DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

**Master's Thesis**

# Machine Learning of Stochastic Differential Equations for Infection Models using Neural Networks

**Atalay Furkan Yırık**

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

**Master's Thesis**

# Machine Learning of Stochastic Differential Equations for Infection Models using Neural Networks

# Machinelles Lernen Stochastischer Differentialgleichungen für Infektionsmodelle mit Neuronalen Netzwerken

| | |
|---|---|
| Author: | Atalay Furkan Yırık |
| Supervisor: | Dr. rer. nat. Felix Dietrich |
| Submission Date: | 15.04.2023 |

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

15.04.2023 Atalay Furkan Yırık

Acknowledgments

I thank Dr. Felix Dietrich who supervised me through the study. I also thank my family, my mother Gullu, my father Murat, my brother Mert who supported me during the study for the external ups and downs.

# Abstract

Mathematical models which take the observed data into account provide favourable insights for dynamical systems. In this project, we identify and model the behaviour of coarse-grained particles in dynamical crowd setting from data which we observed through number of experiments. The dynamical system we study focuses on the spread of a contagious disease. We represent the behaviour of the coarse-grained particles - which are infectious agents in this case- by stochastic differential equations (SDE). In the first place, we gather corresponding agent-based simulation data for different scenarios which the spread of the virus is probable in daily life cycle through our simulation tool 'Vadere'. We then approximate the drift and diffusivity functions of SDEs through Artificial Neural Networks (ANN) which can be considered as effective stochastic ResNets. In our ANN structure, our loss function is inspired by low order approximators , namely, Euler Maruyama and Milstein methods. We then identify the surrogate models for virus spread/coarse grained particles in a stochastic crowd dynamics setting with the learned model. The infected rate for each individual is learned through the model and used to predict the likelihood of the viral infection spreading to other individuals in the crowd. We then visualized the spread of the infection through the crowd using a network with nodes representing population in an environment. The results of this study provide valuable insights into the potential spread of a viral infection in a crowd.

# Contents

# 1 Introduction

In this study, we worked on the spread of infections. Our goal was to model infection spread through representing the spread with stochastic differential equations and learning them with machine learning and finally constructing a graph for this representation.

The document consists of a couple of chapters:

- Chapter **State of The Art** : It is literature research part which mainly focuses on simulation modeling, SIR model, stochastic differential equations, artificial neural networks and modeling dynamics of infection spread on graphs. The chapter mentions about the main topics which this study is mostly related.
- Chapter **Main Part**: This chapter constitutes the body of the document. It includes task description and motivation, general overview to the process, experiments and graph construction to combine local infection models.
- Chapter **Conclusion & Future Work** : This chapter summarizes the work and mentions about the obstacles during the work, possibilities to widen the domain for future studies.

# 2 State of The Art

## 2.1 Simulation Modelling

Modeling real-world dynamical systems is a complicated task to achieve. Throughout history, many methods proposed to tackle the problem of having a general framework for mapping these systems. Mathematically modeling a complex system which is the spread of viral disease in our case through simulation is the core of this project. Current research areas on that subject could be classified into 3 categories.

### 2.1.1 System Dynamics

This method is first created by MIT professor Jay Forrester in the 1950s. His idea was to use electric circuits and the law of Physics to investigate the behavior of economics, stocks, and flows where a flow is the accumulation of stock prices. It is a differential equation-based model [1]. Feedback effects and delays are major System Dynamics elements. These models are known for strategic decisions or decision problems affecting a population [2, 1].

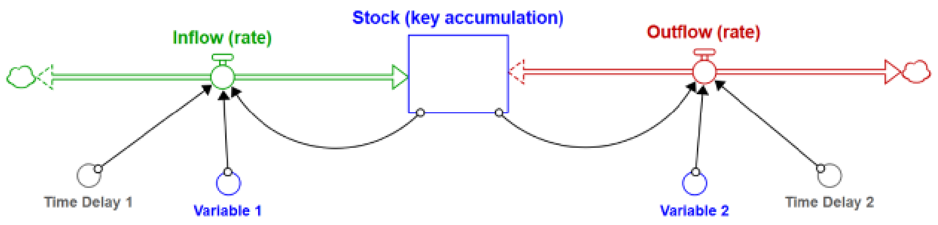


Figure 2.1: How the resources flow in System Dynamics [3].

### 2.1.2 Discrete Event Simulation

Another method whose first implementation roots go back to the 1960s by a former IBM engineer Geoffrey Gordon. [4] Discrete Event Simulation models the dynamical systems as operations which are a sequence of events in time. The states change at each event occurrence at a particular instant time. To simply put, the system is considered a process being performed through entities. They are suitable for modeling resource availability. [5] Typical use cases of the methods include the utilization of resources, time spent in the system or its part by an agent, waiting times, queue lengths, system throughput, bottlenecks, cost of the agent processing, and its structure. [4]

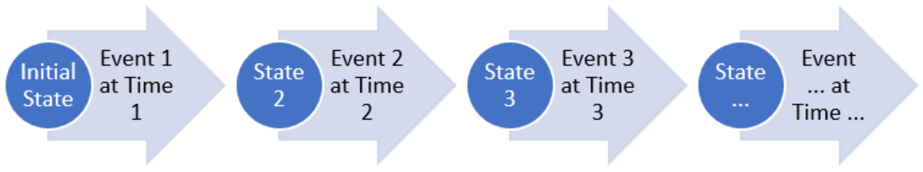


Figure 2.2: Discrete event simulation

### 2.1.3 Agent Based Modelling

Agent-Based Modelling(ABM) is another simulation modeling technique that is relatively new compared to System Dynamics or Discrete Event Models. In ABM, a dynamical system is modeled as a structure consisting of autonomous entities called agents [6]. In other words, it is a computational model for simulating the actions and interactions of autonomous agents in order to understand the behavior of a system and what governs its outcomes [7].

**Why is ABM becoming widespread?**

1. We live in a complex world. The dynamical systems we analyze are complex enough that we apply some assumptions due to interdependencies. ABMs allow us to flex these assumptions by inspecting individual agents rather than system-wide assumptions. [8]
2. Data collected for these systems are becoming more fine-grained. ABMs provide computing individual based simulations. [8]
3. By recent advancements I computational powers for CPU and memory it is now more efficient to model large-scale social interactions.
4. Advancements in theoretical computer science research and its applications for software development pipelines, UML diagrams, charts and etc. allow us to take more benefits from new technologies. [4]
5. Availability to have deeper insights into the system behavior which is not possible through traditional approaches. [4]

**What is 'Agent'?**

There are varying definitions of agents in the ABM setting, however, we could consider an agent as the main component which behaves based on certain set of rules in a dynamical system. For practical reasons it could be simplified in Table 2.1 and Table 2.2

| **Must-have agent features** | **Explanation** |
|---|---|
| Autonomous andself-directed | Agent functions independently. |
| Modular or self-contained | Agent is a identifiable, discrete individual which could be differed from its characteristics,behaviorr, decision-making capability. |
| Social | Agent interacts with other agents and could change its or system's state based on these interactions. |

Table 2.1: Must-have Agent Features

| **Optional agent features** | **Explanation** |
|---|---|
| Live in an environment | Agent may be situated in the environment which it changes its state. |
| Have explicit goals that drive its behaviour | The goals may or may not be objectives to maximize. |
| Have the ability to learn | Agent adapts its behavior based on its experience. |
| Have the resource attributes which indicate its current stock of resources | For example, wealth, information, health condition, energy, etc. |

Table 2.2: Optional Agent Features

### 2.1.4 An ABM Example:

Simulation of Boids (Birds) could be considered a good and mainstream example to examine how an agent interacts with other agents and the environment within. It was first proposed by Craig Reynolds to analyze the organized behavior of leaderless flocks [10]. Modeling it from the perspective of ABM provides good insights into the movement patterns of birds and their flocking behavior in a structured and organized manner. A bird which is an agent has predefined rules to follow in the simulation environment.

| **Rules** | **Behavior** |
|---|---|
| Separation | Each agent steers to avoid local flockmates. |
| Cohesion | Each agent steers to move towards the average position of its nearby flockmates (set by a predefined region area). |
| Alignment | Each agent steers towards the average heading of its local flockmates. |

Table 2.3: Rules of a *Boid*

Other rules such as predator avoidance, obstacle avoidance or more complex ones such as the direction while encountering other flocks could also be applied to enrich the complexity of the model. However, even applying these 3 basic individual rules is enough to recognize the collective movement of the boids and patterns for flocking behavior in ABM setting. Interaction between agents without a lead and behaving as a group is one of the major takeaways from the model to understand what ABMs could offer. Figure 1 shows the start of the simulation of boids with no flocking formation. Figure 2 shows the end state of the simulation with the applied 3 basic rules. The formation of flocking behavior could be noticed through the use of ABM simulation.

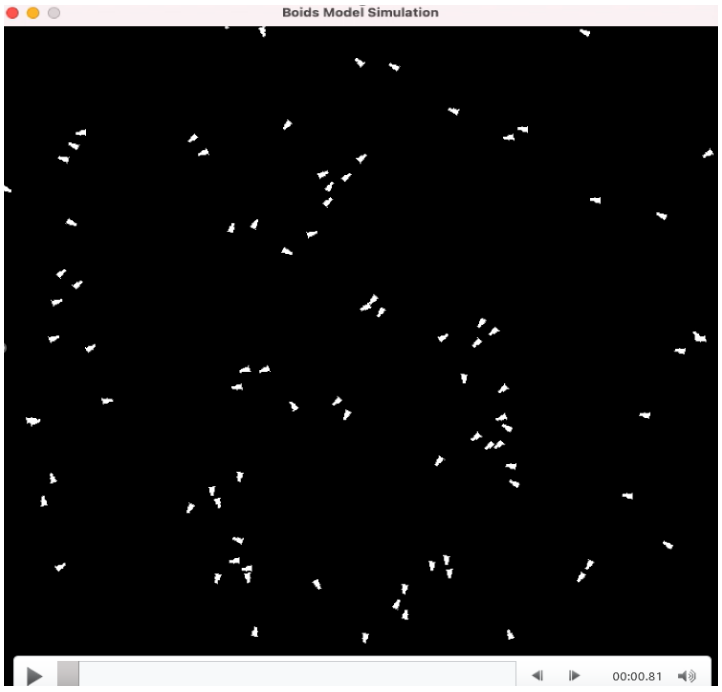


Figure 2.3: Boids simulation at start

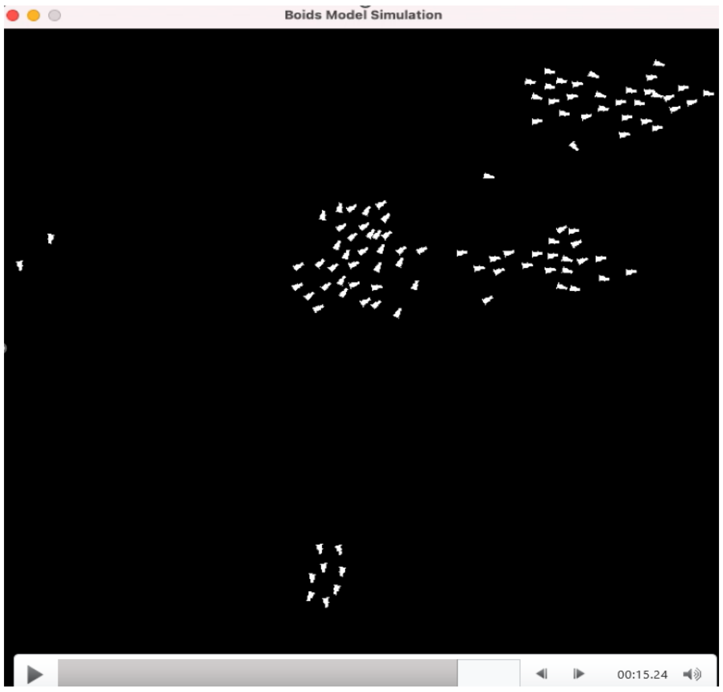


Figure 2.4: Boids simulation after certain time t

### 2.1.5 Vadere and ABM Simulation

Vadere is an open-source framework for the simulation of microscopic pedestrian and crowd dynamics. It provides generic model classes and visualization and data analysis tools for two-dimensional systems. It is developed by Kleinmeier et. Al. at Technical University of Munich and Munich University of Applied Sciences [9]. In our work for simulations of viral spread disease in crowds, we use Vadere. It is open source, has user friendly GUI, has many modeling options and it offers much flexibility when it comes to creating scenarios containing short term basis pedestrian movements. The fact that focus of Vadere researchers is walking behaviour of a pedestrian while implementing the framework makes it focus on locomotion models. Therefore it offers useful and reliable simulation models for a viral infection which is affected mainly by movement of individual pedestrians in the crowd. Namely, Optimal Steps Model (OSM) by Gerta Köster et. al. [10], gradient navigation model which is first proposed by Dietrich and Köster [11], Social Force Model which was proposed by Helbing et. al. [12] are the examples for available locomotion models in Vadere. These models are also the basis for simulation studies.

In our current work, we focus on Optimal Steps Model. OSM defines motion by a series of discrete footsteps [9].To simply put, in OSM, agents (pedestrians in our case) are attracted by targets and repulsed by obstacles and other agents. Agents maximize their utility which is based on distance to the targets and to other agents (negatively affecting) . Agents move on the available optimal position in a circular region set by their current spot. The circular region's radius is agent's maximum stride length [13]. Agents' movements in OSM are motivated by 'target orientation', 'pedestrian avoidance' and 'obstacle avoidance' [14].

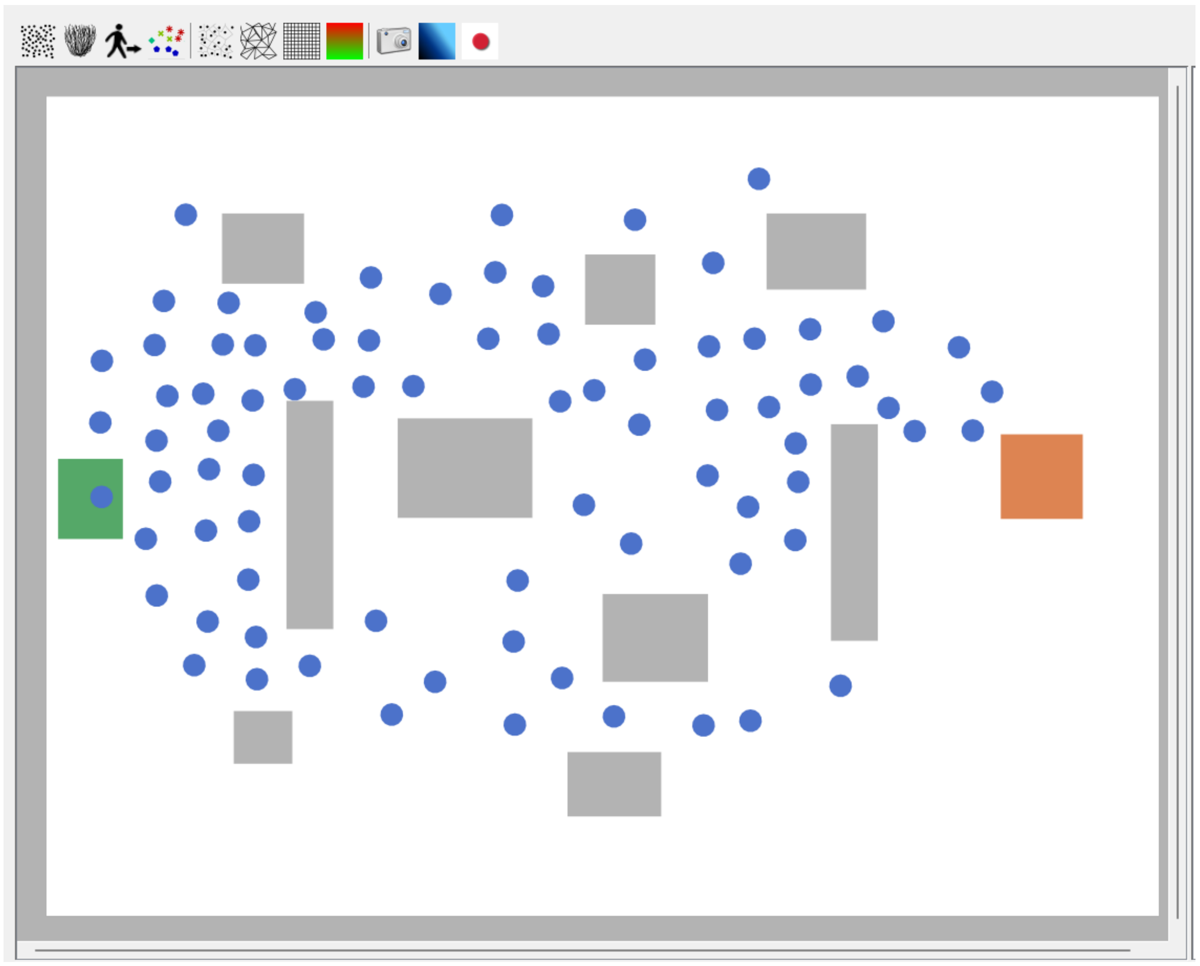


Figure 2.5: Dynamical simulation of a crowd moving out from a source towards a destination region in Vadere at an instant time.

| | |
|---|---|
| Blue dots | Each agent steers to avoid local flockmates. |
| Orange Square | Each agent steers to move towards the average position of its nearby flockmates (set by a predefined region area). |
| Green Square | Each agent steers towards the average heading of its local flockmates. |
| Gray Areas | Walls, or other types of obstacles. |

Note : Other individual agents are also considered as obstacles and agents try to avoid from collisions.

Table 2.4: Legend in Figure 2.5

## 2.2 SIR Model

SIR model is one of the simplest variations of Compartmental Models which are mainly used in epidemiology to mathematically model an infectious disease. It was first published by McKendrick and Kermack in 1926 [15, 16]. In SIR, the population is assigned to 3 compartments with labels as S, I, R (Susceptible, Infectious, Recovered). To further explain the model, suppose a population of size of N with only one infectious state. Also assume that an agent or a person could be either susceptible, infected or recovered. If the disease is fatal recovered means death. Let S(t), I(t) and R(t) be the number of people in each state. The SIR model proposes 3 nonlinear ordinary differential equations,

$$\frac{dS}{dt} = -\alpha SI, \tag{2.1}$$

$$\frac{dI}{dt} = \alpha SI - \beta I, \tag{2.2}$$

$$\frac{dR}{dt} = \beta I, \tag{2.3}$$

where t is time $\alpha$ is infection rate and $\beta$ is the recovery rate. There are a couple of assumptions in this, first one is population size N is fixed and probability of having infection of an individual is the same for each individual.
The key value governing the time evolution of the equations is $R_0 = \frac{\alpha S}{\beta}$, which is basic reproduction number.

There are models which include E (Exposed), D (Deceased), V (Vaccinated), M (Maternally-derived Immune) states. In our work, we mainly use SIR/SEIR models in the simulations.

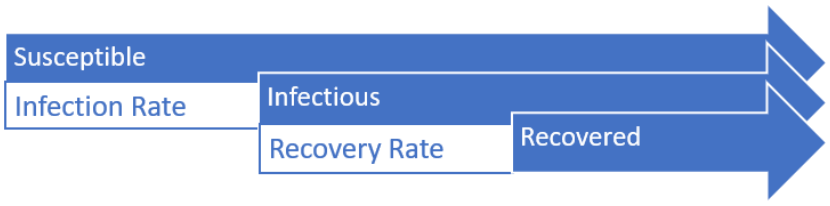


Figure 2.6: Flow of SIR model.

## 2.3 Stochastic Differential Equations

In this study Stochastic differential equations (SDEs) play role in modelling the spread of the viral study. In order to better understand SDEs, we need to mention about stochastic processes, stochastic systems and will come back again to SDEs.

Stochastic differential equations (SDEs) are differential equations that include one or more random or noisy variables. Models which display randomness or stochasticity could be represented by SDEs. They are used in various fields such as physics, engineering, economics to model systems that are subject to random fluctuations.

### 2.3.1 Stochastic Process

A formal definition of a stochastic process is a collection of random variables ($X_\theta$), indexed by a parameter $\theta$ where $\theta$ belongs to index set $\Theta$. Stochastic processes have applications in many fields including biology, chemistry, ecology,neuroscience, physics, computer science. Usually index set displays time steps. There are different ways to classify stochastic processes. If the index set is a set of integers, the process is called **discrete time** stochastic process. If it is a line of time or an interval of time, it is called **continuous time** stochastic process. In our study, time steps are discrete to show the spread of the disease. There are different types of specific stochastic processes such as :

- Random walks
- Bernoulli process
- Wiener process
- Poisson process
- Markov processes
- Lévy processes

### 2.3.2 Stochastic Process Application Examples:

- **Epidemic models**

Another application area of stochastic processes that would be helpful for understanding our study is epidemic models.

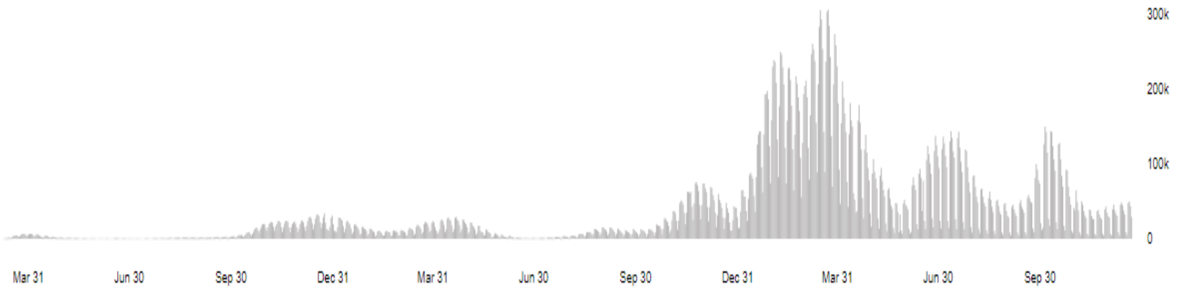


Figure 2.7: Covid cases in Germany for the period between March 2021 - November 2022 the period 1/1/02-31/7/04. [17]

Here in Figure 2.7, both the time and the number of cases are discrete. The number of infectious cases, probability of infection spread in a region or specific population could be modeled as a stochastic process.

### 2.3.3 Stochastic Differential Equations

A stochastic differential equation (SDE) is a differential equation in which at least one term is a stochastic process. Even though they might be random functions, their statistical properties (e.g. underlying assumptions, coefficients ) should be given. [18] What mainly makes SDEs different than Ordinary differential equations (ODE) is SDEs involve random processes unlike ODEs. Thus, the solution to SDEs includes probability density functions. A typical SDE could be shown as :

$$\mathrm{d}X_t = \mu(X_t, t)\,\mathrm{d}t + \sigma(X_t, t)\,\mathrm{d}B_t, \tag{2.4}$$

where $X_t$ is random variable, $B_t$ is a Wiener process ( standard Brownian motion). The function $\mu$ corresponds to drift coefficient and $\sigma$ corresponds to diffusion coefficient. The drift term represents the deterministic part of the solution, while the diffusion term represents the random part.

### 2.3.4 Euler-Maruyama Scheme

One of the methods to solve an SDE is Euler-Maruyama method. It approximates the solution with any level of accuracy. In this study, we take the advantage of Euler method for loss functions in neural network training, which was proposed by Dietrich et. al. [19]. Other methods for numerically solving an SDE include Milstein and Runge-Kutta methods.

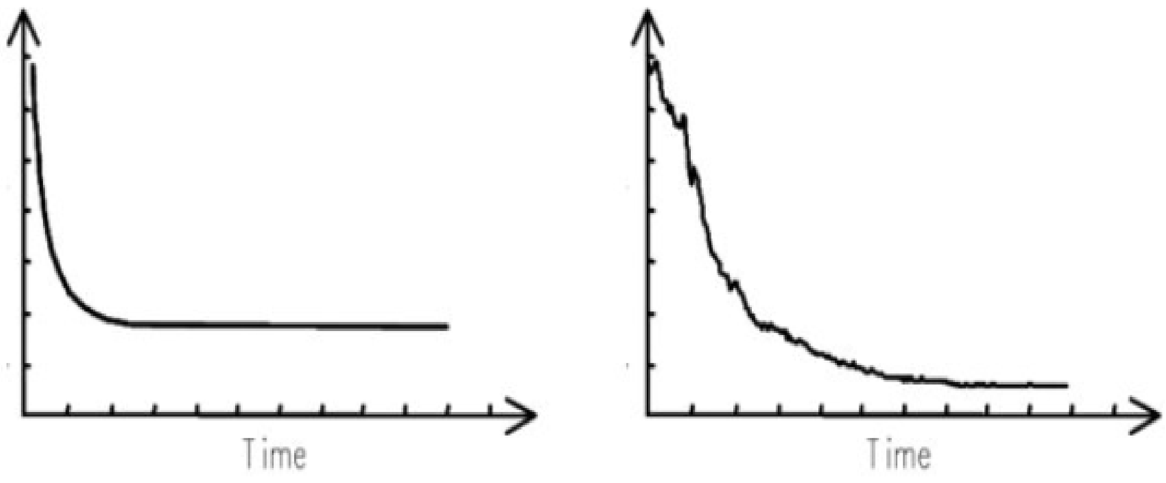


Figure 2.8: Deterministic vs. Stochastic Difference Example

## 2.4 Artificial Neural Networks

Artificial Neural Networks are computing systems inspired by the function and structure of biological neural networks of an animal brain. The first attempts of ANN architectures copied the mechanism of the nature of the brain in a way of simulating the electrical activity of the brain and nervous system. Each node carries the information to its adjacent nodes in the following layer through connections and gives an output which is analogous to that each neuron transmits the electrical signal through synapses to its adjacent neurons and gives firing frequency. The information that nodes in ANN convey possess weight which affects the influence of the nodes on the next nodes.

The idea of combining processes into a network was drilled back in the 40s by the perceptron algorithm which is one of the first examples of the simplest neural networks. It was first proposed by McCulloch and Pitts as artificial neuron then developed by Rosenblatt as the first perceptron learning algorithm [21, 22].

### 2.4.1 Training

Neural networks learn the most likely outcome by iterating the input data with weighted connections. Training is the process to decide the parameters that have most likely match between the observed data and prediction. After training, we evaluate the difference between the prediction from the model we trained with training data for unseen data and its true values.

### 2.4.2 Hyperparameters

Hyperparameters are constant parameters whose values are set before the training process. The word hyper refers that they are top level. Typical hyperparameters are as such :

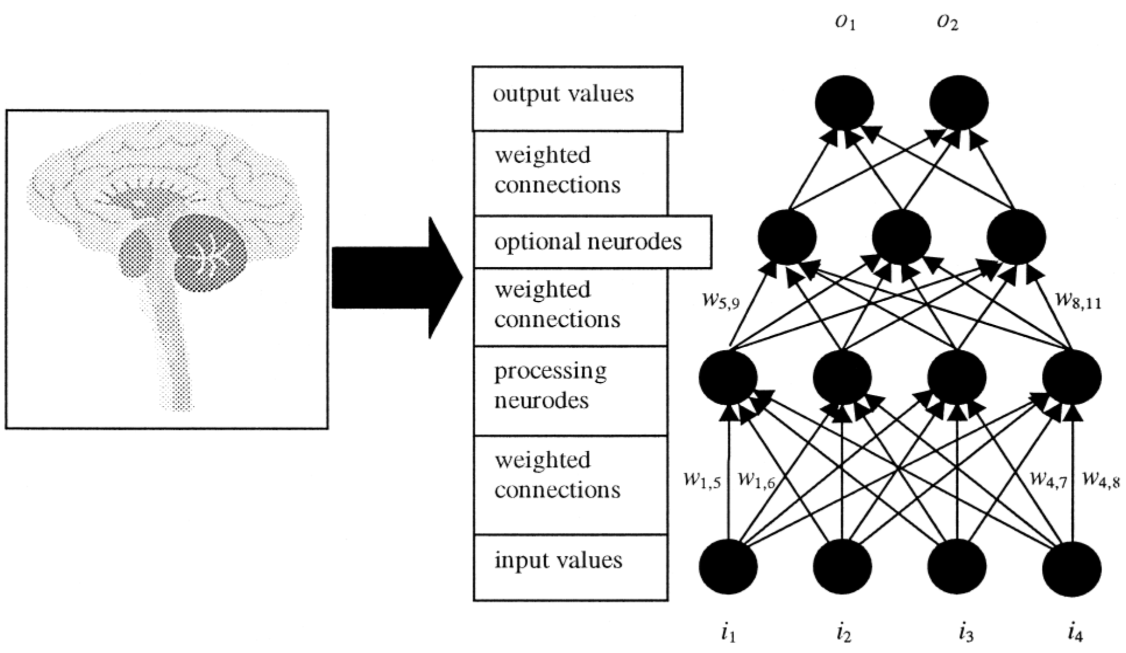


Figure 2.9: ANN Archhitecture Example [20]

- Optimization algorithm (Stochastic gradient descent, Adam , AdaMax , Adagrad etc.)
- Learning rate in the chosen optimization algorithm
- Type of loss function
- Number of layers
- Type of activation function
- Number of Epochs/Iterations
- Batch size
- Dropout rate
- Pooling size
- Filter/Kernel size

The examples could be added or removed depending on the model's usage. As an example, decision for number of clusters is a hyperparameter in a clustering task. Fine-tuning of these hyperparameters which give better results for the specific data and model structure plays a key role in having the best results.

### 2.4.3 Loss Function

Loss function or sometimes called the cost function is one of the most important hyperparameters which needs to be decided. It is the function to measure the difference between the true output and the training model. Based on the difference result, the parameters are adjusted iteratively to have the optimally least difference. In this project, we use a loss function inspired by Euler-Maruyama Scheme.

### 2.4.4 Euler-Maruyama inspired Loss Function

As stated earlier, the loss function used in this study is inspired from Euler-Maruyama scheme by the work from Felix et. al. [19] The explanation for it is as such in the training dataset $D$ we don't have access to drift $f$ or diffusivity $\sigma$. However, we have access to the variables $x_0$, $s_1$ and $h$. $x_1$ is the points which are drawn from the multivariate normal distribution conditioned on $x_0$ and step size $h$.

$$x_1 \sim \mathcal{N}(x_0 + hf(x_0), h\sigma(x_0)^2) \tag{2.5}$$

To approximate drift and diffusivity, we define a probability density $p_\theta$ of the normal distribution then we take the log-likelihood of the data $D$ under the assumption in equation 2.5. When we put the network weights $\theta$ we get the log-likelihood equation as such:

$$\theta := \arg\max_\theta \mathbb{E}[\log p_{\hat{\theta}}(x_1|x_0,h)] \approx \arg\max_\theta \frac{1}{N}\sum_{k=1}^{N} \log p_{\hat{\theta}}(x_1^{(k)}|x_0^{(k)}, h^{(k)}) \tag{2.6}$$

Since now we set the log-likelihood of our loss function, we could take the derivative of it to get the point where the likelihood is high. Constant terms are also removed because they do not affect the minimum likelihood points. In the end, we have a loss function as such:

$$L(\theta|x_0,x_1,h) := \frac{(x_1 - x_0 - hf_\theta(x_0))^2}{h\sigma_\theta(x_0)^2} + log \mid h\sigma_\theta(x_0)^2 \mid + log(2\pi) \tag{2.7}$$

## 2.5 Modelling dynamics of infection spread on graphs

Modelling infection spread is a complex task which requires that a number of variables must be taken into account. A part of this study focuses on modelling the spread of the disease in a location that covers certain area where the population commute for work, visit social places, stores, use transportation channels. In a recent study, this type of modelling is done for the German city of Tübingen. The sites include educational institutions (universities, public schools),social places(restaurants, bars, cafes),bus stops, supermarkets, offices. The viral disease in this case is COVID-19. The research spotlights an epidemic model which was benefitted by spatiotemporal data for Tübingen. The research also takes a couple of different variables such as mobility, testing, tracing, social distancing into account while modelling the epidemics. [23]

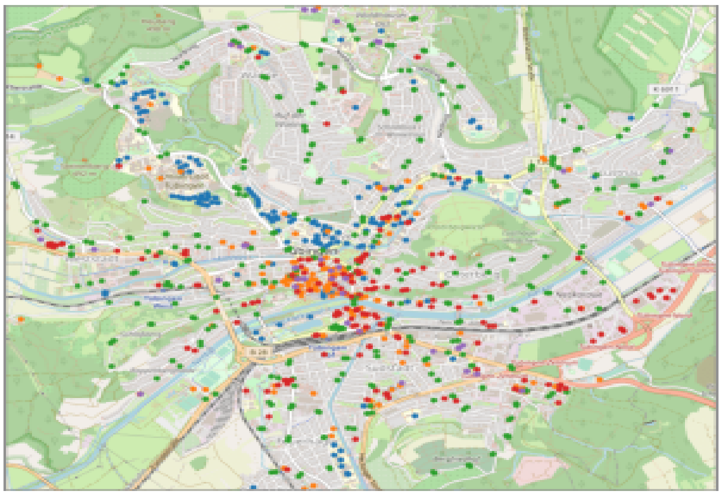


Figure 2.10: Distribution of sites for the city of Tübingen (Germany). [23]

At Figure 2.10, schools, universities and research institutes are blue, offices are red, bus stops are green, social places are orange and supermarkets are purple.

There are also other ways to model physical contact patterns of disease spread. For instance, Eubank et al. created bipartite contact graph to apply agent-based modelling and simulation. The graphs are generated by large-scale individual based urban traffic simulations built on actual census, land-use and population-mobility data. They find that the contact networks among people is a strongly connected small-world-like graph with a well-defined scale for the degree distribution. They find that the contact networks among people are strongly connected graphs which display small-world effect and suit for the degree distribution with wide range of scalability options. [24]

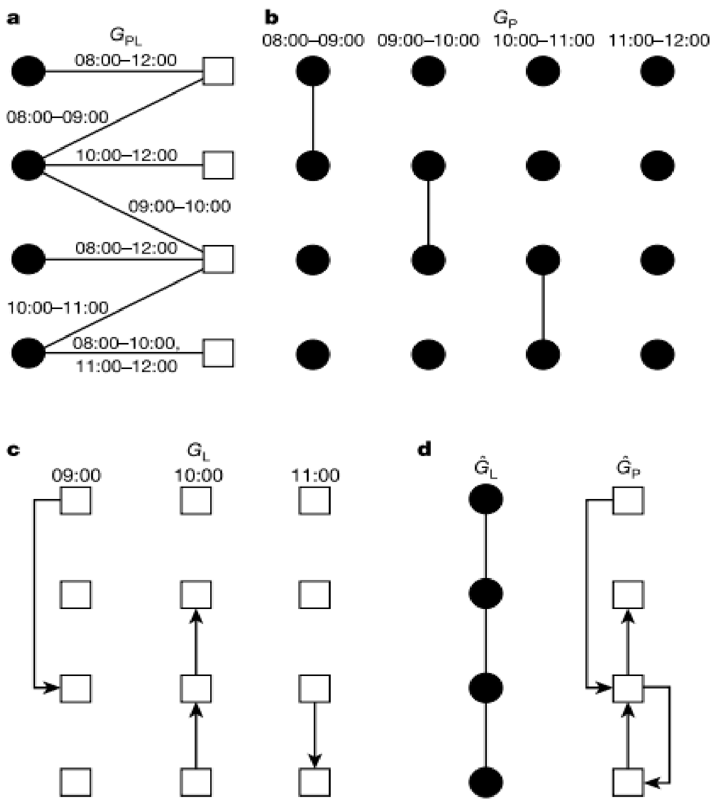


Figure 2.11: An example of a small social contact network. "a" depicts a graph $G_{PL}$ with 4 people (black circles), 4 locations (white squares). If person visits a location, there is an edge between them in the graph. Vertices labelled with available demographic or geographic information and edges contain arrival and departure times. "b", "c" are two disconnected graphs $G_P$ and $G_L$ induced by connecting vertices that were separated by exactly two edges in $G_{PL}$. "d" is static projections $\hat{G_P}$ and $\hat{G_L}$ resulting from ignoring time labels in $G_P$ and $G_L$. [24]

# 3 Main Part : Machine Learning of Stochastic Differential Equations for Infection Models using Neural Networks

This chapter constitutes the body in the study. It focuses on motivation while working on the study, methods which were attempted through the path of solving the Stochastic Differential Equations and Infection Models, data generation and preprocessing for the movements of pedestrians for infection, computational experiments in general and lastly the results.

## 3.1 Task Description and Motivation

In the study, the main problem we tackle is analyzing the dynamics of the spread of a contagious viral disease. There are multiple different relatively small problems around understanding the behavior of a viral spread in a population.

Nowadays, as the population from almost all countries, all continents we have been dealing with a pandemic called COVID-19. To our current knowledge, COVID-19 is air transmitted viral disease which stems from SARS-CoV-2 virus. In a couple of months, the disease turned into an epidemic and then shortly after pandemic throughout the globe and caused millions of deaths up till now and counting. This is only one example of how an infectious deadly virus could affect millions of lives from every corner of the world in a relatively short of time.

On one hand, relieving the effects of such disease after the spread arises must be prioritized before it turns into such a critical incident which has that much influence. On the other hand, if such an incidence occurs, having estimate of the direction of the spread could play a key role for the mitigation efforts. With the lights of these concerns, modeling such disease would contribute to solution of removing and easing the influence of such disaster. The main motivation for this study is to be able to model the spread through solving stochastic differential equations and coming up with a method to model the populations during the spread as well.

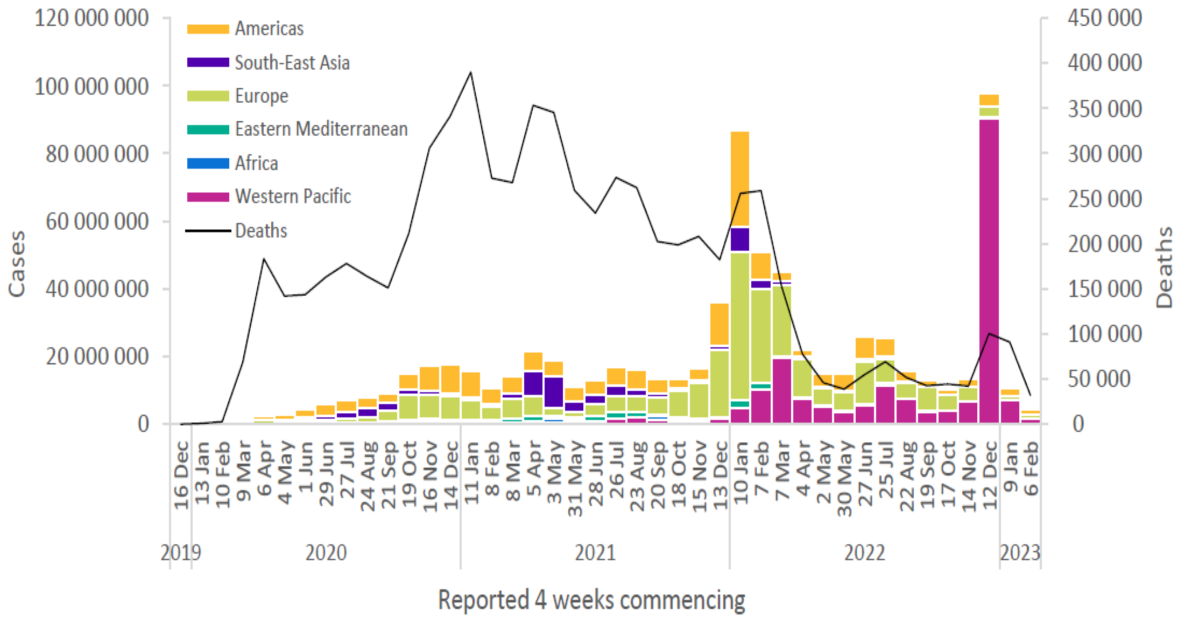


Figure 3.1: WHO Covid-19 Cases Statistics (Last updated on 08.03.2023) [25]

## 3.2 General Overview To The Process

As mentioned earlier, we identify Stochastic Differential Equations for coarse observables of fine grained particles. These observables are done by agent based simulations. We took advantage of the method proposed by Felix et. al. [19] for mathematically modeling a viral disease spread. In this study, one of the few important assumptions we make is that the distribution of disease spread could be predicted stochastically through learning the variables in the equations.

Here, the bird-eye of the process for the study could be seen. We will give a short explanation for each step at this section.

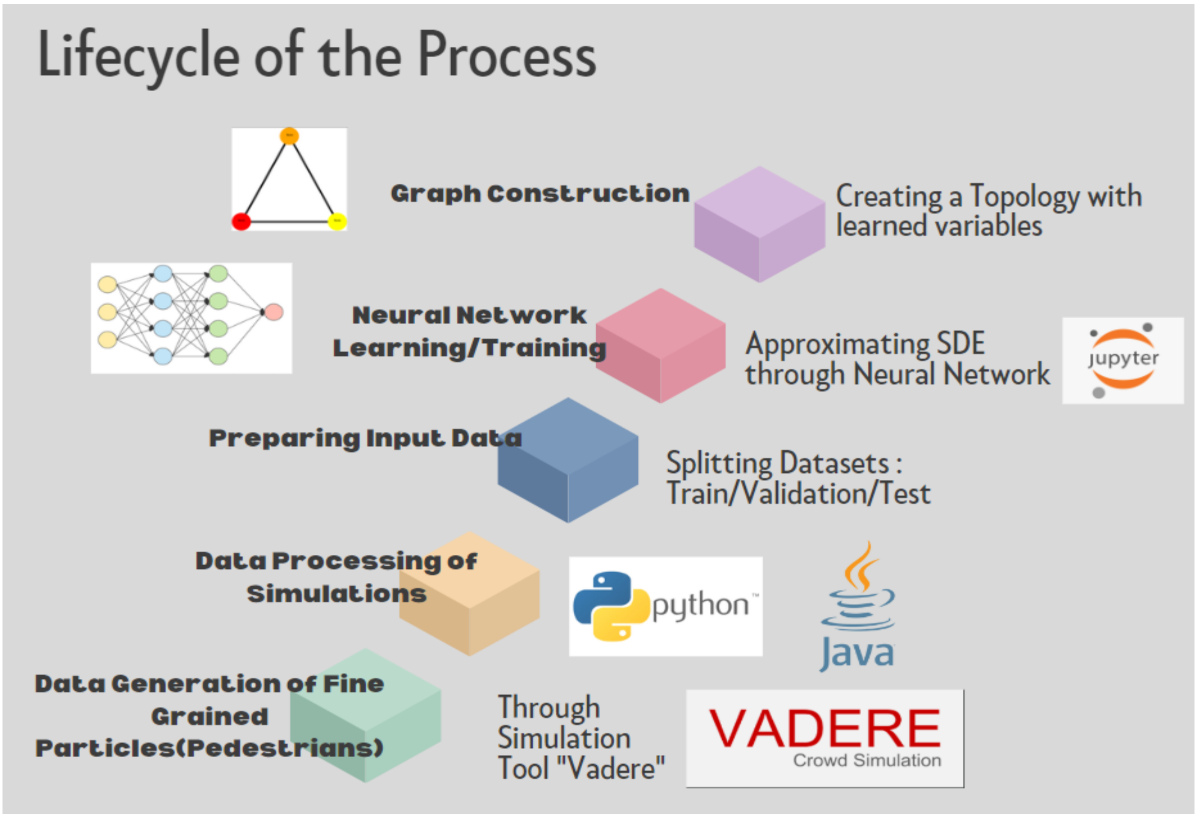
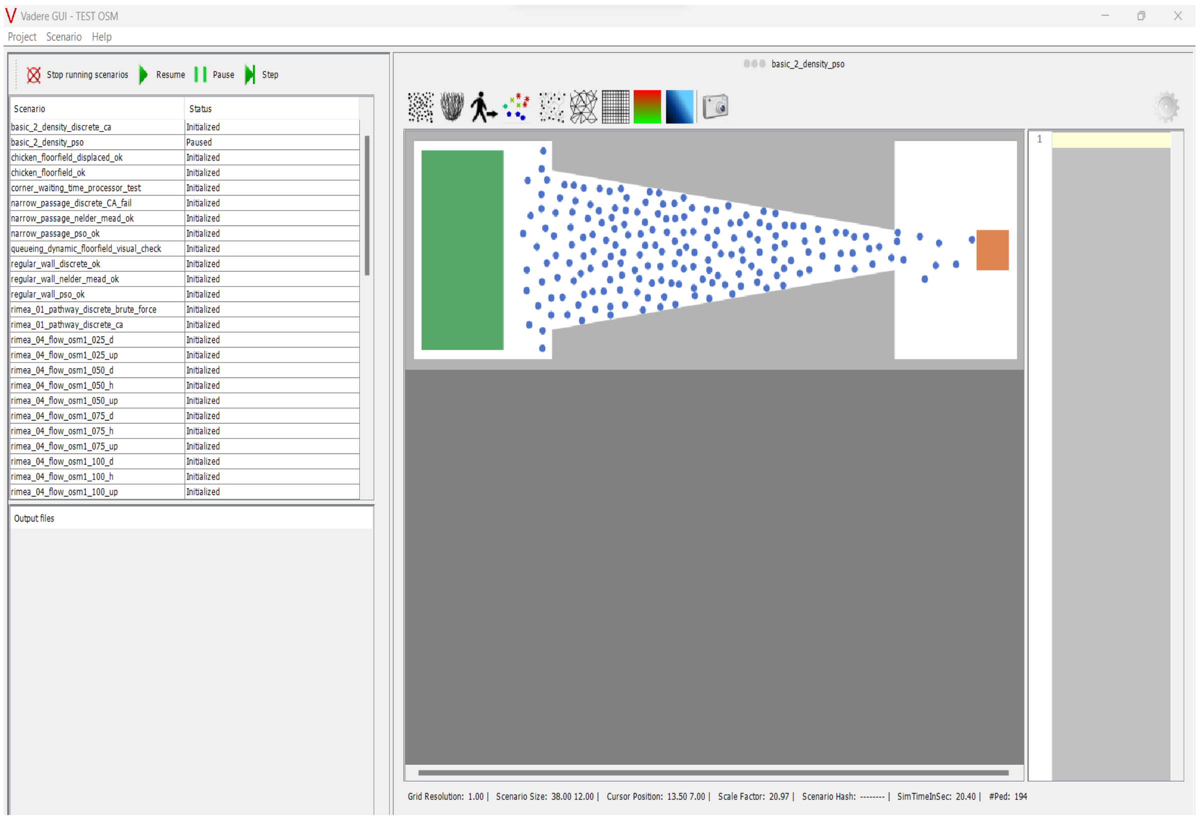

Lifecycle of the Process
Graph Construction
Creating a Topology with learned variables
Neural Network Learning/Training
Approximating SDE through Neural Network
jupyter
Preparing Input Data
Splitting Datasets : Train/Validation/Test
Data Processing of Simulations
python
Java
Data Generation of Fine Grained Particles(Pedestrians)
Through Simulation Tool "Vadere"
VADERE
Crowd Simulation

Figure 3.2: Overview of the process of the study

### 3.2.1 Data Generation of Fine Grained Particles

At this step, we generate data points for infection spread of a population in a closed area at a certin time. We used an open-source simulation software for agent-based modelling which is called 'Vadere' and is written in Java. It is a tool developed by the teams at Technical University of Munich and The Munich University of Applied Sciences. It allows us to display instantaneous rate of infected/susceptible populations in a crowd for a desired custom scenario.



| Scenario | Status |
|---|---|
| basic_2_density_discrete_ca | Initialized |
| basic_2_density_pso | Paused |
| chicken_floorfield_displaced_ok | Initialized |
| chicken_floorfield_ok | Initialized |
| corner_waiting_time_processor_test | Initialized |
| narrow_passage_discrete_CA_fail | Initialized |
| narrow_passage_nelder_mead_ok | Initialized |
| narrow_passage_pso_ok | Initialized |
| queueing_dynamic_floorfield_visual_check | Initialized |
| regular_wall_discrete_ok | Initialized |
| regular_wall_nelder_mead_ok | Initialized |
| regular_wall_pso_ok | Initialized |
| rimea_01_pathway_discrete_brute_force | Initialized |
| rimea_01_pathway_discrete_ca | Initialized |
| rimea_04_flow_osm1_025_d | Initialized |
| rimea_04_flow_osm1_025_up | Initialized |
| rimea_04_flow_osm1_050_d | Initialized |
| rimea_04_flow_osm1_050_h | Initialized |
| rimea_04_flow_osm1_050_up | Initialized |
| rimea_04_flow_osm1_075_d | Initialized |
| rimea_04_flow_osm1_075_h | Initialized |
| rimea_04_flow_osm1_075_up | Initialized |
| rimea_04_flow_osm1_100_d | Initialized |
| rimea_04_flow_osm1_100_h | Initialized |
| rimea_04_flow_osm1_100_up | Initialized |

Grid Resolution: 1.00 | Scenario Size: 38.00 12.00 | Cursor Position: 13.50 7.00 | Scale Factor: 20.97 | Scenario Hash: -------- | SimTimeInSec: 20.40 | #Ped: 194

Figure 3.3: Example View of a Scenario Run in Vadere

In our simulation case, we needed to simulate the crowd for an infection spread which means that there should be 2 different states for a pedestrian. However, the latest version of Vadere does not allow this feature. We changed the code to include `SIR Model` (Susceptible-Infectious-Recovered Model). In our environment, we focus on infection rate, and for the sake of simplicity we included `Susceptible` and `Infectious` states, did not include `Recovered` state. Then in order to have multiple scenarios, we automated the process of running a simulation with a Python script. Vadere has many options for output processors which the desired data depending on the types wanted such as timely state of a pedestrian's location, a pedestrian's target object, nearby pedestrians etc. We added the `FootStepGroupID` Processor that tracks the current infection state of a pedestrian in timely mannner.

After we decided on the scenarios which best represent the real life cases in order to generate sample datasets for our neural network training, we simulated them for 4 days interval which were run from Console for 1500 different iterations.

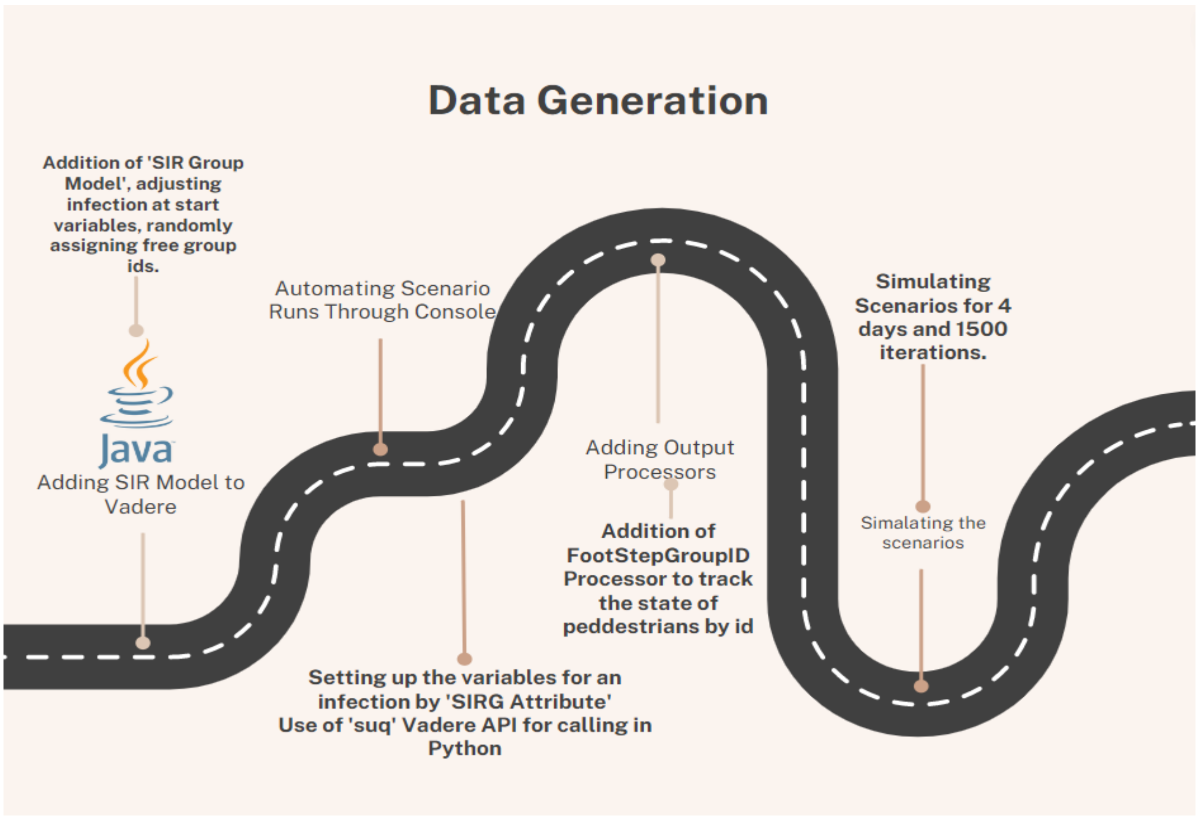

Data Generation
Addition of 'SIR Group Model', adjusting infection at start variables, randomly assigning free group ids.
Java
Adding SIR Model to Vadere
Automating Scenario Runs Through Console
Setting up the variables for an infection by 'SIRG Attribute' Use of 'suq' Vadere API for calling in Python
Adding Output Processors
Addition of FootStepGroupID Processor to track the state of peddestrians by id
Simulating Scenarios for 4 days and 1500 iterations.
Simalating the scenarios

Figure 3.4: Summary of Data Generation Process

Above, the summary of data generation process could be seen. We will explain each steps in the following sections elaborately.

### 3.2.2 Data Processing of Simulations

The data which Vadere provides, includes only the information of a pedestrian's current infection state at a certain time point. To train our data, we need to train the model with cumulative information of a crowd at a certain time point. Hence, we calculated the cumulative states of the crowd till the end of the simulation for each simulation. Then, we made it prepared for the learning part to be as training data. The training data stores 1-D information which takes the infected rate of a crowd at the start of a simulation as input data and the rate at the end as output data (labels).

Below, a short summary of data processing phase is displayed.

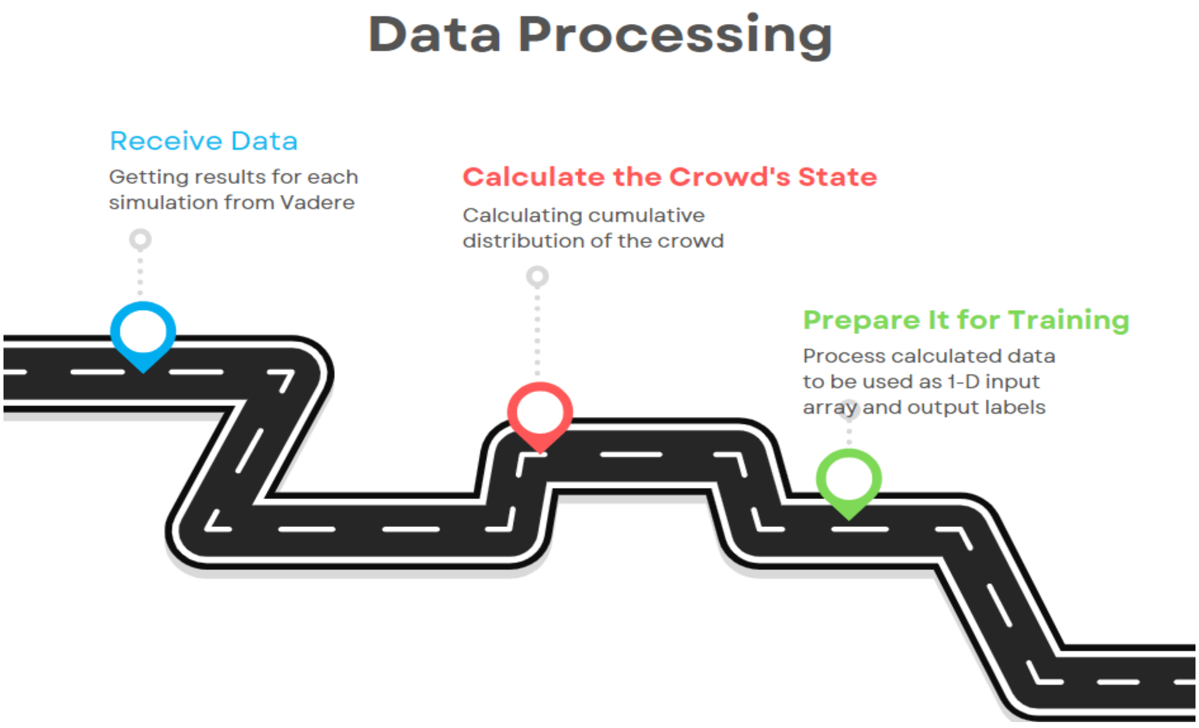
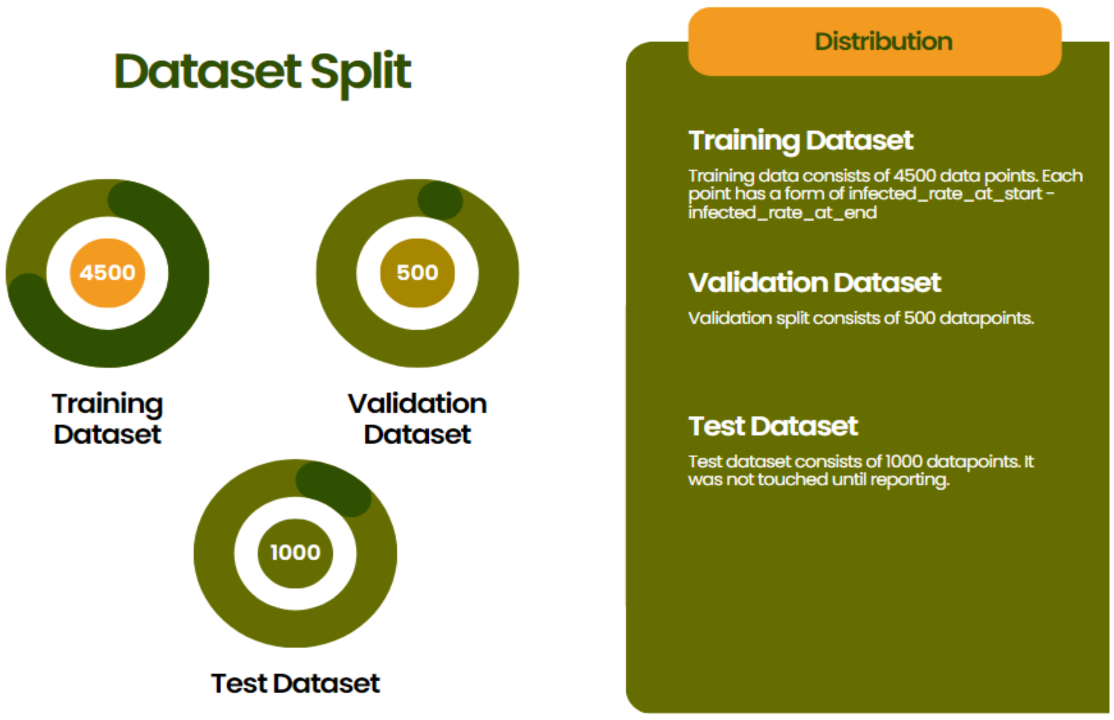

Data Processing
Receive Data
Getting results for each simulation from Vadere
Calculate the Crowd's State
Calculating cumulative distribution of the crowd
Prepare It for Training
Process calculated data to be used as 1-D input array and output labels

Figure 3.5: Summary of Data Processing

### 3.2.3 Preparing Input Data

After having the data points, Splitting data into training, validation and test sets is required. In total there are 6000 datapoints (1500 simulations with 4 days of iteration each).



Training Dataset

Training data consists of 4500 data points. Each point has a form of infected_rate_at_start - infected_rate_at_end

Validation Dataset

Validation split consists of 500 datapoints.

Test Dataset

Test dataset consists of 1000 datapoints. It was not touched until reporting.

Figure 3.6: Distribution of Datapoints

### 3.2.4 Learning Local Infections with Neural Network

After the input data was prepared to be used as training and validation data, there comes the part where the drift and diffusivity coefficients are learned. After spending much time on hyperparameter tuning, 5 hidden layers have been chosen for the structure. The network architecture was inspired by the work from Felix Dietrich from Technical University of Munich [26].

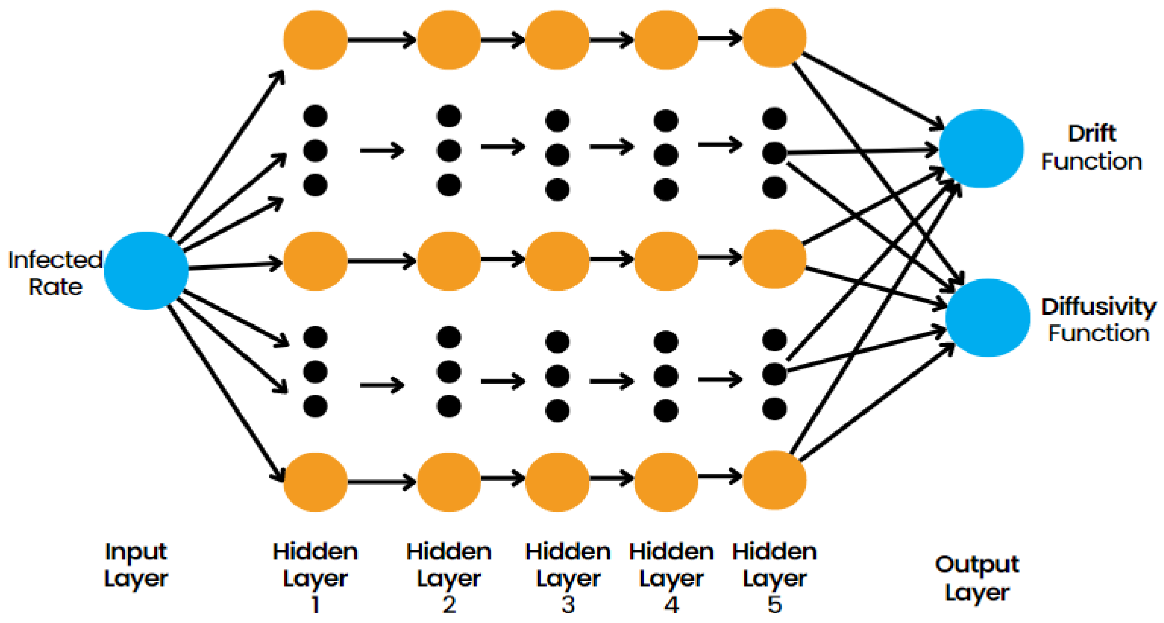


Figure 3.7: Neural Network Architecture

### 3.2.5 Graph Construction to Combine Local Infection Models

Since at this phase, having the model to estimate the next state of the infection spread is possible, we create a network topology represented with graphs which could be considered as a small village network where 3 nodes exist. Then we analyze the behavior of this network based on the movements of the crowd during a day for two different scenarios.

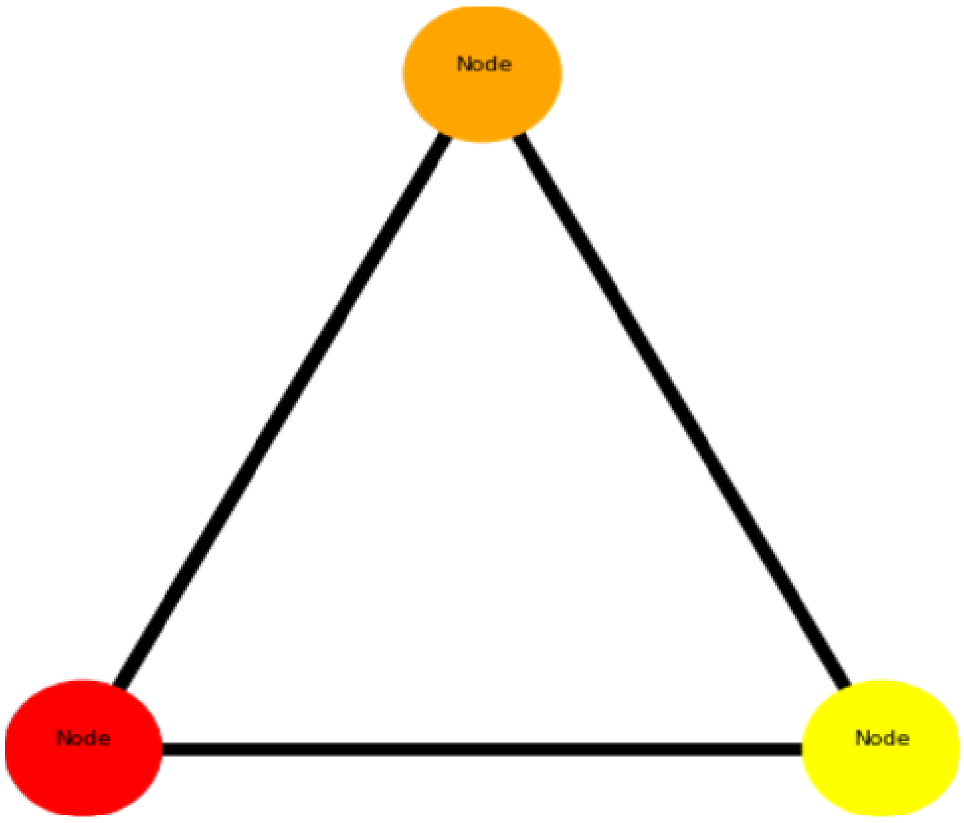


Figure 3.8: Example Visual of 3-Node Network Topology

## 3.3 Computational Experiments

Computational experiments of our study include the complete process which leads to prediction of an infection spread. These steps are as such:

- Data Generation
- Data Processing
- Input Data
- Neural Network Learning
- Results

### 3.3.1 Data Generation

At this section, what was Vadere settings, how it was simulated, selected scenarios, and the data generated through Vadere will be explained.

#### Vadere Setting

There are couple of models which decide the movements of pedestrians namely, Cellular Automation, Gradient Navigation, Social Force Model, Optimal Velocity, Reynolds Steering, Behavioral Heuristics, Biomaechanics Model, Self Cat Threat Model. These models set couple of parameters such as velocity behaviours, movement types, step length , obstacle features, target features etc. For our study, we chose `Optimal Steps Model` since it best suits our social distancing needs in pedestrians.

We use `SIR Model` to represent the states of a pedestrian and evolution of spread by time. In the Vadere version, it was not included. Thus, we added `SIR Group Model` class to the models. Since Vadere is open source, it is possible to fork from the project and build internally.

`distanceFunction` is set to 1.0. `infectionRadius` is set to 1.5. `infectionLikelihood` is set to 0.001. Other numbers such as 0.01 or less than 0.001 has been tried. However this is the one which gives more regular results based on the experiments.

Another point to mention is that since we simulate a scenario for 6000 times, it was not possible to do it through Vadere GUI. For that, we used Python script which sets the environment each time with different seeds, and other parameters through an API called `suq` to run the simulations from the console.

#### Bottleneck Scenario

Bottleneck scenario is a custom scenario made in Vadere in which we simulated a serious of times for our data generation. During the scenario, pedestrians try to reach a target object stemming from the same destination object passing a bottleneck where there intensity of pedestrians and hence high likelihood of infection. The reason why it was decided as go-to scenario is because it could represent many cases where there is high likelihood of infection such as public closed areas (transportation, cafes, schools, offices etc.)

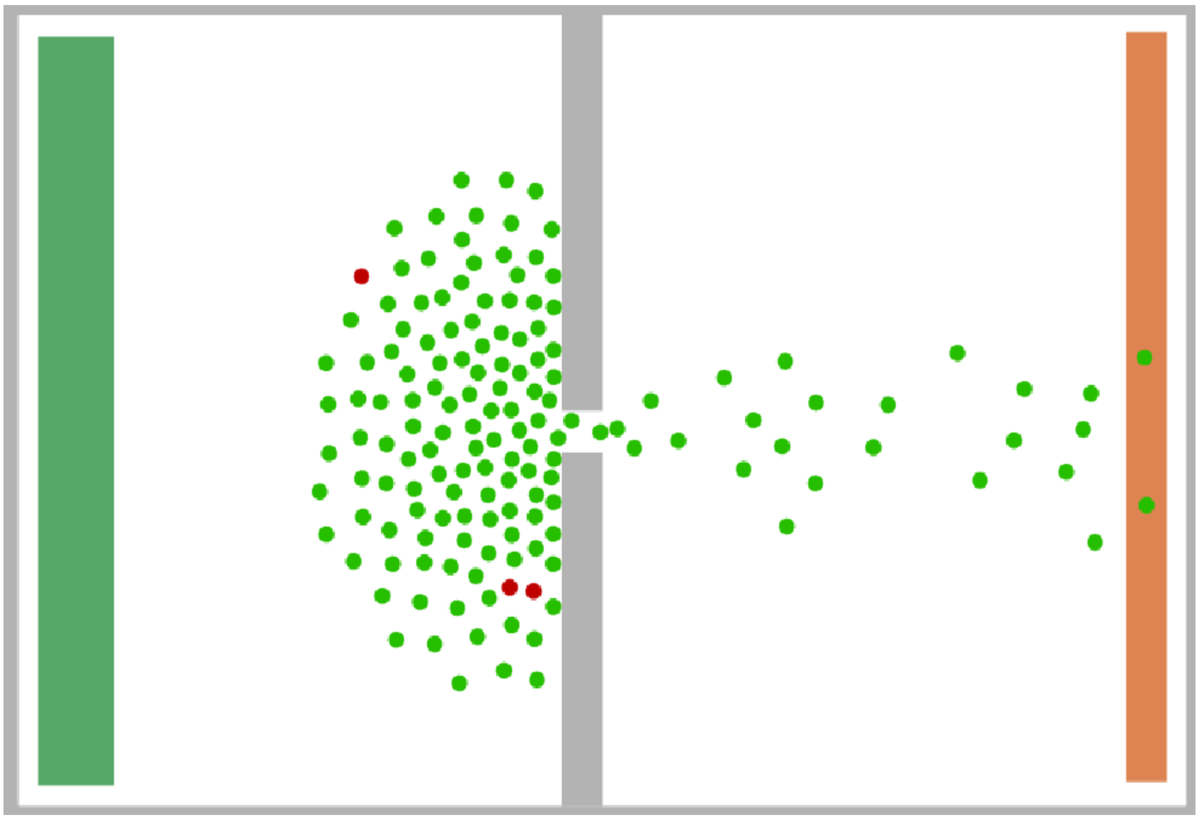
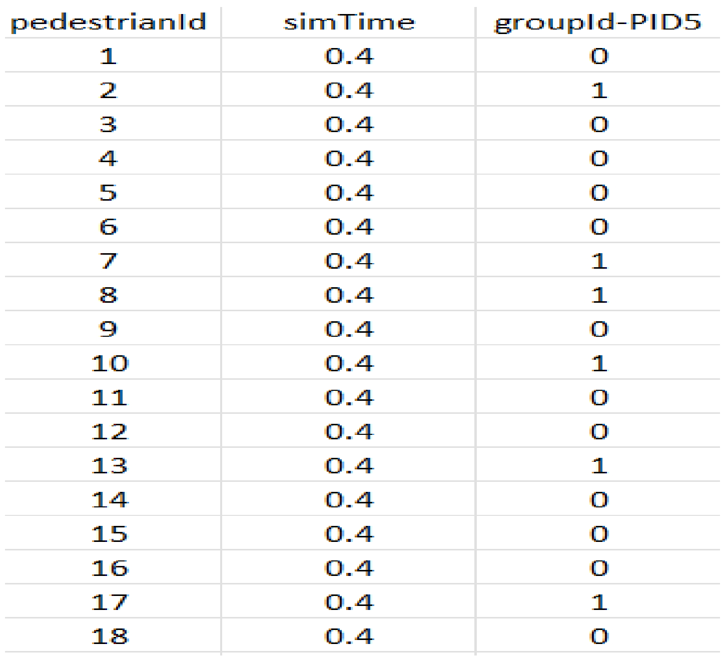


Figure 3.9: View of Pedestrians passing through Bottleneck in the Scenario in Vadere (red:Infectious, green:Susceptible Pedestrians)

We set `finishTime` parameter to 150.0 seconds, `simTimeStepLength` to 0.4, `visualizationEnabled` to `false` to simulate faster. We also set `useFixedSeed` to `true` to be able to control seed number. `fixedSeed` and `simulationSeed` are set to a different number at each simulation by the Python script we wrote. There are 200 pedestrians at each simulation, 2 of whom are infected. The infected people are set randomly by Vadere. For the source object which the pedestrians go out from, `spawnAtRandomPositions` is set to `true` to have random positions for pedestrians and `spawnAtRandomPositions` is set to `false` to have random creation and movement time of a pedestrian to the target. We simulate the scenarios for 4 days for 1500 different simulations. The total infected number at the end of a day is the one at the start of the following day until the next 4 day cycle. A point to mention here is that, the positions of the pedestrians are set randomly through Vadere. Hence the positions and ids of infected and susceptible pedestrians are not kept for the following day but only the amount of infected population. In order to have the information of pedestrians' instant SIR state, `FootStepGroupIDProcessor` has been added to the processors and other unnecessary default output files except a file called 'SIRInformation.txt' were removed, since we only need

time, pedestrian id and pedestrian state.
Below, example output for Bottleneck scenario from Vadere could be seen.



| pedestrianId | simTime | groupId-PID5 |
|---|---|---|
| 1 | 0.4 | 0 |
| 2 | 0.4 | 1 |
| 3 | 0.4 | 0 |
| 4 | 0.4 | 0 |
| 5 | 0.4 | 0 |
| 6 | 0.4 | 0 |
| 7 | 0.4 | 1 |
| 8 | 0.4 | 1 |
| 9 | 0.4 | 0 |
| 10 | 0.4 | 1 |
| 11 | 0.4 | 0 |
| 12 | 0.4 | 0 |
| 13 | 0.4 | 1 |
| 14 | 0.4 | 0 |
| 15 | 0.4 | 0 |
| 16 | 0.4 | 0 |
| 17 | 0.4 | 1 |
| 18 | 0.4 | 0 |

Figure 3.10: Snapshot of Vadere Output for Bottleneck Scenario

### Supermarket Scenario

Another point to mention is that after having data for Bottleneck scenario, we went for a couple of different scenarios including Supermartket scenario to try out the generated data and other real world scenario. However, since both the time for running a single simulation at this scenario is much beyond than expected and there was more complexity and concerns such as the size/type/decoration of a supermarket varies much we decided to go on with Bottleneck scenario as the baseline.

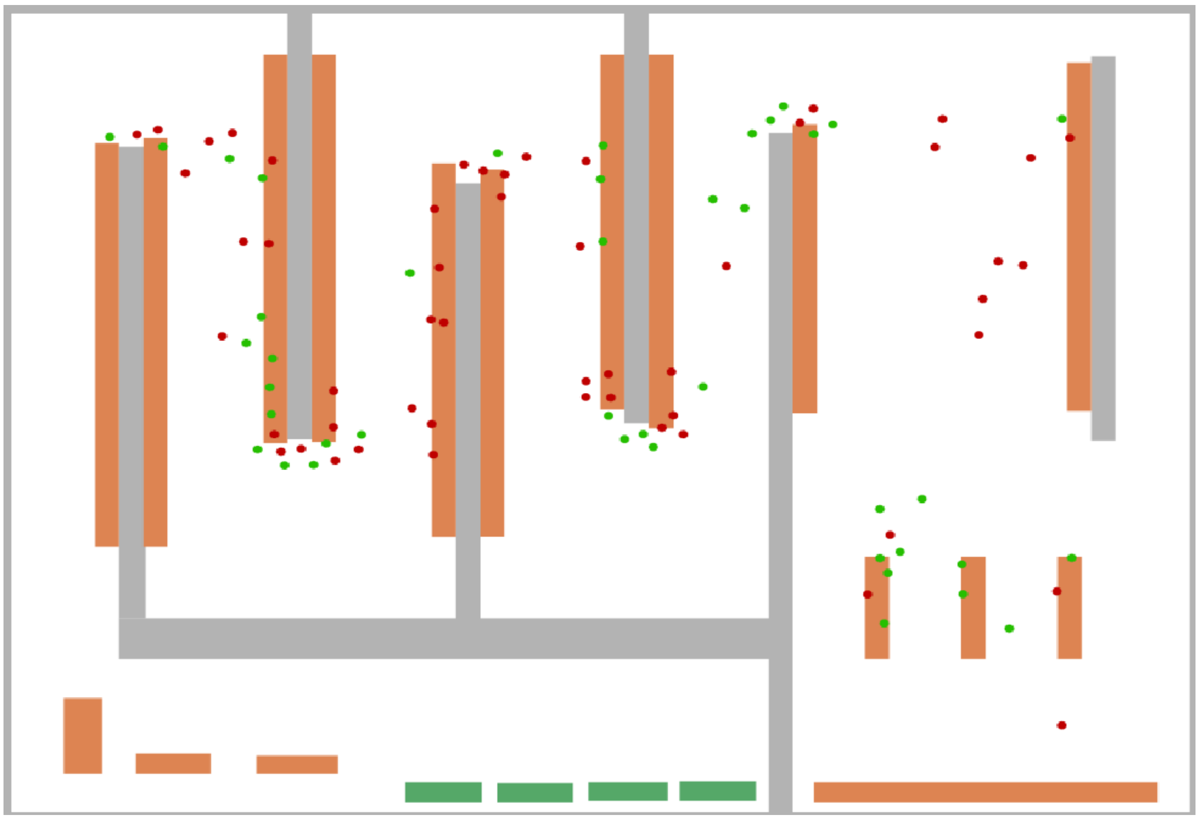


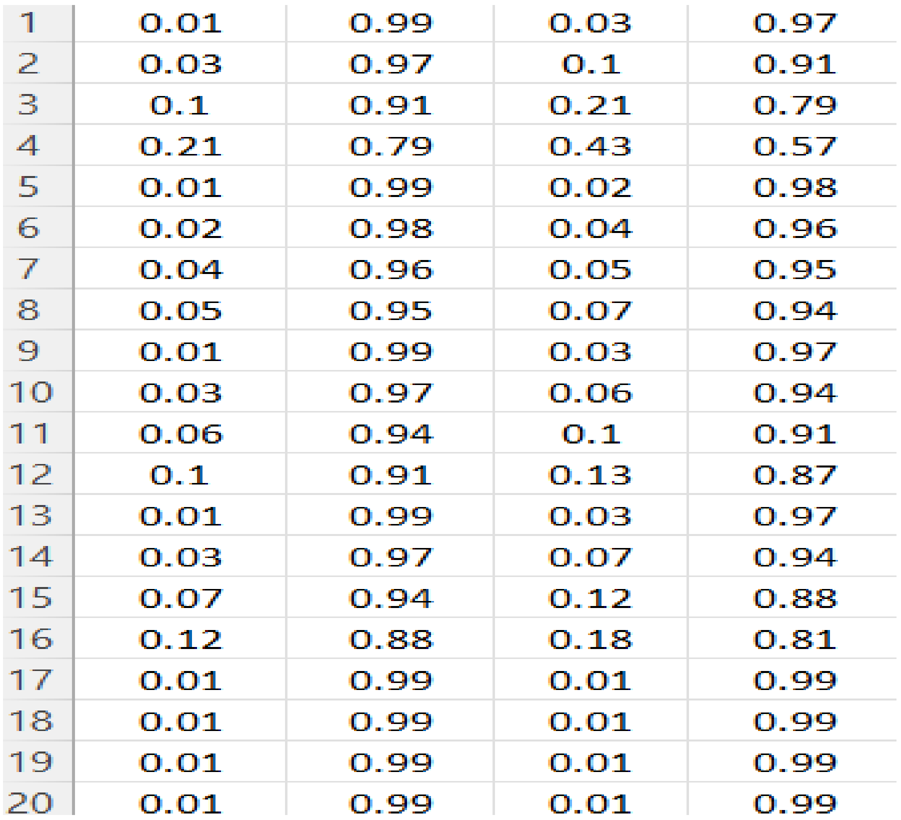
Figure 3.11: View of Pedestrians passing through Supermarket in the Scenario in Vadere (red:Infectious, green:Susceptible Pedestrians)

### 3.3.2 Data Processing

At this section, how the generated data was processed, the data which was used for training, what type of decisions have been made will be explained.

Since we have the data of a pedestrian's instant state, we calculated cumulative infected and total number of pedestrians at an instant time point. Then we calculated the normalized rates for a day. To keep the track of infection during a day, we stored the information of infected rates both at the start and at the end. The infected of the start day of a 4 day simulation phase is set to 0.01, since it gives the best results to track the evolution of infection. If it is not the first day, then the infected rate at start is the infected rate at the end of the previous day.

Below you could see the snapshot of the processed data. What is worth mentioning here is that at some days there is no change in infection rates. Below, snapshot of the processed data could be seen. Columns represent a day's infected rate at start,susceptible rate at start,infected rate at end and susceptible rate at end respectively.

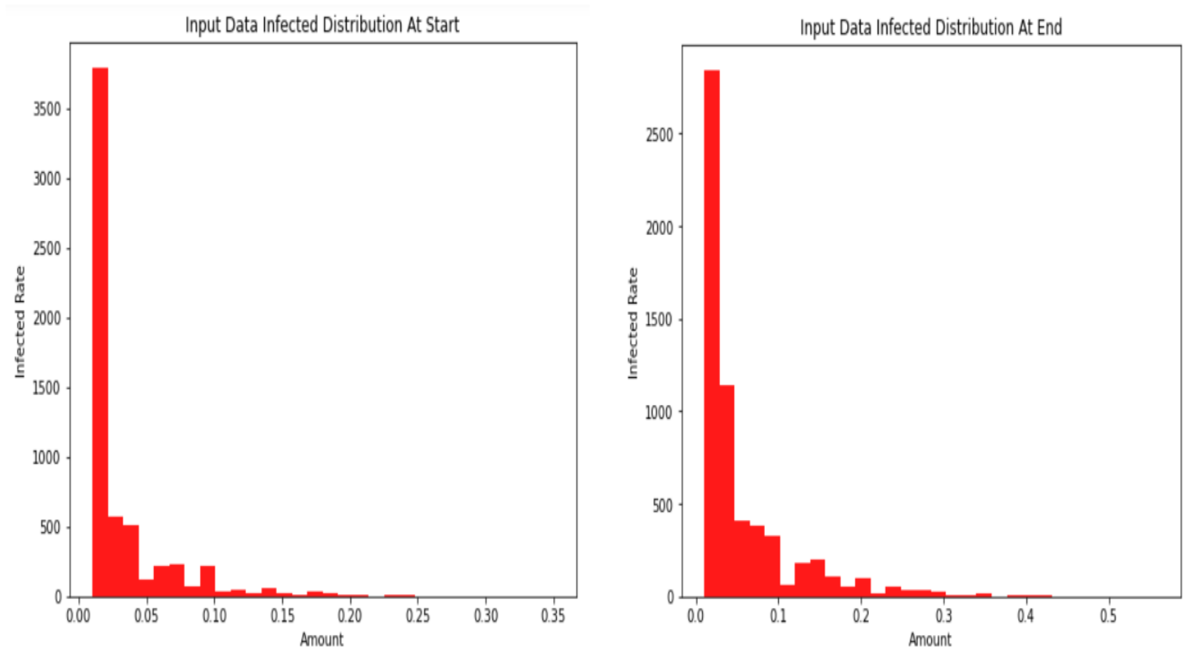


| | | | | |
|---|---|---|---|---|
| 1 | 0.01 | 0.99 | 0.03 | 0.97 |
| 2 | 0.03 | 0.97 | 0.1 | 0.91 |
| 3 | 0.1 | 0.91 | 0.21 | 0.79 |
| 4 | 0.21 | 0.79 | 0.43 | 0.57 |
| 5 | 0.01 | 0.99 | 0.02 | 0.98 |
| 6 | 0.02 | 0.98 | 0.04 | 0.96 |
| 7 | 0.04 | 0.96 | 0.05 | 0.95 |
| 8 | 0.05 | 0.95 | 0.07 | 0.94 |
| 9 | 0.01 | 0.99 | 0.03 | 0.97 |
| 10 | 0.03 | 0.97 | 0.06 | 0.94 |
| 11 | 0.06 | 0.94 | 0.1 | 0.91 |
| 12 | 0.1 | 0.91 | 0.13 | 0.87 |
| 13 | 0.01 | 0.99 | 0.03 | 0.97 |
| 14 | 0.03 | 0.97 | 0.07 | 0.94 |
| 15 | 0.07 | 0.94 | 0.12 | 0.88 |
| 16 | 0.12 | 0.88 | 0.18 | 0.81 |
| 17 | 0.01 | 0.99 | 0.01 | 0.99 |
| 18 | 0.01 | 0.99 | 0.01 | 0.99 |
| 19 | 0.01 | 0.99 | 0.01 | 0.99 |
| 20 | 0.01 | 0.99 | 0.01 | 0.99 |

Figure 3.12: View of a Part of the Processed Data

### 3.3.3 Input Data

Since we have processed data, it is now possible to have datasets for training, validation and test. There are 6000 datapoints consisting of infected/susceptible rates at start/end of the

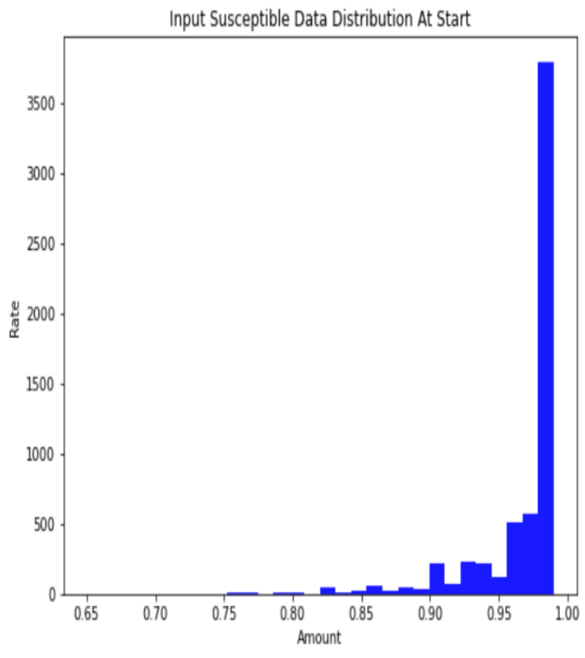


(a) At Starts (b) At Ends

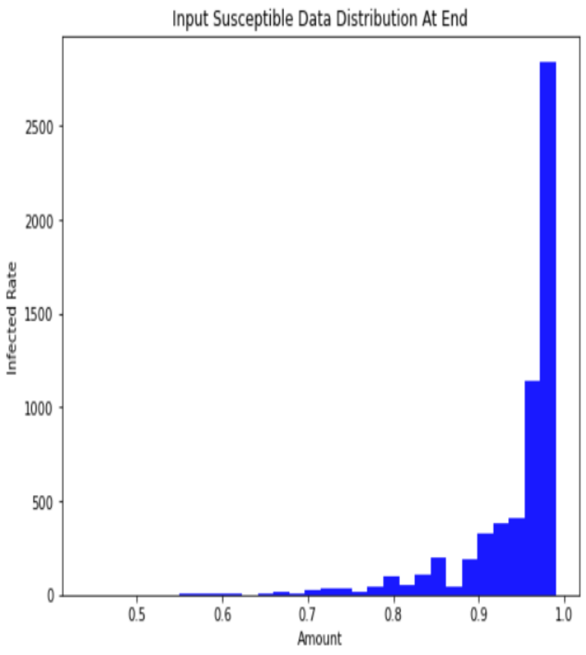
Figure 3.13: Distribution of Infected Numbers of Days

days in total. 4500 datapoints for training, 500 datapoints for validation and the rest 1000 datapoints for test datasets were used. Below, the distribution of the processed data could be seen.

Since susceptible rate at a certain day is equal to 1 - infected rate of the same day, we see that the data is symmetrical for Infected/Susceptible numbers respectively. Another attribute is that intensity of the rate for 0.01 is much higher than the others for infected. This is because a, rates of the first days start at 0.01 and at some cases there is no change in rates.



(a) At Starts



(b) At Ends

Figure 3.14: Distribution of Susceptible Numbers of Days

### 3.3.4 Learning Local Infections with Neural Network

Machine learning of the SDE variables was achieved through Neural Network Training. After processed data is ready to be used as training and validation datasets, the next task is to set up the model architecture for training. At this section, Loss Function, Activation Function, Optimizer, other hyperparameters and tuning process, model architecture, libraries and tools used, which type of experiments conducted will be explained.

#### Loss Function

As we mentioned before, to solve the drift and diffusivity function of SDE, we use a specific type of Loss Function which was inspired by Euler-Maruyama scheme.

#### Libraries and Tools Used

There have been a couple of libraries and tools used during NN training and this study in general. It was written in Python. For model architecture, `TensorFlow` and `Keras` were used. TensorFlow is a free and open-source library for machine learning and artificial intelligence first released by Google Brain team. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow [27]. It was first published as part of TensorFlow, then became a separate library. In our study, Keras was used for model creation and training. Its internal functions such as `fit, history` were used. `tf.keras.Model` is the model we used in training. During experiments, `Anaconda Navigator` distribution, `Jupyter Notebook` were used for coding NN learning. `Anaconda Navigator` is an application allowing to managing applications and packages, environments through its graphical user interface (GUI). `Jupyter Notebook` is a service to execute Python, Julia or R codes.

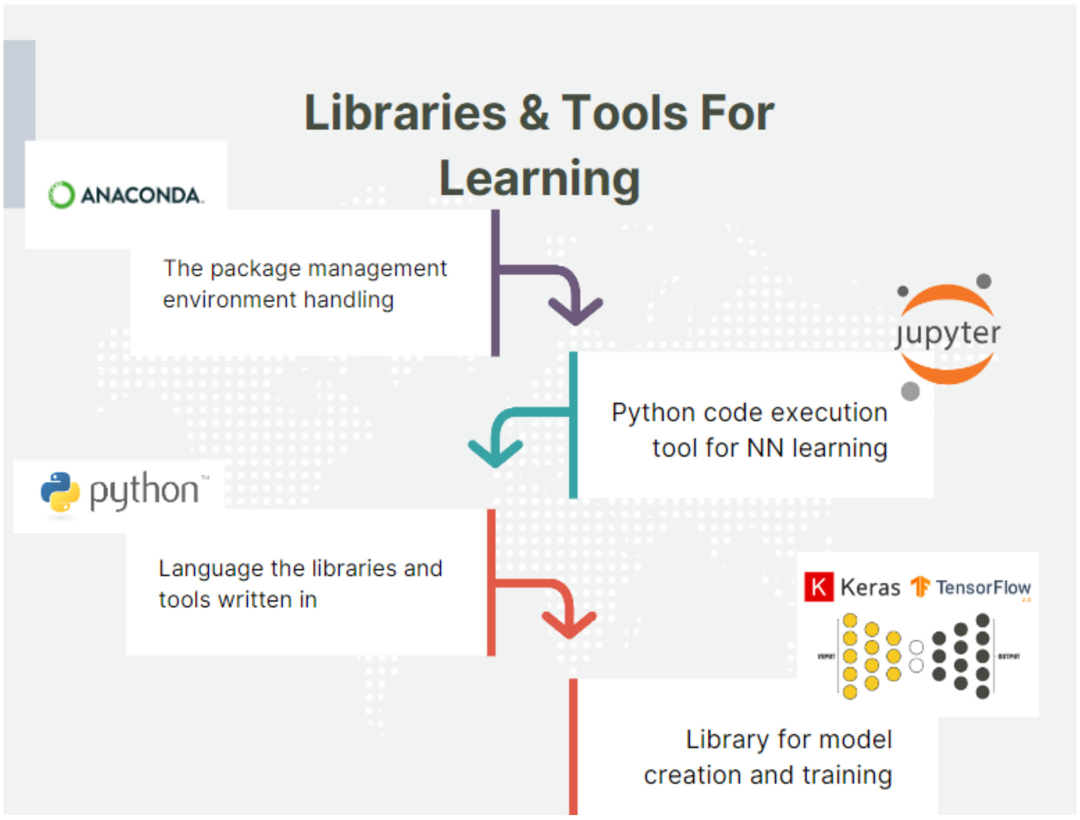

Libraries & Tools For Learning
ANACONDA
The package management environment handling
jupyter
Python code execution tool for NN learning
python
Language the libraries and tools written in
Keras
TensorFlow
Library for model creation and training

Figure 3.15: Libraries and Tools Used During Learning

### Activation Function

Activation functions are crucial parts of neural networks which transform the input signals to output signals. It usually gives nonlinearity to the network. Nonlinearity is the essential feature of neural networks since the network learns through nonlinear relations. On one hand, there is linear activation function, namely Identity Function or Linear Function. It is simply the function of $f(x) = x$. It does not add nonlinearity hence complexity to the network. On the other hand, nonlinear activation functions are the ones which add complexity to the network. There are a couple of examples to that. In this study, during training phase, most possible examples have been tried such as relu,elu,gelu,selu,sigmoid,softmax,softplus,softsign,swish,tanh. After seeing the results for each function, 'tanh' has been chosen which gave the best results for our case. Tanh pushes the output values to be between [-1, 1].

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

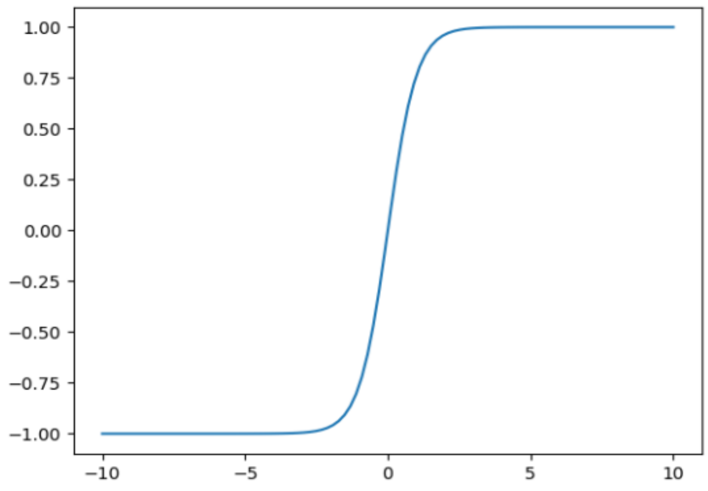


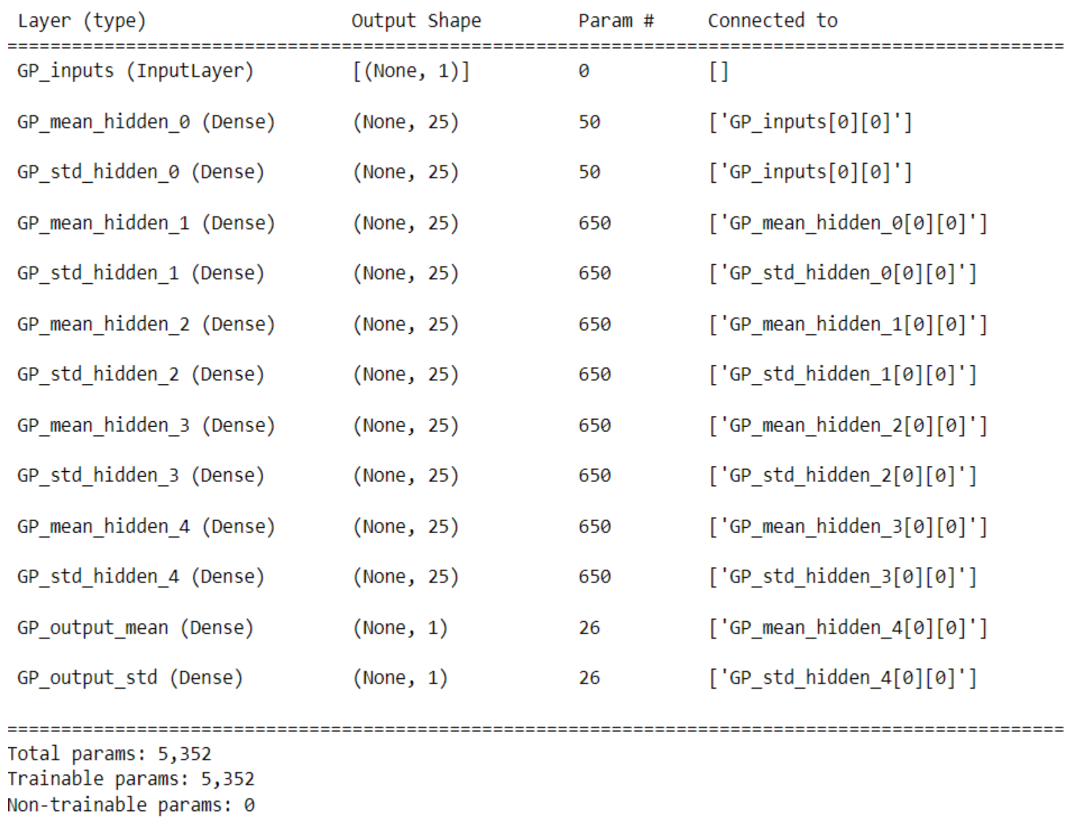
Figure 3.16: Tanh Activation Function

**Hyperparameters**

Hyperparameter tuning, is the process of searching for optimal parameters which give the best results for training. After couple of experiments, the parameters we went for are below:

| | |
|---|---|
| Batch Size | 512 |
| Epoch Number | 40 |
| Optimizer | Adamax |
| Number of Layers | 13 |
| Neurons Per Layer | 25 |

Batch size is the number of samples which will be sent at one time through the network. Depending on the network, batch size of 32 is common. In our network, higher batch size worked better. Epoch number is the one full cycle of the data used for training. In order for the network to learn, more than one time of data is used. In our network, we saw not much significant change in results after 40 epochs. Optimizer is another significant hyperparameter which affects the results, the speed of the convergence. It is an algorithm to search for optimal results for minimizing error function. Depending on the type of data and network structure, there are a couple of options for optimizers. We used `Adamax` optimizer with `0.01 learning rate`.



```
 Layer (type)                Output Shape         Param #     Connected to
==================================================================================================
 GP_inputs (InputLayer)      [(None, 1)]          0           []

 GP_mean_hidden_0 (Dense)    (None, 25)           50          ['GP_inputs[0][0]']

 GP_std_hidden_0 (Dense)     (None, 25)           50          ['GP_inputs[0][0]']

 GP_mean_hidden_1 (Dense)    (None, 25)           650         ['GP_mean_hidden_0[0][0]']

 GP_std_hidden_1 (Dense)     (None, 25)           650         ['GP_std_hidden_0[0][0]']

 GP_mean_hidden_2 (Dense)    (None, 25)           650         ['GP_mean_hidden_1[0][0]']

 GP_std_hidden_2 (Dense)     (None, 25)           650         ['GP_std_hidden_1[0][0]']

 GP_mean_hidden_3 (Dense)    (None, 25)           650         ['GP_mean_hidden_2[0][0]']

 GP_std_hidden_3 (Dense)     (None, 25)           650         ['GP_std_hidden_2[0][0]']

 GP_mean_hidden_4 (Dense)    (None, 25)           650         ['GP_mean_hidden_3[0][0]']

 GP_std_hidden_4 (Dense)     (None, 25)           650         ['GP_std_hidden_3[0][0]']

 GP_output_mean (Dense)      (None, 1)            26          ['GP_mean_hidden_4[0][0]']

 GP_output_std (Dense)       (None, 1)            26          ['GP_std_hidden_4[0][0]']

==================================================================================================
Total params: 5,352
Trainable params: 5,352
Non-trainable params: 0
```

Figure 3.17: Neural Network Architecture

**Neural Network Structure**

we have 5352 parameters in total. Here we have 13 layers including input and output layers. The model is inspired by Gaussian Processes having a mean and a standard deviation values as output. In our case, they are drift and diffusivity coefficients.

**Experiments**

During hyperparameter tuning and training, we first started with using 2-D input data which consists of instant infected and susceptible rates for start and end of a day. Susceptible rate is simply $1 - infectedrate$. Hence it should add up to 1.0. After couple of trials, we noticed that the model attempts to learn both variables and the rates don't add up to 1.0. We then decided to move on 1-D input data which consists only of infected rate for start and end of a day. We had better results this way. NN learns the variable and loss values decreased dramatically.

### 3.3.5 Results

After training, we have an average loss of -1.14 after 40 epochs.

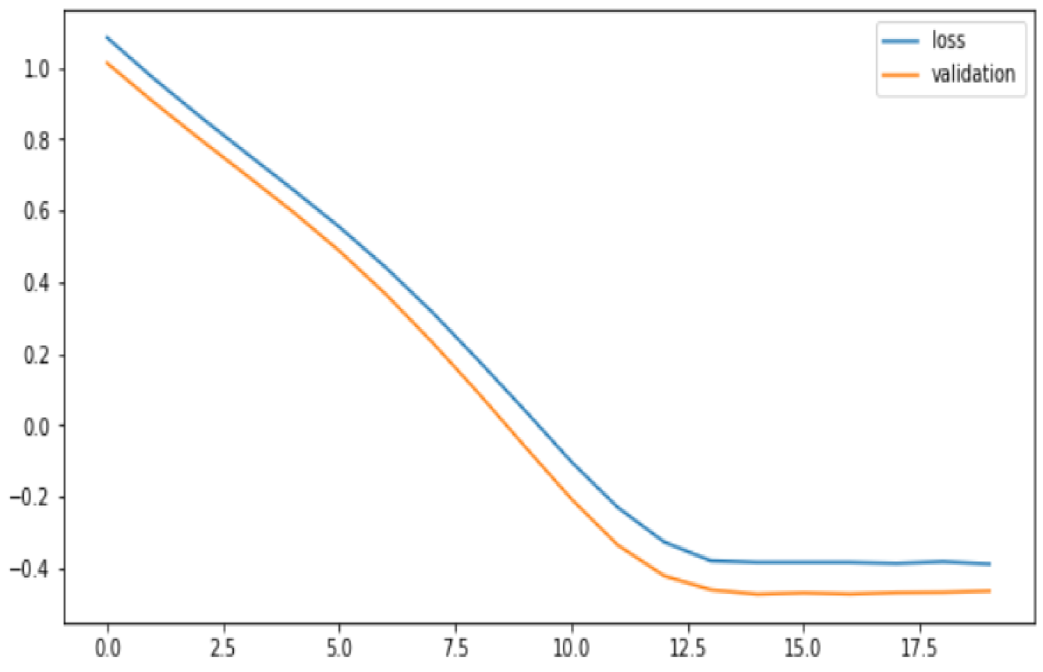


Figure 3.18: Average Loss

After learning the variables, we created network paths for our model. This is where we used the test datasets including 1000 datapoints. Since test datasets consist of the simulation information of 250 different iterations for 4 days, we got estimates from our model for the same setting.

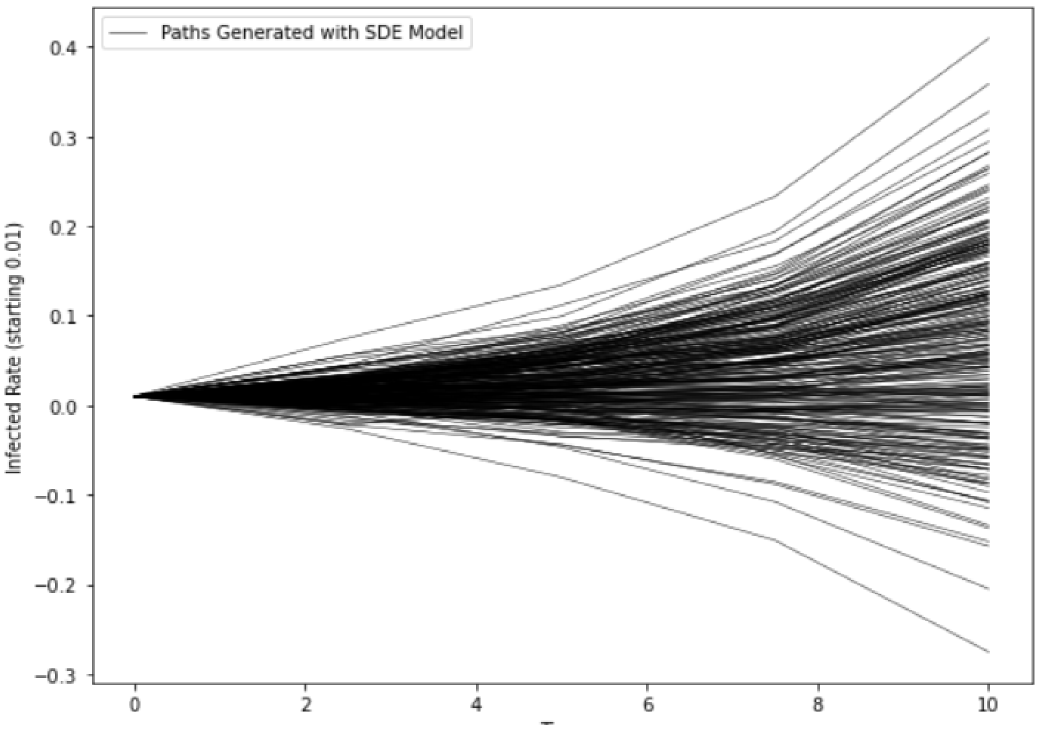


Figure 3.19: Generated Path

Above, generated paths from our learned model could be seen. One thing worth mentioning is that generated paths fluctuate between 0 and 1. Whereas, our test data points are between 0 and 1. Hence on the second visual, the comparison between generated paths from the model and test cases for the values between 0 and 1 could be seen.

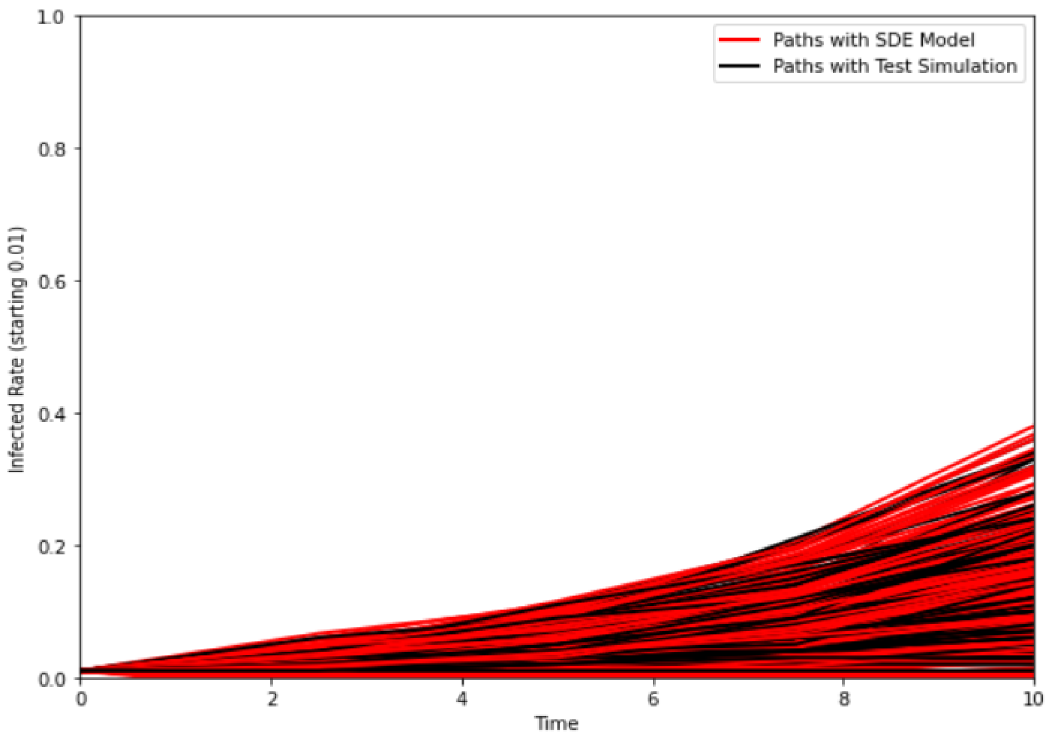
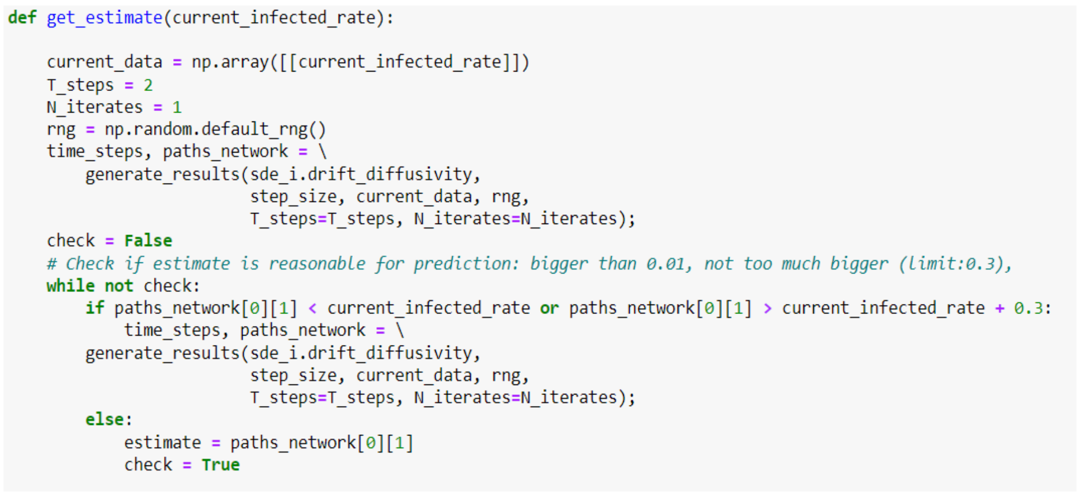


Figure 3.20: True vs Generated Path

## 3.4 Graph Construction to Combine Local Infection Models

Since we have the model of learned SDE variables which predict the next state of infected rate of the crowd at an instant time given the current infected rate, we could use it to estimate spread evolution for a region where there is a crowd living. In order to achieve that, we created a small network topology, a small town network to put it differently. Our network is represented by a graph, consists of 3 nodes. These nodes represent the high likely places a person could visit during a day. The nodes are decided as Transportation, School or Work, and Market. At each iteration for a scenario, first there is initial condition, afterward, movement between each tuple happens, then the crowd has new infected rate at each node based on the new number of populations at nodes. This new infected rate is received by `get_estimate` function. There are a couple of limitations applied for prediction. The first one is to have a rate of more than 0.01 which is our infected rate at start and a prediction could not be below that. This limitation fixes the problem of having negative values as well. Another one is to have a rate not much different than the current state since it should be closer to real-time scenarios this way.



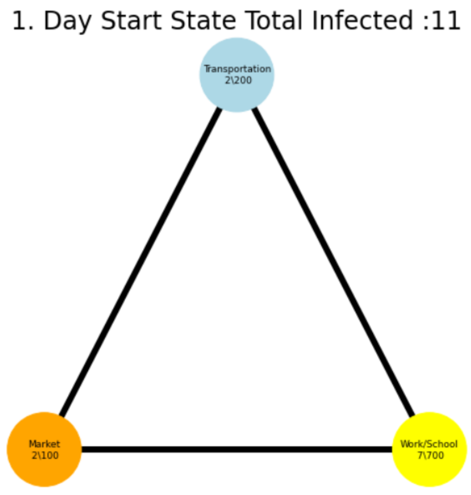
```
def get_estimate(current_infected_rate):

    current_data = np.array([[current_infected_rate]])
    T_steps = 2
    N_iterates = 1
    rng = np.random.default_rng()
    time_steps, paths_network = \
        generate_results(sde_i.drift_diffusivity,
                         step_size, current_data, rng,
                         T_steps=T_steps, N_iterates=N_iterates);
    check = False
    # Check if estimate is reasonable for prediction: bigger than 0.01, not too much bigger (limit:0.3),
    while not check:
        if paths_network[0][1] < current_infected_rate or paths_network[0][1] > current_infected_rate + 0.3:
            time_steps, paths_network = \
        generate_results(sde_i.drift_diffusivity,
                         step_size, current_data, rng,
                         T_steps=T_steps, N_iterates=N_iterates);
        else:
            estimate = paths_network[0][1]
            check = True
```

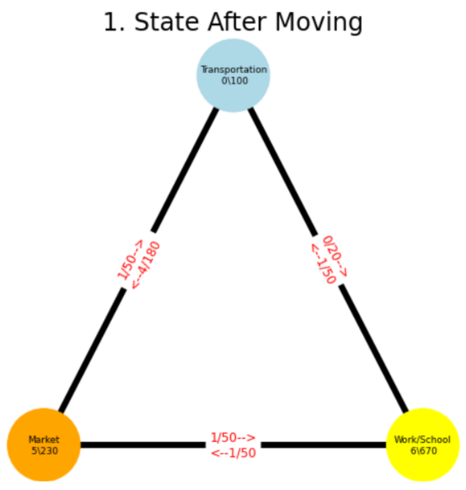
Figure 3.21: get_estimate Function

### Scenario 1

This scenario is simulation of movement and infection in a small place during daytime consisting of supermarket, school/work, and transportation nodes. At initial condition, there are 200 people at transportation node, 100 people at market node and 700 people at work/school node with 0.01 infection rate each. At the second iteration, the movement of the people between tuples is displayed. From market 50 people go to transportation, 50 people go to work. From transportation 180 people go to market and 20 people go to work. Lastly, from work 50 people go to transportation, 50 people go to market. chosen to be reasonable and consistent with real daytime scenarios. During the daytime for a person, probabilistically there is not much time spent at market than transportation or work. People use transportation either to reach work or the market and vice versa. Hence we picked these numbers. At the end of these movements, the new population becomes 100 for transport, 230 for market and 670 for work. Then, we predict new infected numbers at each node through `get_estimate` function. Since the function gives results stochastically, at each run we could receive different numbers.



(a) Initial Iteration



(b) Movement Iteration

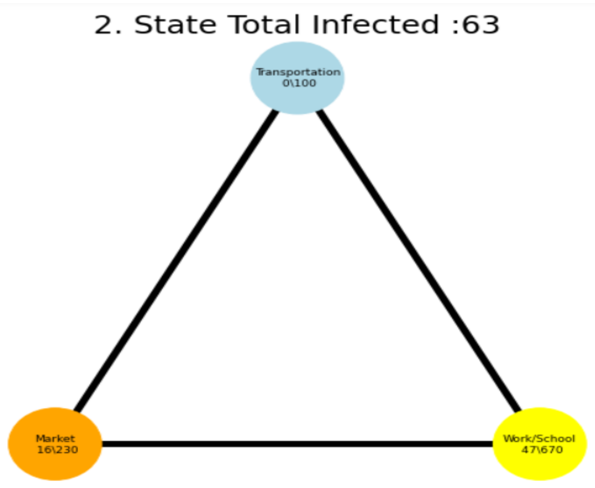


Figure 3.22: Scenario 1 after infection

**Stochastic Matrix**

The stochastic process we used in our modeling is Markov process or Markov chain in other words. Markov process is simply a stochastic model describing events which the probability of each event depends only on the state by the the previous event. As we showed, we get results from our model stochastically. At the scenario we mentioned, we made assumptions about the number of people at each node. However, only the probabilities of movements would be enough to have the distribution of the infected people at each through iterating Stochastic Matrix infinitely many times. Stochastic Matrix is a square matrix used to describe transitions of Markov chain with probabilities.

| | Market | Work/School | Transportation |
|---|---|---|---|
| Market | 0.0 | 0.5 | 0.5 |
| Work/School | 0.07 | 0.86 | 0.07 |
| Transportation | 0.9 | 0.1 | 0.0 |

# 4 Conclusion & Future Work

## 4.1 Conclusion

In the study, we model the coarse-grained particles in dynamical crowd setting. We use agent based modelling simulation systems. For that, during computational experiments Vadere is used. It is strong for some aspects that you have the flexibility to simulate pedestrians and have instant infection state of a pedestrian. We had the requirement of recurring times of simulations to see the evolution of the spread. However, the destination and target locations of pedestrians for the next simulations are not controllable by the user. This was a limitation for us. We then learned the variables of stochastic differential equations though neural networks with Euler-Maruyama inspired loss function. The data we generated through Vadere had more simulations which have no change than expected. This situation affected the distribution of the data. All in all, it was achieved to have learning SDE variables with consistent results with the generated results. Then we managed to construct a graph network which shows local infection spread in a small town scenario.

## 4.2 Future Work

The possible new directions and methodologies for the purpose of the study vary.

- Other software tools which allow more flexibility could be used.
- New scenarios which include more buildings or complex cases could be included.
- Since it is likely to have similar behaviors of infection spread for the same type of buildings (schools, restaurants, supermarkets etc.) in a region, it is possible to work on bigger graphs which represent more populations such as cities, countries etc.
- During our study, working with real world data was not possible due to the lack of available data. In the future, such data which represent recent cases as well could be used.

# Bibliography

[1] C. S. Currie, J. W. Fowler, K. Kotiadis, T. Monks, B. S. Onggo, D. A. Robertson, and A. A. Tako. "How simulation modelling can help reduce the impact of COVID-19". In: *Journal of Simulation* 14.2 (2020), pp. 83–97. DOI: 10.1080/17477778.2020.1751570. eprint: https://doi.org/10.1080/17477778.2020.1751570. URL: https://doi.org/10.1080/17477778.2020.1751570.

[2] W. O. Kermack and À. G. Mckendrick. "A contribution to the mathematical theory of epidemics". In: *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences* 115 (1927), pp. 700–721.

[3] S. D. Group. *System Dynamics*. URL: https://www.uib.no/en/rg/dynamics/39282/what-system-dynamics (visited on 03/20/2023).

[4] D.-L. Ngo-Hoang. "The three methods in simulation modeling [Chapter 2. The three methods in simulation modeling]". In: (May 2020). DOI: 10.13140/RG.2.2.29143.09125.

[5] W. Foundation. *Discrete Event Simulation*. URL: https://en.wikipedia.org/wiki/Discrete-event_simulation (visited on 03/20/2023).

[6] E. Bonabeau. "Agent-based modeling: Methods and techniques for simulating human systems". In: *Proceedings of the National Academy of Sciences* 99.suppl_3 (2002), pp. 7280–7287. DOI: 10.1073/pnas.082080899. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.082080899. URL: https://www.pnas.org/doi/abs/10.1073/pnas.082080899.

[7] W. Foundation. *Agent Based Model*. URL: https://en.wikipedia.org/wiki/Agent-based_model (visited on 03/20/2023).

[8] C. Macal and M. North. "Agent-based modeling and simulation". In: Dec. 2009. DOI: 10.1109/WSC.2009.5429318.

[9] B. Kleinmeier, B. Zönnchen, M. Gödel, and G. Köster. "Vadere: An open-source simulation framework to promote interdisciplinary understanding". In: *CoRR* abs/1907.09520 (2019). arXiv: 1907.09520. URL: http://arxiv.org/abs/1907.09520.

[10] M. J. Seitz and G. Köster. "Natural discretization of pedestrian movement in continuous space." In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 86 4 Pt 2 (2012), p. 046108.

[11] F. Dietrich and G. Köster. "Gradient navigation model for pedestrian dynamics". In: *Physical Review E* 89.6 (June 2014). DOI: 10.1103/physreve.89.062801. URL: https://doi.org/10.1103%2Fphysreve.89.062801.

[12] D. Helbing and P. Molnár. “Social force model for pedestrian dynamics”. In: *Physical Review E* 51.5 (May 1995), pp. 4282–4286. DOI: 10.1103/physreve.51.4282. URL: https://doi.org/10.1103%2Fphysreve.51.4282.

[13] C. M. Mayr and G. Köster. “Social distancing with the Optimal Steps Model”. In: *CoRR* abs/2007.01634 (2020). arXiv: 2007.01634. URL: https://arxiv.org/abs/2007.01634.

[14] I. von Sivers and G. Köster. “Dynamic stride length adaptation according to utility and personal space”. In: *Transportation Research Part B: Methodological* 74 (Apr. 2015), pp. 104–117. DOI: 10.1016/j.trb.2015.01.009. URL: https://doi.org/10.1016%2Fj.trb.2015.01.009.

[15] K. Kermack. *Proceedings of the Royal society of London. VOL115*. Royal Society of London, 1927.

[16] N. Bacaër. “McKendrick and Kermack on epidemic modelling (1926–1927)”. In: *A Short History of Mathematical Population Dynamics*. London: Springer London, 2011, pp. 89–96. ISBN: 978-0-85729-115-8. DOI: 10.1007/978-0-85729-115-8_16. URL: https://doi.org/10.1007/978-0-85729-115-8_16.

[17] W. H. Organization. *Coivid Cases*. URL: https://covid19.who.int/region/euro/country/de (visited on 03/20/2023).

[18] N. Van Kampen. “Stochastic differential equations”. In: *Physics Reports* 24.3 (1976), pp. 171–228. ISSN: 0370-1573. DOI: https://doi.org/10.1016/0370-1573(76)90029-6. URL: https://www.sciencedirect.com/science/article/pii/0370157376900296.

[19] F. Dietrich, A. Makeev, G. Kevrekidis, N. Evangelou, T. Bertalan, S. Reich, and I. Kevrekidis. “Learning effective stochastic differential equations from microscopic simulations: Linking stochastic numerics to deep learning”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 33 (Feb. 2023), p. 023121. DOI: 10.1063/5.0113632.

[20] S. Walczak and N. Cerpa. “Artificial Neural Networks”. In: Dec. 2003, pp. 631–645. ISBN: 9780122274107. DOI: 10.1016/B0-12-227410-5/00837-1.

[21] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65 (1958). Place: US Publisher: American Psychological Association, pp. 386–408. ISSN: 1939-1471(Electronic),0033-295X(Print). DOI: 10.1037/h0042519.

[22] W. S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[23] L. Lorch, H. Kremer, W. Trouleau, S. Tsirtsis, A. Szanto, B. Schölkopf, and M. Gomez-Rodriguez. *Quantifying the Effects of Contact Tracing, Testing, and Containment Measures in the Presence of Infection Hotspots*. 2020. DOI: 10.48550/ARXIV.2004.07641. URL: https://arxiv.org/abs/2004.07641.

[24] S. Eubank, H. Guclu, V. S. Anil Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang. "Modelling disease outbreaks in realistic urban social networks". In: *Nature* 429.6988 (May 2004), pp. 180–184. ISSN: 1476-4687. DOI: 10.1038/nature02541. URL: https://doi.org/10.1038/nature02541.

[25] W. H. Organisation. *Weekly epidemiological update on COVID-19 - 8 March 2023*. URL: https://www.who.int/publications/m/item/weekly-epidemiological-update-on-covid-19---8-march-2023 (visited on 03/20/2023).

[26] F. Dietrich. *Felix Dietrich Gitlab Repositories*. URL: https://gitlab.com/felix.dietrich/sde-identification (visited on 03/20/2023).

[27] Keras. *Keras About*. URL: https://keras.io/about/ (visited on 03/20/2023).