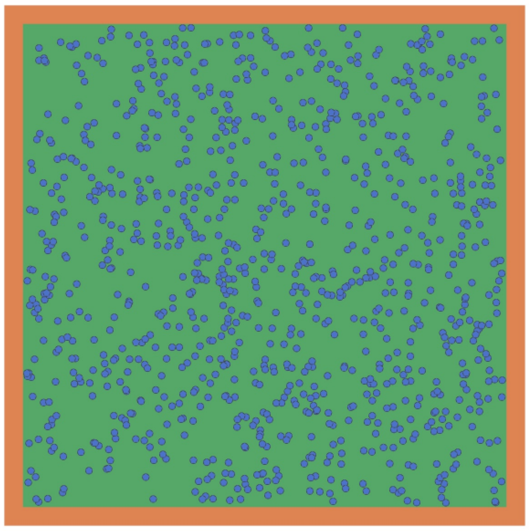


Figure 3.7: Vadere scenario inspired by a concert

The total area of floor field in this experiment is $30m \times 30m$. Value of `"createMethod"` attribute in `"AttributesFloorField"` set is set to `None`, this saves our computation resources and time while generating data sets.

Source

Source in this scenario is the green box at the top that creates all the agents. Value of `"spawnNumber"` in source is set to 1000.0 in order to create 1, 000 agents.

Target

Target is the orange box that lies underneath the source in this scenario. We have set the value of the `"absorbing"` attribute to `false` so that the agents do not exit the scenario and stay within it throughout the scenario.

Input Data

Input data for this scenario is generated by using the process described in section[3.3]. Number of days is set to 2 and number of iterations is set to 500 which gives us a total of 1, 000 simulations. After processing and normalization we get 4, 000 time snapshots. While creating the training data from Vadere we noticed that the data processor we were using, misses 12 different values, all of them almost at the beginning of the simulation. When we analysed these values further we found out that there was not really much change happening in the SIR system of our interest during this time period. Which is why we ignore this limitation as it does not have an impact on the quality of our input dataset.

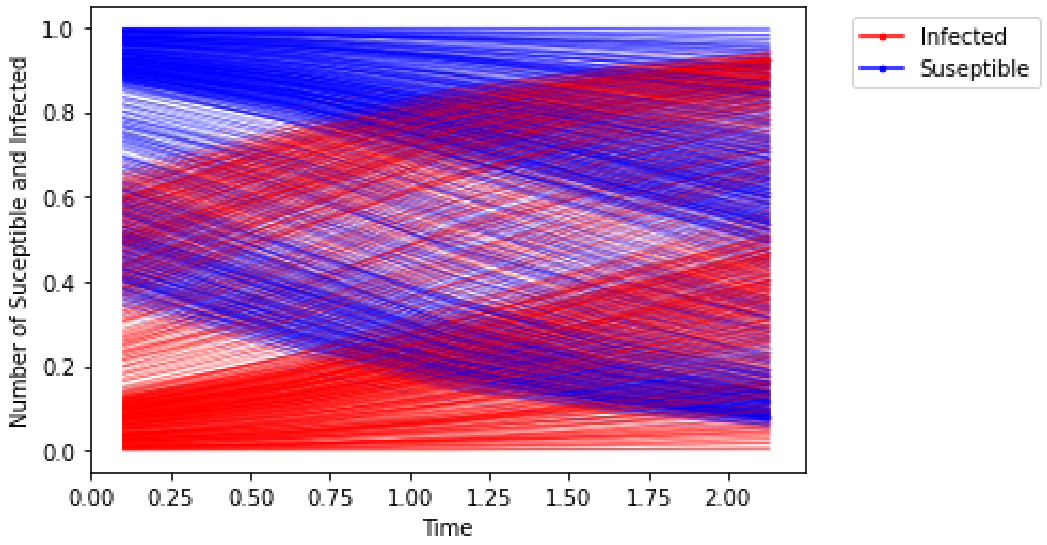


Figure 3.8: Number of susceptible and infected population over time

Input data that we have gathered for our system of interest over time is visualized in figure(3.8), number of susceptible population is represented in blue colored lines and infected population is denoted by red. It shows that number of susceptible population steadily decreases over time as the infection spreads, while the number of infected population increases. Some of the red and blue graphs start with higher initial value, it is because they possibility belong to later days in the simulated time duration.

Now let us explore the distribution of the training data that we have shown in figure(3.9), number of susceptible population is denoted by θ_0 and number of infected population is denoted by θ_1. We create a histogram that visualizes our dataset in multiple bins, number of input data points are stacked in each bin and the height of each bin represents the amount of data in it. Each bin in our histogram is defined by the value of θ_0 or θ_1; for example if there are 20 points with θ_0 value of 0.19, the height of the bar at x-axis 0.19 will correspond to 20 at y-axis.

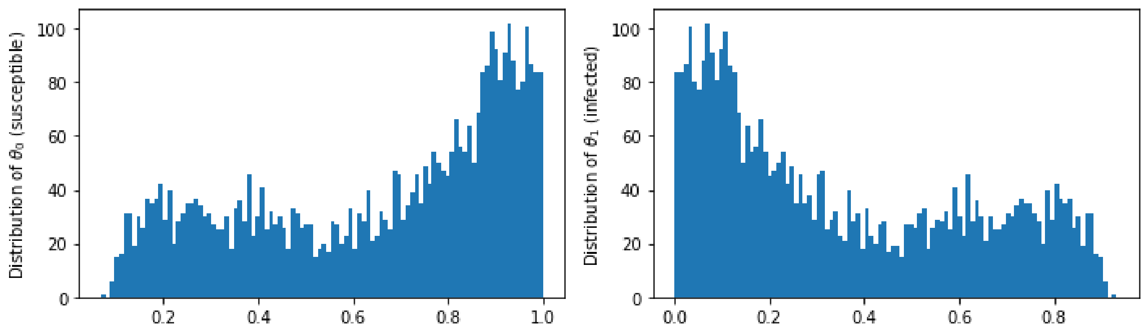


Figure 3.9: Distribution of θ_0 and θ_1

Values of θ_0 in this dataset are skewed more towards lower values i.e. between 0.0 and 0.2. Values of θ_1 in this dataset are skewed more towards higher values i.e. between 0.8 and 1.0.

Training, Validation and Loss

We train our neural network according to the hyperparameters defined in previous section. Our final training loss comes out to be -5.49 and our final validation loss turns out to be -5.24, which means that the network is training fairly well and have good values of loss in training as well as validation phase. Both training and validation loss converge after almost 20 epochs, which means that the computation time required to train our neural network is not that high. We trained the network for 100 epochs with $1,000$ time snapshot data and both validation and training combined took less than one minute. Training and validation loss are plotted in figure(3.10), with blue and orange lines respectively.

Test Data

At this point our drift and diffusivity networks are both fully trained so we move on to the testing phase of our network. In order to generate test data for this scenario we picked

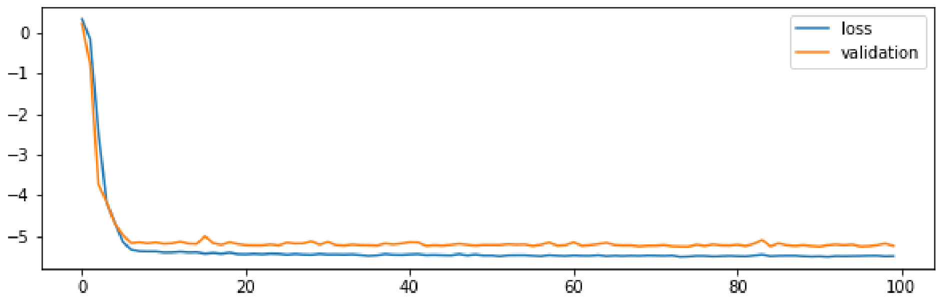


Figure 3.10: Training and Validation Loss curves

a step size of $2e^{-1}$, $\theta_0 = 0.95$ and $\theta_1 = 0.05$. At first, we picked the time duration of 2.5 seconds, which was also the total time used in the training data. Then we decided to extend it to 10 seconds to see how well our network performs then; which is 4 times that of that total duration we previously had. We followed the process similar to that of training data, except that we set number of days to 4 and number of iterations to 100 to generate 400 test points.

In the test data we have created piece-wise data for 10 seconds which is divided in 4 days, such that the first day represents the first 2.5 seconds and so on. We will be stacking this data on top of each other to get the full system data for 10 seconds. However, as we had noticed that the data processor we were using, misses 12 different values at the beginning of each simulation. In order to fill this gap, we took the last value of θ_0 from day one and the first value of θ_0 from the second day and approximated 12 values in between. Then we repeated the same process for θ_1.

True vs Approximated SDE Paths

Now that we have approximated the drift and diffusivity functions of an SDE we can construct SDE paths using them. We also have the ground truth from test dataset that provides us with the true SDE paths. We decided to do 100 iterations, which means that we will be generating 100 SDE paths for both true and approximated SDEs. We generated the approximated SDE by using the same initial conditions, step size and total time as we did to create the test data. It can be seen in figures(3.11,3.12); that the SDE paths predicted by the network are very close to the paths generated using Vadere.

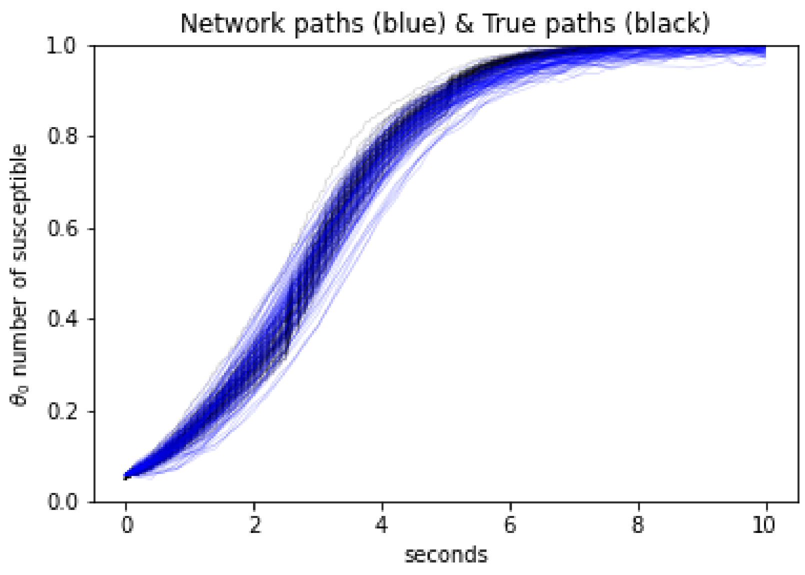


Figure 3.11: Network approximated (blue) and true (black) SDE paths forθ_0

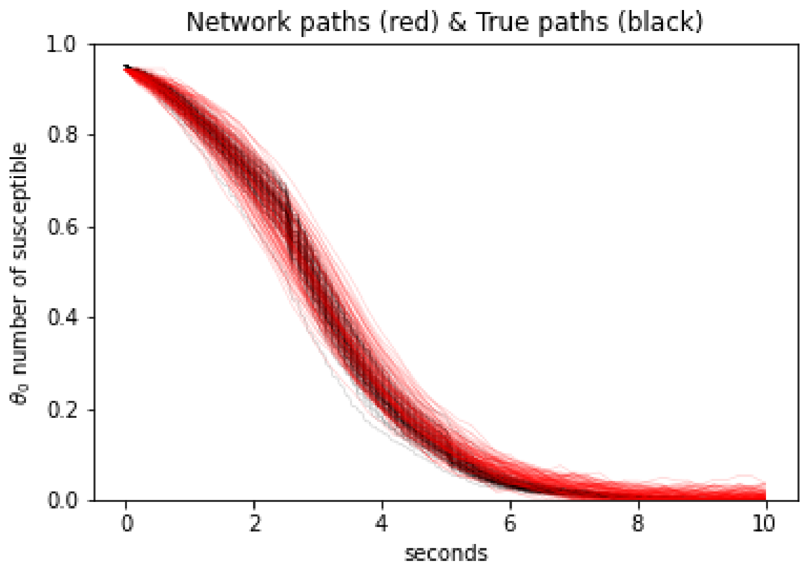


Figure 3.12: Network approximated (red) and true (black) SDE paths forθ_1

3.6.3 Pedestrians Exiting through a Door Scenario

As the name of this section suggests, this experiment is inspired by a real world scenario where people are leaving a space and in order to exit from it they are using a door. We will open this section by specifying details of the scenario and its attributes, the attributes not mentioned in this chapter are either default or described in input data section[3.3] of this chapter previously.

Simulation

The value of `"finishTime"` is set to 125.0 for this experiment, it means that the Vadere simulation runs from 0 to 125 Vadere seconds. `"simTimeStepLength"` is set to 0.4, we have selected a smaller time step because the agents are moving within this scenario and if we set a smaller time step their motion is smoother. Values of `"usefixedseed"` attribute are controlled by our code and go from $0 - 1000$ for training data and $0 - 200$ for the test dataset.

Topography

We build this scenario in Vadere, its graphical topography is shown in figure(3.13). Due to the topographical look of this scenario and for the sake of simplicity; we also call it Bottleneck scenario later on. The walls together with the door act like a small bottleneck, the agents have to stop just before the door and wait for their turn.

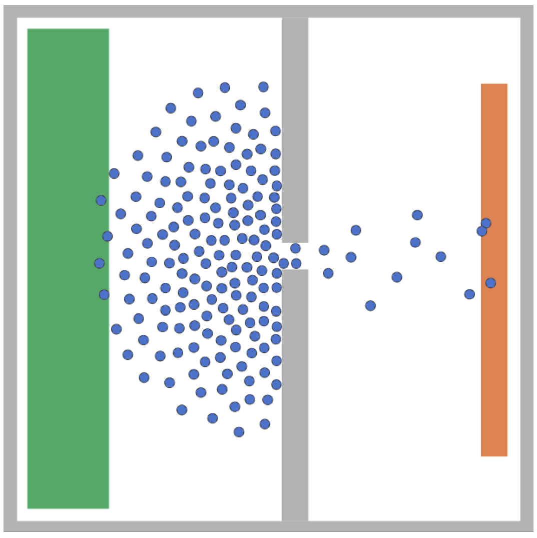


Figure 3.13: Vadere scenario inspired by a door through which pedestrians are exiting

The total area of floor field in this experiment is $20m \times 20m$. Value of `"createMethod"` attribute in `"AttributesFloorField"` set is set to

"HIGH_ACCURACY_FAST_MARCHING", it is necessary to create the floor field in this scenario because the agents move from one place to another and they need a floor to do so.

Obstacles

The obstacles in the middle are placed in such a way that the gap between them is almost Value $1m$, this gap be imagined to be a door. The agents go until the gap between the obstacles and if they can not pass through, they wait for their turn just before it. In this way the obstacles and the gap between them acts as a bottleneck in the scenario.

Source

Source in this scenario is the green box on the left side of the door that creates all the agents and spawns them towards the door. Value of "spawnNumber" in source is set to 200.0 in order to create 200 agents.

Target

Target is the orange box that lies right side of the door in this scenario. We have set the value of the "absorbing" attribute to `true` so that the agents can exit through the door.

Input Data

Input data for this scenario is generated by using the process described in section[3.3], similar to the previous experiment. Number of days is set to 4 and number of iterations is set to 250 which gives us a total of $1,000$ simulations. After processing and normalization we get $4,000$ time snapshots. While creating the microscopic data from Vadere we noticed that the data processor we were using, misses 12 different values, all of them almost at the beginning of the simulation. When we analysed these values further we found out that there was not really much change happening in the SIR system of our interest during this time period. Which is why we ignore this limitation as it does not have an impact on the quality of our input dataset.

Input data that we have gathered for our system of interest over time is visualized in figure(3.14), number of susceptible population is represented in red colored line and infected population is denoted by blue. It shows that number of susceptible population steadily decreases over time as the infection spreads, while the number of infected population increases. Some of the red and blue graphs start with higher initial value, it is because they possibility belong to later days in the simulated time duration.

Now let us explore the distribution of the training data that we have shown in figure(3.15), number of susceptible population is denoted by θ_0 and number of infected population is denoted by θ_1. We create a histogram that visualizes our dataset in multiple bins,

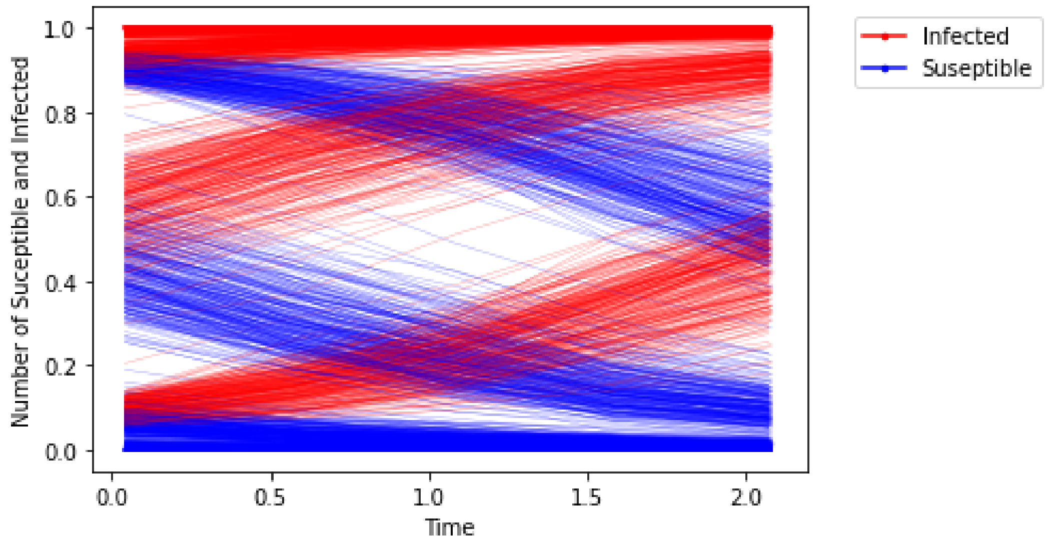


Figure 3.14: Number of susceptible and infected population over time

number of input data points are stacked in each bin and the height of each bin represents the amount of data in it. Each bin in our histogram is defined by the value of θ_0 or θ_1; for example if there are 40 points with θ_0 value of 0.25, the height of the bar at x-axis 0.25 will correspond to 40 at y-axis.

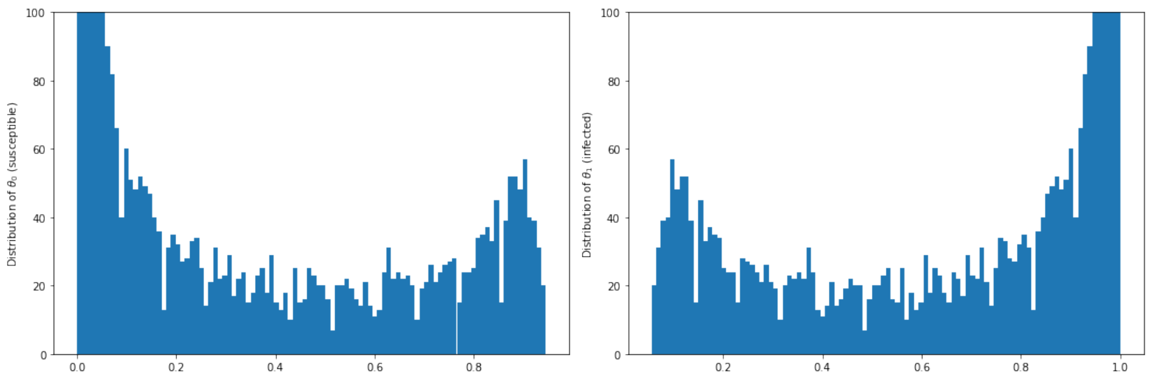


Figure 3.15: Distribution of θ_0 and θ_1

Training, Validation and Loss

We train our neural network according to the hyperparameters defined in previous section. Our final training loss comes out to be -2.88 and our final validation loss turns out to be

-2.84, which means that the network is training fairly well and have good values of loss in training as well as validation phase. Both training and validation loss converge after almost 20 epochs, which means that the computation time required to train our neural network is not that high. We trained the network for 100 epochs with $1,000$ time snapshot data and both validation and training combined took less than one minute. Training and validation loss are plotted in figure(3.16), with blue and orange lines respectively.

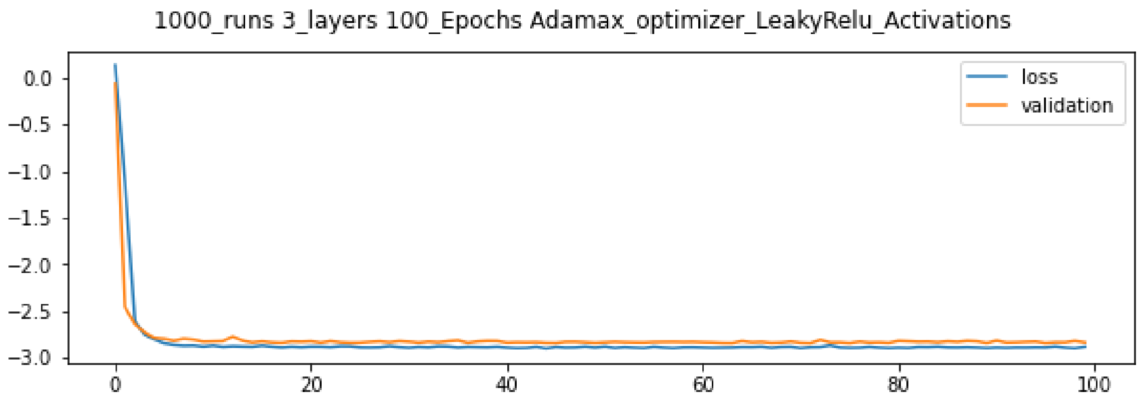


Figure 3.16: Training and Validation Loss curves

Test Data

At this point our drift and diffusivity networks are both fully trained so we move on to the testing phase of our network. In order to generate test data for this scenario we picked a step size of $2e^{-1}$, $\theta_0 = 0.85$, $\theta_1 = 0.15$ and total time of 12 seconds which was also used in training data. In this experiment we again followed the data generation process similar to that of training data, except that we run each scenario only once and start with initial conditions again to generate next test point. We set the number of iterations to 200 to generate 200 test points.

True vs Approximated SDE Paths

Now that we have approximated the drift and diffusivity functions of an SDE we can construct SDE paths using them. We also have the ground truth from test dataset that provides us with the true SDE paths. We decided to do 200 iterations, which means that we will be generating 200 SDE paths for both true and approximated SDEs. We generated the approximated SDE by using the same initial conditions, step size and total time as we did to create the test data. It can be seen in figures(3.17,3.18); that the SDE paths predicted by the network are close to the paths generated using Vadere in the beginning.

After almost 3 seconds the two paths diverge. On one hand in figure(3.17), the network paths go all the way down to 0 which means that according to the network predictions all the agents will eventually get infected. On the other hand, paths generated using our test data maintain a steady state at 0.1 value of θ_0, this means that 10% of the agents never get infected. Similar results show up in case of θ_1, network paths go all the way down to 0 and the test data maintains its value at 0.9. This reinforces the same result that 90% or the agents always stay susceptible.

This should not happen because the crowd of agents just before the bottleneck is similar to the box scenario that we work with in the first experiment. In the box scenario we have already seen that all the agents do get infected over time, this leads us to believe that this behavior is not normal and should be analyzed further. However, we have not analysed these results further during this project, it requires looking at the actual simulations while they are running in Vadere.

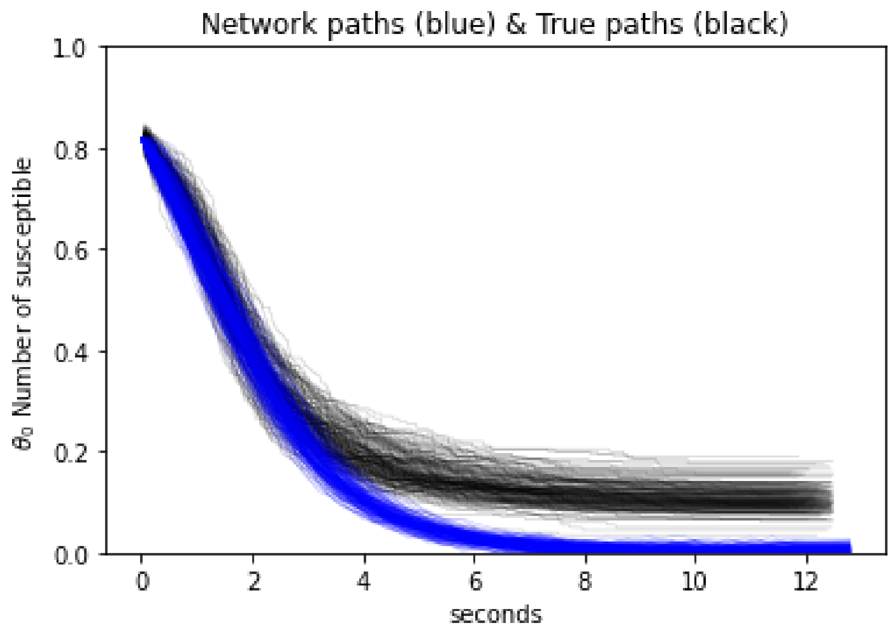


Figure 3.17: Network approximated (blue) and true (black) SDE paths forθ_0

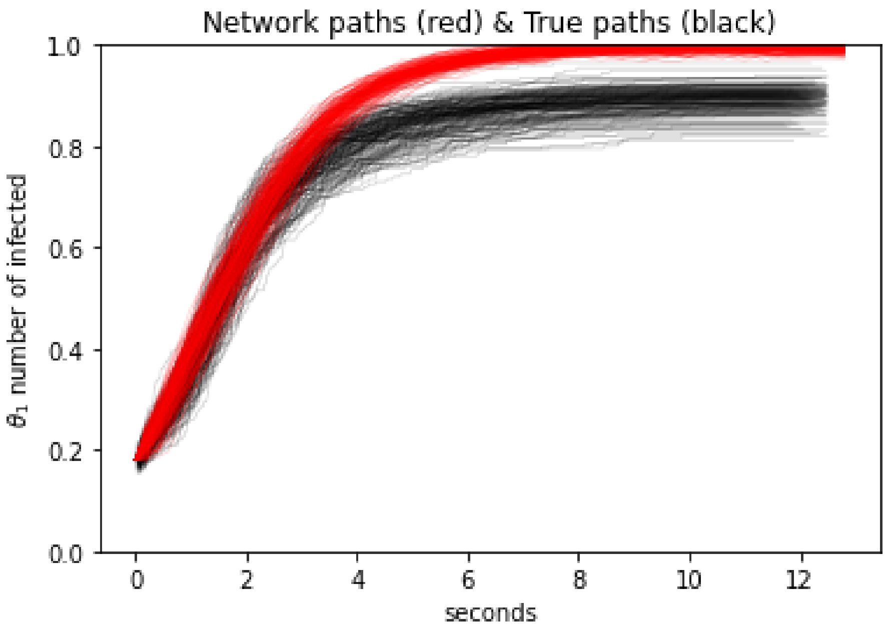


Figure 3.18: Network approximated (red) and true (black) SDE paths forθ_1

3.6.4 Key Challenges

Some of the key challenges that we faced during generation of data sets and neural network training are as follows:

- Generation of data using agent based modelling is a time consuming and cumbersome task; multiple iterations of data generation have to be done to get a good dataset
- On one hand the number of infected population rises quickly which means that we have a lot of data points where θ_1 is high; causing data skewness
- On the other hand the number of susceptible population declines sharply, we do not have many snapshots with high value of θ_0; contributing to data skewness
- At times agent based simulations behaved in a very unpredictable way due to which we had to go beyond our automation code that runs Vadere simulations and look at data records for a deeper analysis
- We found out that there is an imperfection in the data generated by Vadere; the output data processor that we were using, missed recording of data at 12 time steps

- Hyperparameter tuning was a lengthy and time consuming process

3.7 Evaluation metrics

We have used training loss function as a performance measure for our neural network. It evaluates how well the neural network learns the drift and diffusivity functions of the stochastic system and how accurately it can produce the SDE paths using these functions. A combination of training and validation loss is used to diagnose model overfitting, underfitting or convergence.

The problem we have at hand is a regression problem; in this case we can also apply other evaluation matrices as well that are used in regression setting. Root mean squared error (RMSE) or simply MSE is the most popular evaluation metric used for such problems. In case of our case we have a model that predicts future states on an SDE given an initial condition. We can use values from our test data as y_i and values predicted by the model $\hat{y}_i$, and calculate MSE as in equation(3.3) [68]:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.3}$$

Mean absolute error is another metric which calculates the average absolute distance between the true and approximated target values, it is defined in equation(3.4) [68]:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3.4}$$

If we are to apply this method to a real world problem; these evaluation metrics make sense only to the researchers who are either specialized in this area or at least have a good mathematical background. In such a setting we can use metrics such as Inliner Ratio metric, it calculates inliner ratio, which is the percentage of predicted data points with an error less than a margin [68].

4 Discussion and Conclusion

Throughout this project, we aimed to design and evaluate a neural network that is able to learn surrogate mathematical models to identify a stochastic dynamical system in the form of stochastic differential equations and later predict its future states. In order to demonstrate this methodology, we considered a stochastic system that is the spread of a contagious viral disease in a closed population. We used an agent based modelling and simulation framework to generate raw microscopic data. This raw data was processed and then used to train our neural network.

In the following section 4.1, we start our final discussion, highlight the contributions we have made, discuss the limitations of our approach and provide an overview of ethical concerns we should keep in mind while working with this approach. Section 4.2, concludes the discussion we start in this chapter. In section 4.3, we discuss possible future directions that can be taken to extend this work.

4.1 Discussion

We use an artificial neural network architecture SDE-NN consisting of two neural networks drift-net and diffusivity-net to approximate the drift and diffusivity respectively. We use this architecture to learn the stochastic dynamics in the form of coarse-surrogate stochastic differential equations; SDEs can predict future states of the system without having to know fine-grained dynamical data. Loss function of our neural network is inspired by an embodies the stochastic integration scheme; Euler-Maruyama scheme. Biggest advantage of such a neural network is that it already encodes a lot of structure in the algorithm itself which is why it does not need much training data to produce accurate results.

As a case study we consider agent based modelling and simulation systems; ABMs are great tools for capturing microscopic dynamics of a system. However, they are not general purpose, contain irrelevant details of they system, are computationally expensive and time consuming.

In order to demonstrate our methodology, we study the spread of a contagious viral disease in a group of people that are moving together in a crowd. Microscopic simulation data is created using Vadere which takes about 3 hours to generate for one scenario. We separate the data relevant to our system of interest, calculate the values of coarse-observable from this data, pre-process it and use it to train our neural network. After the neural network has been setup, it takes about 5 minutes of training and learn the coarse-surrogate models in the form of SDEs. Learned SDEs are used to approximate SDE paths and provide pretty good predictions compared with the true paths.

4.1.1 Contributions

Our contributions during this thesis project are as follows:

- We generate agent based simulations data and use them to calculate the values of coarse observables
- We use neural network loss function derived from a stochastic integration scheme; Euler-Maruyama scheme
- We explore and use a neural network training loss function to approximate the drift and diffusivity of an SDE from the coarse-observables
- We work with a system identification approach that does not require long time series: scattered paired snapshots (x_t, x_{t+h}) are sufficient.
- The approach works point-wise: No distributions need to be approximated from data, nor do we need to compute distances between distributions
- We demonstrate that we can use this approach to approximate the drift and diffusivity of an epidemiological model.

4.1.2 Limitations

During the course of this project we have discovered some limitations of this approach which are discussed in this section.

- The transition probabilities induced by the Euler-Maruyama scheme imply that, in general, it will not be possible to exactly reproduce the data-generation density. This limitation can be resolved by using more refined integration schemes such a Milstein method.
- On one hand, if the time step h between large number of data samples is too small, drift can not be accurately approximated because the diffusivity will be larger in this case. This can be resolved with approximating the diffusivity first with small h and then drift can be approximated using sub-sampled data sets. In section(2.3.5) we discussed that for convergence of Euler-Maruyama scheme equation(2.7), h has to go to zero. However, even with infinite data the drift is not easy to estimate.
- On the other hand, if the time step is too big, approximating the diffusivity function accurately is not possible.
- Presence of rare events in the coarse-grained observables is also a challenge, since not a lot of data will be available to learn them they will be ignored by the surrogate model.

4.1.3 Ethical Statement

Increase in capabilities of machine learning models as well as their accuracy has led them to be highly complex nested structures that are not easily understandable by human beings. Most of the machine learning models are applied in an inscrutable black-box manner. Biggest reason for this opacity is that heaps of data and prediction accuracy has been the driving force of machine learning research so far. Largest data set combined with a powerful and highly complex algorithm leads to more accuracy which is considered the yardstick of success. Careful dataset engineering can reduce the risk that the machine learning model will inherit biases of prior decision makers, research has shown that algorithms can also introduce unintended biases or amplify existing ones[28].

Black-box models for agent-based systems are also subject to unintended biases. Furthermore, they can be used in questionable ways that are not ethically acceptable if they are to be used in real world. One such example could be a biased or irresponsible generalization in predicting the behavior of societal systems with a learned black-box model. We as a machine learning community as well as a society should be very careful of such intended or unintended biases. However, there is a lack regulation in this area which stems from the lack of control over black box models.

In such situations, use of grey-box or physics informed systems is ideal. They provide a very clear mathematical explanation to their predictions and are easy to probe and decode. Our proposed methodology offers a physics-informed model of stochastic systems that is easy to interpret and its predictions are explainable.

4.2 Conclusion

Our proposed methodology can be used for modelling a dynamical system in a stochastic setting. Such an approach is very useful in situations where the stochastic system is partially observable, a small dataset is available or quality of the data is not very high. This methodology can be used for model reduction in computationally expensive problems. It can also be applied to many more agent-based models.

4.3 Future Work

- Different sources of training data can be used in the future
- We are using SEIR model to generate data for our system of interest, however we use only a reduced model containing the values of S and I; this can be extended predicting the to E and R values as well
- Other agent-based models can be coarse-grained using this approach in future
- Application of this approach can also be expanded to particle-based models as well

- Expansion of this work to a real world scenario such as COVID-19 spread with actual infection spread rates etc
- Hyperparameter tuning of the system identification neural network can be done using a specialized algorithm, we did not do this since it was beyond the scope of this project
- Numerical analysis or backward error analysis of stochastic integration schemes is one of our motivations. These integration schemes iteratively produce discrete time data that can be exploited by our identification algorithm to perfect its predictions iteratively. However, this type of analysis is not done in this project, but could be part of the future work
- Identification of stochastic partial differential equations (SPDEs) based on numerical analysis can be done
- Protein Folding is a very important and widely researched problem, it helps in disease understanding and drug discovery. However, it is a very computation intensive problem. In future a model reduction approach like we presented can be used to reduce computational complexity of problems like protein folding.

Bibliography

[1] Seir and seirs models. https://docs.idmod.org/projects/emod-hiv/en/latest/model-seir.html. Accessed: 2021-08-30.

[2] Viral diseases. https://www.atsu.edu/faculty/chamberlain/Website/Lects/Viral.htm. Accessed: 2021-08-30.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Sameera Abar, G. Theodoropoulos, P. Lemarinier, and Gregory M. P. O'Hare. Agent based modelling and simulation tools: A review of the state-of-art software. *Comput. Sci. Rev.*, 24(4):13–33, 2017.

[5] James A Anderson. *An introduction to neural networks*. MIT press, 1995.

[6] Osei Antwi. Stochastic modeling of stock price behavior on ghana stock exchange. 2017.

[7] Hassan Arbabi and Themistoklis Sapsis. Generative stochastic modeling of strongly nonlinear flows with non-gaussian statistics. *arXiv:1908.08941*, 2019.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[9] Mustafa Bayram, Tugcem Partal, and Gulsen Buyukoz. Numerical methods for simulation of stochastic differential equations. *Advances in Difference Equations*, 2018:17, 01 2018.

[10] A. Bazghandi. Techniques, advantages and problems of agent based modeling for traffic simulation. Number 3, 2012.

[11] Tyrus Berry, Dimitrios Giannakis, and John Harlim. Nonparametric forecasting of low-dimensional dynamical systems. *Physival Review E*, 91(3), 3 2015.

[12] Tom Bertalan, Felix Dietrich, Igor Mezić, and Ioannis G. Kevrekidis. On learning hamiltonian systems from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(12):121107, December 2019.

[13] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[14] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(2):7280–7287, 2002.

[15] Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

[16] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America*, 113(15):3932–3937, 2016.

[17] S. Cauchemez, N. Hozé, A. Cousien, B. Nikolay, and Q. T. ten Bosch. How modelling can enhance the analysis of imperfect epidemic data. *Trends in Parasitology*, 35:369 – 379, 2019.

[18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *NeurIPS conference 2018*, 2018.

[19] Carson C. Chow and Michael A. Buice. Path integral methods for stochastic differential equations. *The Journal of Mathematical Neuroscience (JMN)*, 5(1):8, 2015.

[20] Felix Dietrich. Learning effective stochastic differential equations from microscopic simulations: combining stochastic numerics and deep learning, 2021.

[21] Felix Dietrich and Gerta Köster. Gradient navigation model for pedestrian dynamics. *Phys. Rev. E*, 89:062801, Jun 2014.

[22] AD Dongare, RR Kharde, Amit D Kachare, et al. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194, 2012.

[23] L.C. Evans. *An Introduction to Stochastic Differential Equations*. Miscellaneous Books, 2013.

[24] R. González-García, R. Rico-Martínez, and I.G. Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers & Chemical Engineering*, 22:S965–S968, March 1998.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[26] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[27] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[28] Moritz Hardt, Eric Price, and Nathan Srebro. Equality of opportunity in supervised learning, 2016.

[29] K. Hasselmann. Stochastic climate models part i. theory. *Tellus A*, 28:473–485, 1976.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016.

[31] Sha He, Sanyi Tang, and L. Rong. A discrete stochastic model of the covid-19 outbreak: Forecast and control. *Mathematical biosciences and engineering : MBE*, 17 4:2792–2804, 2020.

[32] Herbert W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42:599–653, 2000.

[33] Desmond J. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Review*, 43(3):525–546, January 2001.

[34] Ajil Jalal, Andrew Ilyas, Constantinos Daskalakis, and Alexandros G. Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv:1712.09196v5*, 2017.

[35] Pengzhan Jin, Zhen Zhang, Ioannis G. Kevrekidis, and George Em Karniadakis. Learning poisson systems and trajectories of autonomous systems via poisson neural networks. *arXiv:2012.03133*, 2020.

[36] Ioannis Karatzas and Steven E. Shreve. *Brownian Motion and Stochastic Calculus*. Springer New York, 1998.

[37] David G. Kendall. *DETERMINISTIC AND STOCHASTIC EPIDEMICS IN CLOSED POPULATIONS:*, pages 149–166. University of California Press, 2020.

[38] W. O. Kermack and A. G. McKendrick. A Contribution to the Mathematical Theory of Epidemics. *Proceedings of the Royal Society of London Series A*, 115(772):700–721, August 1927.

[39] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, December 2014.

[40] Benedikt Kleinmeier, Benedikt Znnchen, Marion Gdel, and Gerta Kster. Vadere: An open-source simulation framework to promote interdisciplinary understanding. *Collective Dynamics*, 4(6), 2019.

[41] Franziska Klügl-Frohnmeyer and A. Bazzan. Agent-based modeling and simulation. *AI Mag.*, 33(5):29–40, 2012.

[42] Stefan Klus, Feliks Nüske, Sebastian Peitz, Jan-Hendrik Niemann, Cecilia Clementi, and Christof Schütte. Data-driven approximation of the koopman generator: Model reduction, system identification, and control. *Physica D: Nonlinear Phenomena*, 406:132416, May 2020.

[43] Lingkai Kong, Jimeng Sun, and Chao Zhang. Sde-net: Equipping deep neural networks with uncertainty estimates, 2020.

[44] A. Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012.

[45] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *arXiv:2001.01328*, 2020.

[46] Pablo Álvarez López, Michael Behrisch, Laura Bieker-Walz, J. Erdmann, Yun-Pang Flötteröd, R. Hilbrich, Leonhard Lücken, Johannes Rummel, P. Wagner, and Evamarie WieBner. Microscopic traffic simulation using sumo. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, 2018.

[47] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 3 2021.

[48] R. Lu, Xiang rong Zhao, Juan Li, P. Niu, Bo Yang, Honglong Wu, Wenling Wang, Hao Song, Baoying Huang, N. Zhu, Y. Bi, Xuejun Ma, F. Zhan, L. Wang, Tao Hu, H. Zhou, Zhenhong Hu, Weimin Zhou, Li Zhao, J. Chen, Y. Meng, Ji Wang, Y. Lin, Jianying Yuan, Z. Xie, Jinmin Ma, William J. Liu, Dayan Wang, W. Xu, E. Holmes, G. Gao, Guizhen Wu, Weijun Chen, Weifeng Shi, and W. Tan. Genomic characterisation and epidemiology of 2019 novel coronavirus: implications for virus origins and receptor binding. *Lancet (London, England)*, 395:565 – 574, 2020.

[49] C.M. Macal and M.J. North. Tutorial on agent-based modeling and simulation. In *Proceedings of the Winter Simulation Conference, 2005.*, number 1, pages 14 pp.–, 2005.

[50] Martin I Meltzer, Nancy J Cox, and Keiji Fukuda. The economic impact of pandemic influenza in the united states: priorities for intervention. *Emerging infectious diseases*, 5(5):659, 1999.

[51] Igor Mezić. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41(1):309–325, August 2005.

[52] V. Mnih, K. Kavukcuoglu, D. Silver, Andrei A. Rusu, J. Veness, Marc G. Bellemare, A. Graves, Martin A. Riedmiller, A. Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, A. Sadik, Ioannis Antonoglou, Helen King, D. Kumaran, Daan Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[53] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[54] GA Pavliotis. *Stochastic Processes and Applications Diffusion Processes, the Fokker-Planck and Langevin Equations*. Springer, 2014.

[55] N. Pelechano, J. Allbeck, and N. Badler. Virtual crowds: Methods, simulation, and control. In *Virtual Crowds: Methods, Simulation, and Control*, 2008.

[56] R Perez-Carrasco and JM Sancho. Stochastic algorithms for discontinuous multiplicative white noise. *Physical Review E*, 81(3):032104, 2010.

[57] Ali Rahimi and Ben Recht. Unsupervised regression with applications to nonlinear system identification. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 1113–1120, Cambridge, MA, USA, 2006. MIT Press.

[58] R. Rico-Martínez and I.G. Kevrekidis. *Nonlinear system identification using neural networks: dynamics and instabilities*, chapter 16, pages 409–442. Elsevier Science, 1995.

[59] R. Rico-Martínez, K. Krischer, I.G. Kevrekidis, M.C. Kube, and J.L. Hudson. Discrete- vs. continuous-time nonlinear signal processing of Cu electrodissolution data. *Chemical Engineering Communications*, 118(1):25–48, November 1992.

[60] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[61] Ronald Ross. An Application of the Theory of Probabilities to the Study of a priori Pathometry. Part I. *Proceedings of the Royal Society of London Series A*, 92(638):204–230, February 1916.

[62] Ronald Ross and Hilda P. Hudson. An Application of the Theory of Probabilities to the Study of a priori Pathometry. Part II. *Proceedings of the Royal Society of London Series A*, 93(650):212–225, May 1917.

[63] Michael J. Seitz. *Simulating pedestrian dynamics*. Dissertation, Technische Universitt Mnchen, Mnchen, 2016.

[64] Michael J. Seitz and Gerta Köster. Natural discretization of pedestrian movement in continuous space. *Phys. Rev. E*, 86:046108, Oct 2012.

[65] Kster G. Seitz MJ, Bode NW. How cognitive heuristics can explain social interactions in spatial movement. 2016.

[66] S. Shreve. Stochastic calculus for finance ii: Continuous-time models. 2010.

[67] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[68] Tilo Strutz. *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and Beyond*. Vieweg and Teubner, Wiesbaden, DEU, 2010.

[69] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

[70] C. Truesdell. Early kinetic theories of gases. *Archive for History of Exact Sciences*, 15(1):1–66, 1975.

[71] I. von Sivers, A. Templeton, F. Künzner, G. Köster, J. Drury, A. Philippides, T. Neckel, and H.-J. Bungartz. Modelling social identification and helping in evacuation simulation. *Safety Science*, 89:288–300, 2016.

[72] Isabella von Sivers and Gerta Köster. Dynamic stride length adaptation according to utility and personal space. *Transportation Research Part B: Methodological*, 74:104–117, 2015.

[73] Isabella von Sivers and Gerta Kster. Dynamic stride length adaptation according to utility and personal space, 2015.

[74] Liu Yang, Constantinos Daskalakis, and George Em Karniadakis. Generative ensemble-regression: Learning stochastic dynamics from discrete particle ensemble observations. *arXiv:2008.01915v1*, 2020.

[75] Shihao Yang, Samuel W. K. Wong, and S. C. Kou. Inference of dynamic systems from noisy and sparse data via manifold-constrained gaussian processes. *Proceedings of the National Academy of Sciences*, 118(15):e2020397118, 2021.

[76] Cagatay Yildiz, Markus Heinonen, Jukka Intosalmi, Henrik Mannerstrom, and Harri Lahdesmaki. Learning stochastic differential equations with with Gaussian Processes without gradient matching. In *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, September 2018.

[77] Aiqing Zhu, Pengzhan Jin, and Yifa Tang. Deep hamiltonian networks based on symplectic integrators. *arXiv:2004.13830*, 2020.

[78] K. C. Zygalakis. On the existence and the applications of modified equations for stochastic differential equations. *SIAM Journal on Scientific Computing*, 33(1):102–130, January 2011.