



Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

**Coarse-Graining Crowd Dynamics
Simulations with Stochastic Differential
Equations**

Kaili Wang





Computational Science and Engineering (International Master's Program)

Technische Universität München

Master's Thesis

Coarse-Graining Crowd Dynamics Simulations with Stochastic Differential Equations

Author: Kaili Wang
Examiner: Prof. Dr. Hans-Joachim Bungartz
Advisor: Dr. Felix Dietrich
Submission Date: March 15, 2022



I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

March 15, 2022

Kaili Wang

Acknowledgments

I would like to thank Dr. Felix Dietrich for taking the responsibility to be my advisor. All members, especially the coordinators of TU Munich's Computational Science and Engineering program for doing their best to help me succeed with my academic studies, namely extending the duration of my studies multiple times, it was perhaps a record. The Munich KVR department for international students, with multiple problems I ran into, their willingness to help exceeded my own expectation in a good way. M.J for offering my first place to stay in Munich and welcomed me into their home. Prof Dr. F.J for offering me my first student research position and vouched for me. Prof Dr. Q.C and Dr M.L from my previous institution for supporting my application to my masters program. Ms L.L from my high school for supporting my application to my dream university's bachelors program, which I sadly ended up not taking the offer. Mr Z.T.Q and other teachers from my middle and elementary school in China for supporting my interest in Physics and Mathematics. Some of the above parties may sound unconventional for me to mention. I took extra long to finish my studies. This would not be smooth and possible without many parties offering a little help here and there along the way. Many of these parties' jobs are to support students, therefore, I hope I am right about this being the right place to acknowledge their efforts.

My parents W.N.N and W.A.Z, my grandpa W. B. H for supporting me. My grandma, Li.Xiu.Fang, who loved and protected me and was also my first math teacher. When I was curious about things outside the curriculum she was happy to guide me instead of being annoyed. I wish I had more time with her.

Lastly, my significant other, Ms O.Nikolett. for sticking with me through thick and thin. Always guiding me with her intelligence and positivity and special care for me, regrettably I was too stubborn, scared, and slow to understand her wisdom and subtlety.

Thanks to all the people above, and those I failed to mention yet have helped me no matter how little or much, for burdening themselves with me. There are certainly too many for me to mention. I hope I will also do my part right, so all the goodness toward me will end with a good legacy.

“It is our choices, that show what we truly are, far more than our abilities.”

-J.K. Rowling

Abstract

Agent-based simulations provide accurate and detailed insights into dynamical systems. However, the runtimes and computational burden of these microscopic simulations often prohibit large-scale parameter studies or overviews on coarser scales. These tasks can be done with coarser models, but those often suffer from low accuracy compared to the agent-based approach. As a remedy, this project aims to learn coarse models directly from the agent-based data. We identify effective stochastic differential equations (SDE) for coarse observables of fine-grained particle- or agent-based simulations. These SDE then provide coarse surrogate models of the fine scale dynamics. The coarse variables in question for this project are the infection states for a viral disease that is spreading through a local population. The crowd simulation software *Vadere* will be used for the agent-based modelling and simulation, all necessary software implementations are already available, but can be extended and tailored to our needs. To learn the SDE, we approximate the drift and diffusivity functions of the effective SDE through neural networks, which can be thought of as effective stochastic ResNets. The loss function is inspired by the structure of established stochastic numerical integrators, namely the Euler-Maruyama integrator. Our approximations can thus benefit from error analysis of these underlying numerical schemes. They also lend themselves naturally to “physics-informed” gray-box identification when approximate coarse models, such as mean field equations, are available. The learning procedure we use does not require long trajectories, works on scattered snapshot data, and is designed to naturally handle different time steps per snapshot.

Contents

Acknowledgements	vii
Abstract	ix
1 Introduction	1
2 State of the Art	3
2.1 Agent Based Modelling and Simulation Systems in Vadere	3
2.2 Neural Network	4
2.3 Stochastic Differential Equations	6
2.3.1 Euler-Maruyama Scheme	7
2.4 Implementation	7
2.4.1 Scenario Building	8
2.4.2 Infection Model	10
2.4.3 Training and Testing Data	11
3 Scenarios	15
3.1 Train Platform Scenario	15
3.1.1 Inspiration	15
3.1.2 Design in Vadere	17
3.1.3 Data Generation	19
3.1.4 Results	20
3.1.5 Train Platform Long day	25
3.1.6 How NN learns the data	28
3.2 Train Platform Scenario with New Group Counting Method	28
3.3 Bottleneck Scenario	31
3.3.1 What was done before this thesis	31
3.3.2 Observing the previous results	32
3.3.3 Bottleneck Scenario Rerun Results	34
3.4 Classroom Scenario	37
3.4.1 Testing Data	38
3.4.2 Training Validation and Loss	38
3.4.3 True vs Approximated SDE Paths and Discussion	39
4 Conclusions	41
4.1 What can be improved in the future	41

1 Introduction

We identify effective stochastic differential equations, SDE, for coarse observables of agent based simulations of a train platform, a classroom, and an entrance. The drift and diffusivity functions of these SDEs are being approximated by neural networks. Agent based models are often used to model complex dynamical systems, they are thorough and capture great detail regarding interactions between each individual agents amongst one another in a group. They can include additional attributes on top of movements of pedestrians, such as the spread of infectious disease SEIR model. They can be, however, time consuming and require a lot computational resources, while outputting microscopic fine-grained dynamics that may not be the priority of interest. Therefore work has been done joining agent based simulation, stochastic differential equations, and neural networks together. The agent based simulation provides the coarse observable data to train the neural network, which approximates the drift and diffusivity of the SDEs. The abundant data of the agent based simulation in this case is an advantage of training the neural network (NN)

In this project we subject the setup to vigorous tests and push its potential as far as possible by giving it scenarios that are demanding, versatile, useful and with a degree of realism. The scenarios also let us peer into the spread of infectious diseases and how the scenarios size, the spawning frequency of agents affect or not affect the spread of infectious disease. Through demanding the setup to learn a wide range of scenarios and making improvements when necessary, we ensure the set up is applicable and competent for any scenario we throw at it.

Chapter 2 covers the background knowledge and introduces the tools we are using to construct this project. Chapter 3 Walks thoroughly over multiple scenarios, such as a train platform, a class room, and revisit a pedestrian exiting the door scenario developed before. Chapter 4 we discuss what we have learned in the project and from the scenarios.

2 State of the Art

This chapter highlights the background knowledge, and areas needed for this experiment, as well as the tools being used to solve its problem. The work connects three well researched disciplines of computer science, which are: Agent Based Modelling and Simulation Systems, detailed in 2.1. Neural Networks, at 2.2, Stochastic Differential Equations, detailed in 2.3. Previous works have been done to connect the three disciplines together. Section 2.4 introduces the workflow and implementation.

2.1 Agent Based Modelling and Simulation Systems in Vadere

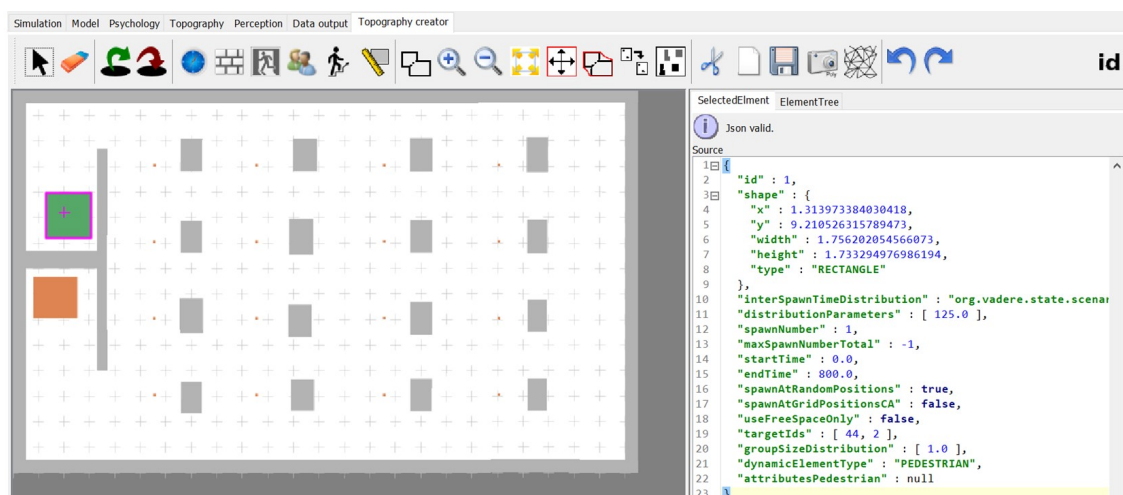


Figure 2.1: Vadere User Interface and Topography Creator

Vadere is a free, open source framework for simulating pedestrians and crowd dynamics. [1] Several locomotion models are already implemented in the framework such as the optimal steps model which we will be using. [2]. Vadere has an ergonomic, easy user interface that does not contain too many buttons, making it easy for users to get started. Such as in figure 2.1, the topography contains mainly of four elements: "source", colored in green, responsible for spawning pedestrians into the scenario; "target" colored in orange, responsible for letting pedestrians pause and/or exit the scenario; "obstacles" colored in grey; agents, which are the pedestrians shown as red and green circles in 2.2.

The user can focus on designing the cosmetic look of the overall scenario, for detailed attributes of each element in JSON presents itself only after selecting such element in the scenario. This way Vadere does not overwhelm the user with too much details, while allowing simple access to the details if needed.

Vadere keeps itself open and accessible. Not only we can run Vadere through its own gui as a normal application, we can also run it with a python script on command prompt or another terminal. This allows our project to be developed modularly, changes to one part do not require restructuring the whole project. The Vadere scenario files can also be edited using a text editor. Vadere can be connected, integrated well with other ecosystems, in this case python, while not inhibiting users' usage of external tools. Robustness and adaptability can be exceptionally helpful in simulating the spread of a disease, as the disease is spreading in the real world, to allow us query into different versions of futures before they may or may not take place.

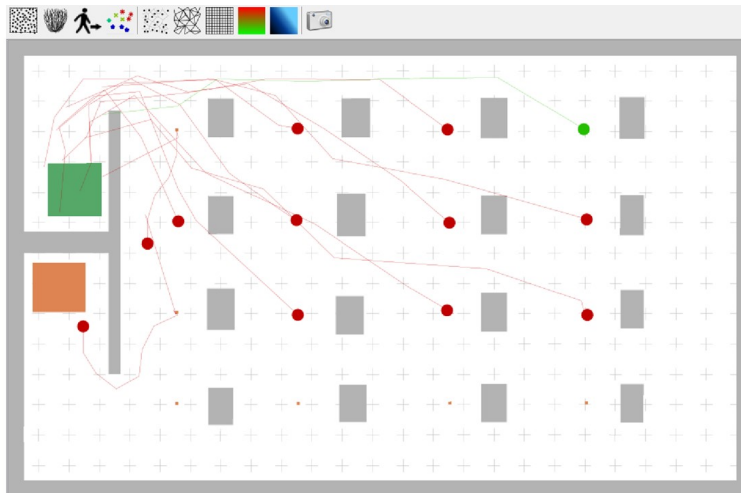


Figure 2.2: Students Inside a Small Classroom and the Spread of an Infectious Disease

2.2 shows a scenario of a small classroom with students as agents. Additional attributes can be added to Vadere such as "AttributesSEIRG" for the spread of infectious diseases, which enables the simulation to that aspect in addition to pedestrian movements integrating these two disciplines. In 2.2 the agent susceptible to infection is colored in green, the agents that are infected are colored in red.

2.2 Neural Network

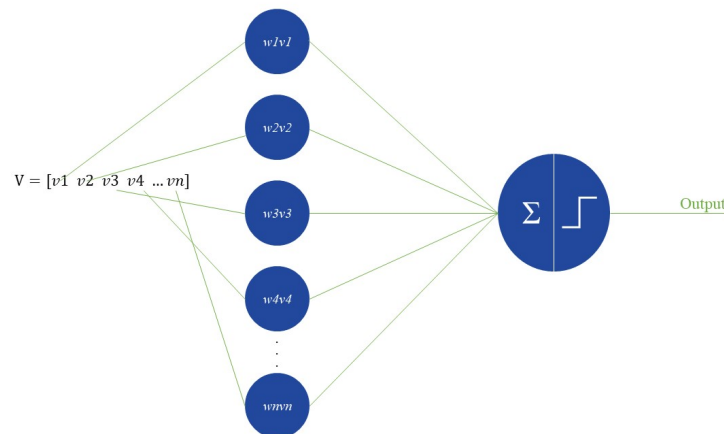


Figure 2.3: A One Layer Neural Network

Artificial Neural Network (ANN), are more commonly referred to as just **Neural Network (NN)**, is a family of the most well known algorithms in Machine Learning. It is inspired by biological neural networks in animal brains. Although the idea has been around in the 20th century, it was popularized as computers computational power has gotten better. The basics of NN are already widely available from many educational institutions and other sources. [3]

The cosmetic appearance of NN is similar to biological neural networks while also using similar naming scheme [4]. Such as in 2.3. The input is practically always a vector. Each of the vector's elements get passed on to its own corresponding neuron in the first layer of the NN, and multiplied with the weight at that particular neuron. The products at the neurons are added and then put in an activation function, at last outputting the result.

Modern Neural Networks being used are almost always containing multiple layers. In practice neural networks other than for educational purposes, are mostly multilayered, such as the network used for this project in 2.4. These networks have an input layer, output layer, and few hidden layers. Input layer takes inputs from outside the network, output layer exports outputs out of the network, all other layers are hidden layers since they only interact with elements passed within the neural network.

The general purpose of neural networks is to first learn from known data and corresponding outcome. Then given new data to calculate possible outcome. Say we have a group of known data inputs and outputs. we send the inputs into the NN, the in-

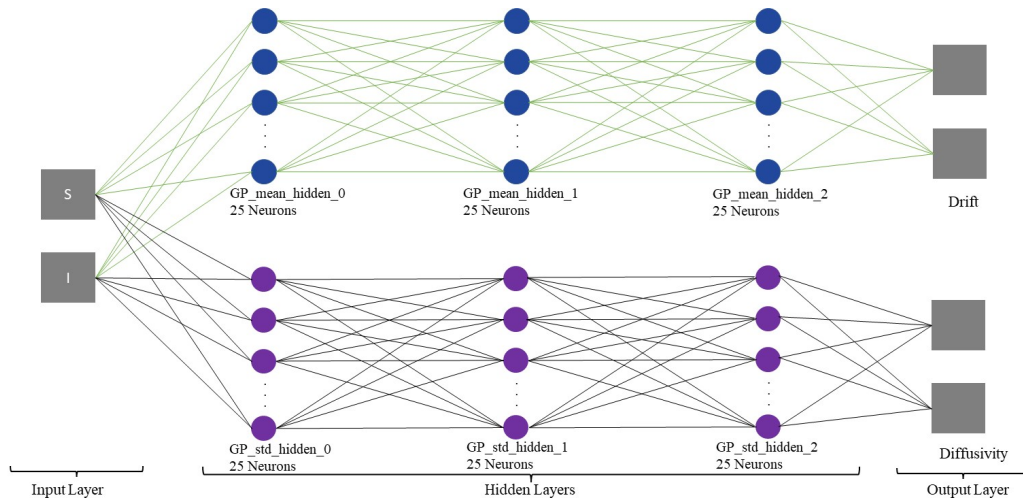


Figure 2.4: Network Structure

put data will be affected by the weights of the network due to multiplications, and the NN returns some outputs. We then calculate the difference between the real output and network out, and prompt our network to change its own weights to minimize that difference till it is considered acceptable. This process is called training the neural network. Once all the weights inside the NN are finalized, the NN can be put to use to predict future data.[3]

2.3 Stochastic Differential Equations

A stochastic system, in contract to deterministic system, is one that does not always produce the same output for a given input.

$$dx_t = f(x_t)dt + \sigma(x_t)dW_t \tag{2.1}$$

The SDE used in this project is formulated as in equation 2.1 [5].

f is the drift of the SDE, and σ is the diffusivity of the SDE

The problem is to identify drift and diffusivity through two neural networks, shown in 2.4. In our case the two networks structures are identical, just one of them being used to identify drift, the other diffusivity. The loss function is inspired by, and embodies the structure of established stochastic numerical integrators, in this case the Euler Maruyama Scheme. [5]

2.3.1 Euler-Maruyama Scheme

The Euler-Maruyama scheme is a forward numerical integrator as shown in 2.2, h is the step size or small changes in time.

$$x_1 = x_0 + hf(x_0) + \sigma(x_0)\delta(W_0) \quad (2.2)$$

x_1 is modelled with Gaussian Distribution as a point drawn from a multivariate normal distribution. [5]

$$x_1 \sim \mathcal{N}(x_0 + hf(x_0), h\sigma(x_0)^2) \quad (2.3)$$

$$\theta := \arg \max_{\hat{\theta}} \mathbb{E}[\log p_{\hat{\theta}}(x_1|x_0, h)] \approx \arg \max_{\hat{\theta}} \frac{1}{N} \sum_{k=1}^N \log p_{\hat{\theta}}(x_0^{(k)}|x_1^{(k)}, h^{(k)}) \quad (2.4)$$

(x_0^k, x_1^k, h^k) are accessible in the training data set, but not drift f and diffusivity σ . To approximate the two, probability density ρ_{θ} has been defined of the normal distribution in 2.3, afterwards, given the neural networks f_{θ} and σ_{θ} , ask that the log-likelihood of the training data under the assumption in 2.3 is high:

The then formulated loss function can be minimized to obtain the neural network weights θ The logarithm of the probability density function of normal distribution, together with the mean and variance from 2.3, yield the loss function [H]

$$\mathcal{L}(\theta|x_0, x_1, h) := \frac{(x_1 - x_0 - hf_{\theta}(x_0))^2}{h\sigma_{\theta}(x_0)^2} + \log|h\sigma_{\theta}(x_0)^2| + \log(2\pi) \quad (2.5)$$

2.4 Implementation

The workflow can be separated roughly into three sections. The first section is to create and finish editing a scenario on Vadere Gui, 2.4.1. The second is to run the scenario a certain amount of times and generate training and testing data, this is done with a python code which accesses the scenario file and run it a number of times and output interested information. The third section is to feed the generated data into the neural network for training.

2.4.1 Scenario Building

Open *vadere-gui* and create a project. Click *scenario*, and start one. To open and continue editing an existing scenario, click *"Generate Scenario from Output"*. [2.1](#) as shown before is an example of a classroom, *"Topography Creator"* enables one view and modify it. It is a class of 12 students. The grey colored are desks the desks for students. The students enter the scenario from the sources, colored in green. The agents will then head to their targets, colored in orange, the target can absorb the agents letting them exit the scenario or make the agents wait for a certain amount of time. In this case they will first go to the seats, the very small orange dots next to the grey desks. They will stay in their seats for the duration of the class then leave through the exits, colored in orange. Both seats and exits are targets.

Attributes

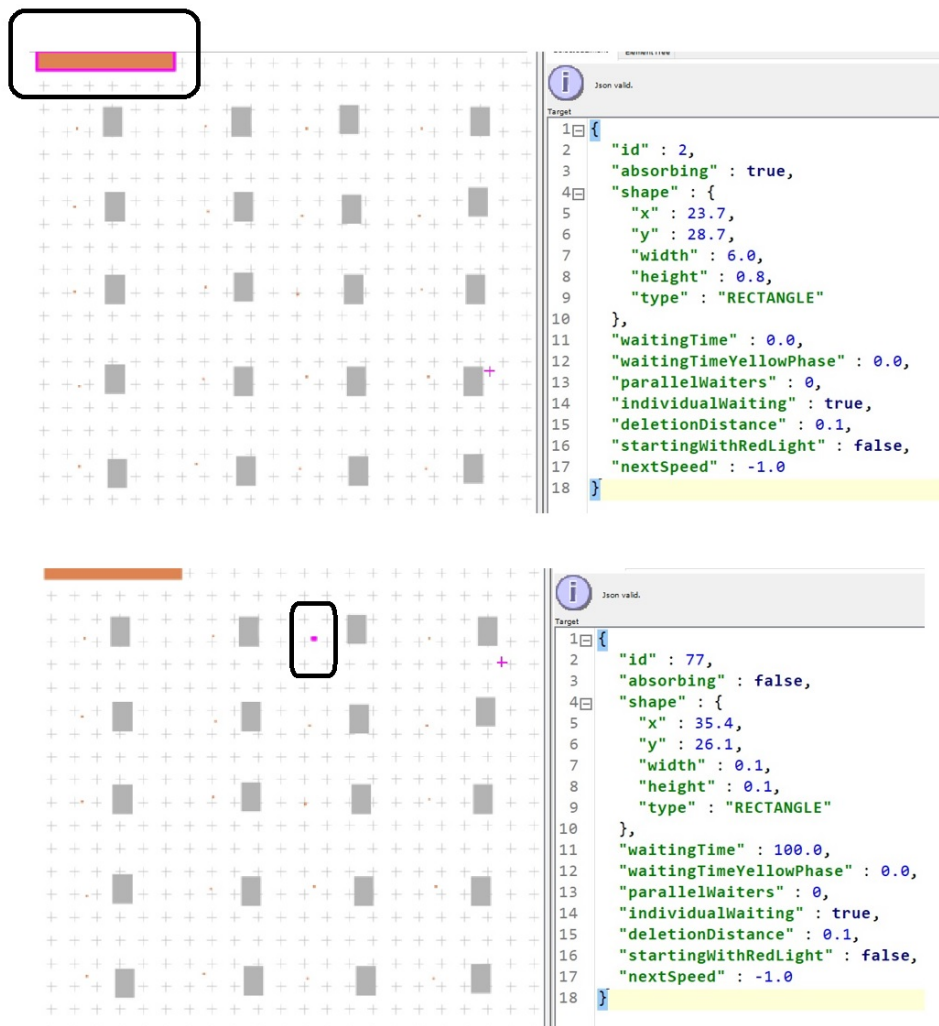


Figure 2.5: Targets in Vadere and their properties

For targets we are mostly concerned with three of its properties: "waitingTime", "id" and "absorbing". To determine if a target should be an exit or a temporary place to pause at, we select the target element to modify its properties in JSON. The four targets on the edges of the scenario's "absorbing" is set as true and "waitingTime" is set to 0, which means the agents disappear and leave the scenario upon reaching the target. The tiny targets representing seats, on the other hand, "absorbing" is set to "false" and

"waitingTime" is set to 100, the agents will sit on their seats for a such duration in Vadere. "WaitingTime" at the seats are set to as low as possible since we observe the infection does not spread when agents are waiting at their seats. "id" allows us to give targets an ID in order to coordinate between sources and targets.

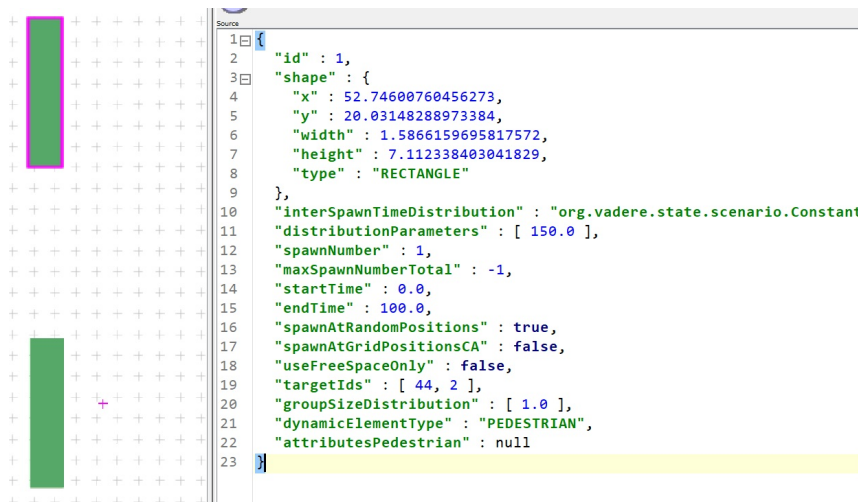


Figure 2.6: Sources in Vadere and their properties

For sources we are mostly interested in its "targetIds", "distributionParameters", "startTime", "endTime", "spawnNumber". "targetIds" determines which targets the agents spawned from selected source will go. "distributionParameters" allows us to set the period of spawning interval, for most scenarios in this projects agents are spawned periodically during the simulation, instead of being released all at once. "startTime" and "endTime" decide the entire duration of spawning. "spawnNumber" is the number of agents each time the source spawns.

2.4.2 Infection Model

Additional attributes can be added to a locomotion model to give it another layer of information about the agents. In Vadere such attributes concerning the spread of infectious diseases can be found under "Model" with "AttributesSEIRG" which initializes attributes regarding the population fraction of Susceptible, Infected, and Recovered.

The rates, "exposedRate", "infectedRate", "recoveredRate", determines the initial rate of agents spawned. For example, if 200 agents are being spawned with exposeRate = 0, recoveredRate = 0, infectedRate = 0.2, then considered some stochastic behavior

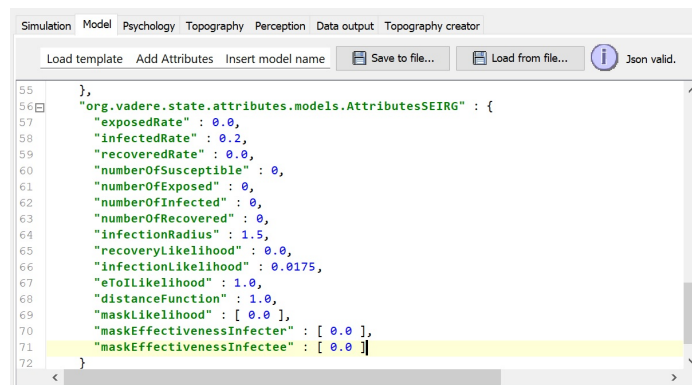


Figure 2.7

with random seeding approximately 40 agents would be initially “infected”. In the experiments all agents will have either susceptible (S), or infected (I) as their status, therefore exposedRate and recoveredRate is set to be 0. The numbers, “numberOfSusceptible”, “numberOfExposed”, “numberOfInfected”, “numberOfRecovered”, actually control the same thing as the rates, which is how many agents are spawned with what initial status. If all the numbers are set to 0, then Vadere will look to the rates for initial spawning conditions. If the numbers are not set to 0, then they will override the conditions of the rates. For example, if infectedRate = 0.3, yet numberOfSusceptible = 5 and numberOfInfected = 195, then for 200 agents spawned roughly 195 agents would be infected, not 60. In this project we will initialize and pass information between scenarios using rates only, and the “numbers” are all set to 0. “Rates” is ergonomically easy since the rate work as an independent variable, and others are dependent variable of rates. For example, if one change the total spawning number from 100 to 200, 345, any number, the initial number of agents in any status is always the total number spawned multiply by the corresponding rates. Whether the experiment is a classroom, a bus stop, or a supermarket, whether we have one source spawning 200 people or 200 sources spawning 1 person every 5 seconds. The rate will be applied correctly without any errors.

2.4.3 Training and Testing Data

Generating

The generating process is done by running Vadere recursively and outputting information of agents of group IDs. It involves hundreds of iterations that need one simulation’s output “infectedRate” as input “infectedRate” of the next simulation, therefore not feasible to click open and run manually. This process is instead done by running a python script on a terminal that recursively run Vadere console and coordinate the

processes, in our case the command prompt of Windows 10.

The python script also include a function that can edit the Vadere scenario's attributes in order to set initial conditions. The items in the script that require checking or changing are: "InfectionLikelihood" determines that likelihood of the disease spread from one agent to another, this can also be set in Vadere GUI and not modified in the script. "totalIterations" is the total number of iterations of the outer loop, there are total of 2 loops, one inner, one outer. "days" is the total number of iterations of the inner loop. For this project we set the "days" to 4 for most of the experiments, meaning there are four consecutive days each continues the previous one with its "infectionRate". "output path" the path of the output folder. "scenario path" the path to the scenario we wish to run, the file ends with *.scenario*. "infectedRate" for training data this is initialized randomly, for testing data we choose an infectedRate we are interested in studying as the initial infectedRate of the first days. All of these are things to be changed if we want to do a different experiment with a different scenario.

Each time the Vadere scenario has been run, a csv file containing information regarding the group counts, the amount of agents that are susceptible, infected, versus time. We can load it in to spreadsheet softwares such as excel to examine them. Such as in 2.8. As we can see for the classroom of 63 students scenario. Initially at simulation time 0, there are 50 susceptible, 13 infected. This corresponds to the 0.2 infected rate. These information will be used to train and test our NN.

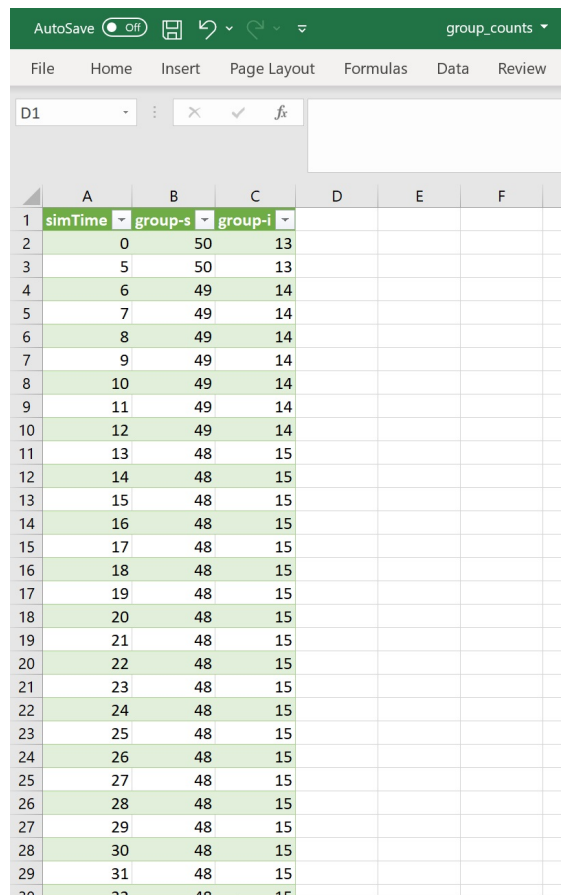
Group Counts

The "Group Counts" count the the agents that have been spawned into the scenario, whether they left the scenario or not. It does not count the agents that have not been spawned into the scenario yet. This is okay for scenarios that all agents at spawned into the scenario at the same time. During our experiments we found this is not ideal for scenarios which agents get spawned into the scenario in multiple waves at different times, as we see later on.

Instead we want to add agents that have not yet been spawned to group-s and group-i, how this is done is to assume the initial susceptible and infected rate applies to agents that have not yet been spawned, and assign them to group s and i accordingly.

Tools

Jupyter Notebook, is the environment which we mostly develop our code in. Its notebook appearance allows us to easily navigate through. Data processing packages such as numpy, pandas are imported in, making it easy to process and visualize large amounts of data.



The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F
1	simTime	group-s	group-i			
2	0	50	13			
3	5	50	13			
4	6	49	14			
5	7	49	14			
6	8	49	14			
7	9	49	14			
8	10	49	14			
9	11	49	14			
10	12	49	14			
11	13	48	15			
12	14	48	15			
13	15	48	15			
14	16	48	15			
15	17	48	15			
16	18	48	15			
17	19	48	15			
18	20	48	15			
19	21	48	15			
20	22	48	15			
21	23	48	15			
22	24	48	15			
23	25	48	15			
24	26	48	15			
25	27	48	15			
26	28	48	15			
27	29	48	15			
28	30	48	15			
29	31	48	15			
30	32	48	15			

Figure 2.8

Spreadsheet program, in our case MS Excel. It is one of the easiest things to use look into one single days information overtime. Sometimes we may not want to look at a few hundred different days together but just look at what happens in a single day. It is also a good tool to sketch some simple adding and multiplying operations before we write the formula in python.

3 Scenarios

It is not as much of a success if our work can be only applied to an exclusive scenario. Therefore we explore a few scenarios of diverse setups to see how useful our work is.

3.1 Train Platform Scenario

3.1.1 Inspiration

The Station

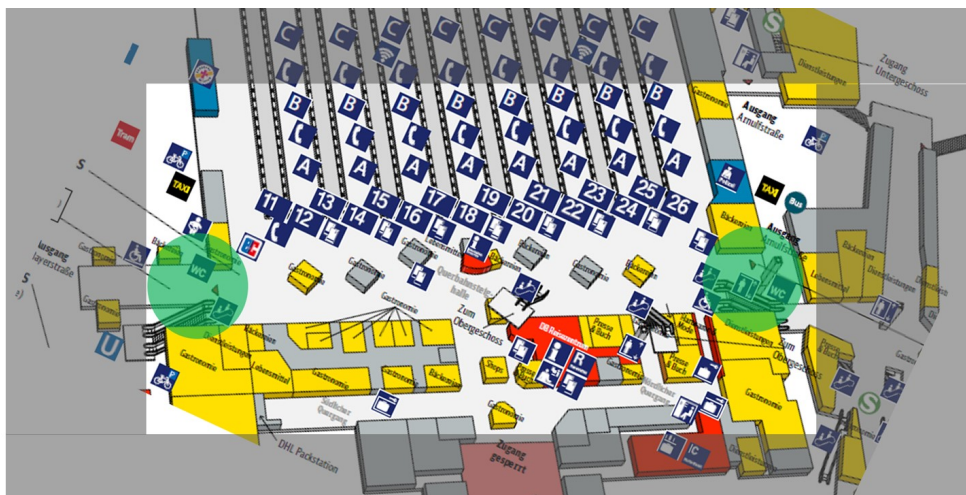


Figure 3.1: Munich Central Station

The design of the train platform and later the train station scenario is inspired by the layout of the Munich Central Station. We consider a section of the main hall. The main hall containing platforms 11-26, shown not shaded in 3.1, is only accessible via the entrances indicated with the green circles in 3.1. Each island, or really peninsula allows access to two trains on its two sides.

The trains

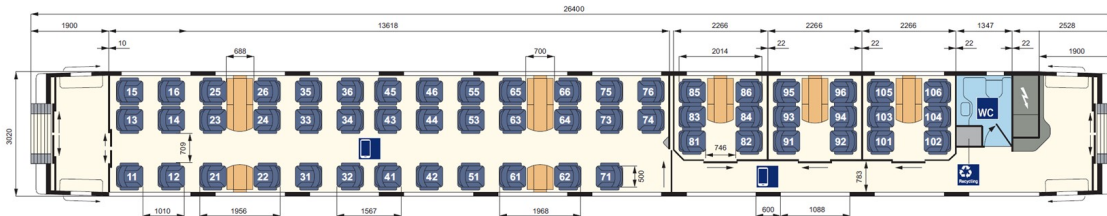


Figure 3.2: A Real Life Train

We have a train with one driving car that takes no passengers, and two typical wagons taking 56 passengers each. A wagon is based on 3.2, 26.4m in length, 3.02m in width. There is one such train on each side of the peninsula, totalling two trains. If no person makes any mistake with getting on the wrong train, or arrives too early or too late. Assuming full booking, 224 passengers will board the two trains.

For the moment we only consider empty trains and departing passengers, future work is encouraged to add arriving passengers getting off the train.

The environment

We will build a platform scenario as shown in 3.3, that contains one peninsula allowing access to two trains on each of its sides, and an area of the main hall. All passengers will arrive to the main hall from the two sides, in the case of the figure it's top and bottom; however, not all passengers in the main hall will board these exact two trains, they may be going for other trains. The majority of agents will simply move from one side to the other side, while 244 agents will go to their trains. The peninsula's width is set to 2x the width of the trains.

The passengers

Not everybody who arrives at the train station need to take the same trains. The exact train station traffic will depend on hours of the day, and more importantly train schedules. Train schedules do not only depend on one train station since they connect and



Figure 3.3: Train Platform Scenario in Vadere

coordinate different regions. They can also be set to take as many people in one trip as possible due to profit maximization. Policies and other factors may also affect scheduling to prevent overcrowding of passengers and commuters. It would be nice to refer to studies of how train schedule is set and its impact on passenger and commuter density at the station. In this project we will assume a simpler case of the constant crowd density, as in 3.1 there are 8 peninsulas, so 1 peninsula would get 1/8 of the total passengers in the station. Therefore the total number of agents spawned is set to 1792 with 224 agents going to the 2 trains in our scenario.

3.1.2 Design in Vadere

The scenario contains 2 platforms sharing a peninsula. The peninsula is only connected to the main land area on one of its sides. The scenario's overall Width x Height is 70 x 24.

Targets

Each target serves as 1/4 of a train, taking 28 passengers. Each target contains a unique ID, the labeling and ID numbers can be seen at 3.4. We have two trains, each train has 2 wagons, each wagon can take 56 passengers. Each wagon has two doors. Since agents seek the closest opening of the target, in order to make the agents go to the door that is further away, a unique target is created for each door of the train. For instance, Target 21 and 22 belongs to the same wagon, but are separate targets otherwise every agents wanting to get on this wagon will only use the opening at target 21. The targets used for trains are: 21, 22, 23, 24, 31, 32, 33, 34. Target 11 and 12 are created to receive passengers

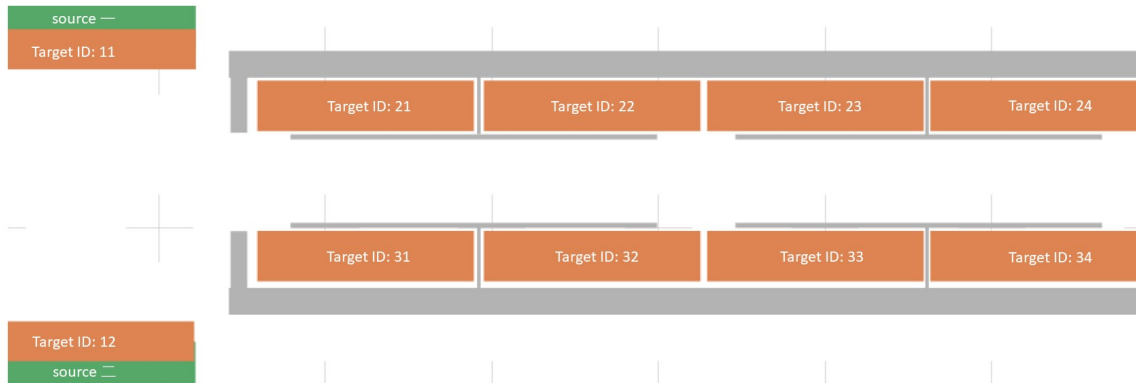


Figure 3.4: Train Platform Scenario in Vadere

in the train station, that do not need to board the trains in the scenario. Two targets are created since people can go toward both directions. As for numbers, target 21, 22, 23, 24, 31, 32, 33, 34 will receive 28 agents each since the two trains can take 224 people at full capacity. Target 11, 12 will receive 784 people each, since 1568 passengers will board other trains or head elsewhere in the train station.

Sources

All sources are super-positioned at position top next to target 11 and position bottom adjacent to target 12. For each target, a corresponding source has been created, totalling 10 sources for the 10 targets. Since they are overlapped on top of one another, they cannot be seen separately on 3.4. Each source spawns agents exclusively for one target, each target receives agents exclusively from one source. For simplicity the sources will be referred to by their corresponding target's IDs, for example, the source spawning agents going to target 21, will be referred to as source 21. Source 11 and 12 spawns agents that are not boarding the trains, which is the majority of agents in this scenario. They are placed on the opposite sides of their targets, target 11 and 12. The sources' "distributionParameters" are set to 10.0, "spawnNumber" set to 14, "startTime" set to 0.0, "endTime" set to 559.0. This means these sources will spawn agents starting at 0.0 seconds in Vadere, spawns 14 agents every 10 seconds, they will spawn 1568 over the course of the simulation. Sources 21, 22, 23, 24, 31, 32, 33, 34 are spawning agents that board the train. Their "distributionParameters" are set to 20.0, "spawnNumber" set to 1, "StartTime" set to 0, "endTime" set to 559.0. Meaning they each spawn 1 agent every 20 seconds, totalling 224 agents over the duration of the entire scenario.

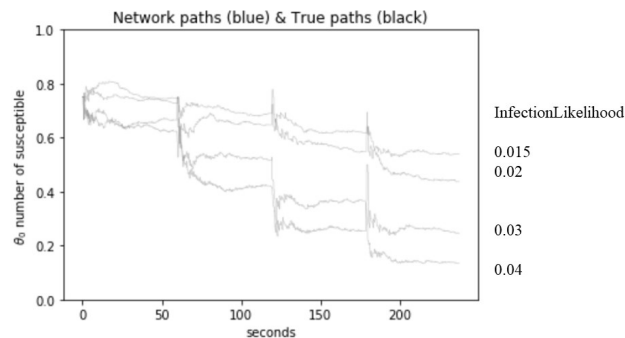


Figure 3.5: Train Platform Scenario Susceptible Rate for Different Infection Likelihoods

3.1.3 Data Generation

Same as the other experiments, in this one we run the scenario recursively for 4 days. Each day's initial conditions are taken from the previous day's final conditions. For training data we do this 200 times, generating totalling of 800 files.

Infection Likelihoods

For `infectionLikelihood`, it was initially set to 0.015, which is the same as the Bottleneck Scenario in 3.3. However, unlike the bottleneck scenario, the disease is not spreading as quickly, not everybody gets infected in 4 days. This may be simply due to the scenario being a bigger, more open space than a small entrance or a small classroom. 3.5 shows 4 different infection likelihoods effects. The horizontal axis for time in seconds is based on 4 consecutive days in the scenario concatenated, including the time reduction factor of 10. Notice for the population to be dominantly infected within 4 days, the infection likelihood has to be as big as 0.04. The `"infectionLikelihood"` attribute is set to 0.04.

Notice on 3.5 the decrease of susceptible rate is not smooth but stair like, and occasionally it even jumped up a little. This may be because in the scenario not all agents are spawned at once, instead, the agents are being spawned periodically. So when the previous day ends and a new day starts, even if the susceptible and infected proportions are exactly the same, the number of agents being spawned are different at the end of the previous day vs the beginning of the new day. Therefore real time group counts may show different proportions. To get a smooth curve, we can just spawn all agents at once; however, that would be an experiment of a different scenario, one which people pour

1	Column1	simTime	group-s	group-i	group-r	SUM	s proportion	i proportion
2	0	0	23	13	0	36	0.64	0.36
3	1	5	23	13	0	36	0.64	0.36
4	2	6	23	13	0	36	0.64	0.36
5	3	7	23	13	0	36	0.64	0.36
6	4	8	23	13	0	36	0.64	0.36
7	5	9	23	13	0	36	0.64	0.36
8	6	10	23	13	0	36	0.64	0.36
9	7	11	23	13	0	36	0.64	0.36
10	8	12	23	13	0	36	0.64	0.36
11	9	13	20	16	0	36	0.56	0.44
12	10	14	20	16	0	36	0.56	0.44
13	11	15	20	16	0	36	0.56	0.44
14	12	16	20	16	0	36	0.56	0.44
15	13	17	20	16	0	36	0.56	0.44
16	14	18	20	16	0	36	0.56	0.44

Figure 3.6: Group Proportions Data

into a place in a short amount of time and leave shortly after

Group Counts

Group counts, or proportions of different groups, are calculated based on how many agent has been spawned. Note, it is not based on how many agents is currently on the field, neither is it based on the total number of agents spawned throughout the entire simulation. It is based on up until the current simulation time ("simTime"), how many agents have been spawned, even if they already left the scenario. For example in 3.6 we can see the total number of agents being spawned is 36 for row 2 to row 16, simTime 0 to 18. From simTime 0 to 12 proportion of susceptible is 0.64, simTime 13 to 18 it dropped to 0.56 as some agents are being infected. However, this is only the first 36 of all 1792 agents that will be spawned. At simTime 13, it is 56% of 36 agents that are susceptible, not 56% of 1792 agents that are susceptible. The rest of the 1792 agents will be spawned with initial s and i proportions of 0.64 and 0.36, assuming the effect of random seed is the same at each round of spawning. So at simTime 13, if we are interested in "s proportion" and "i proportion" of the total populatio of 1792 agents, instead of the 36 that has been spawned. It would be

3.1.4 Results

Training Data

3.6 shows the number and size proportions of different group over time. "simTime" is time in Vadere in measured in deciseconds. "SUM" is the number of agents that have already been spawned. "group-s" and "group-i" are number of agents susceptible or

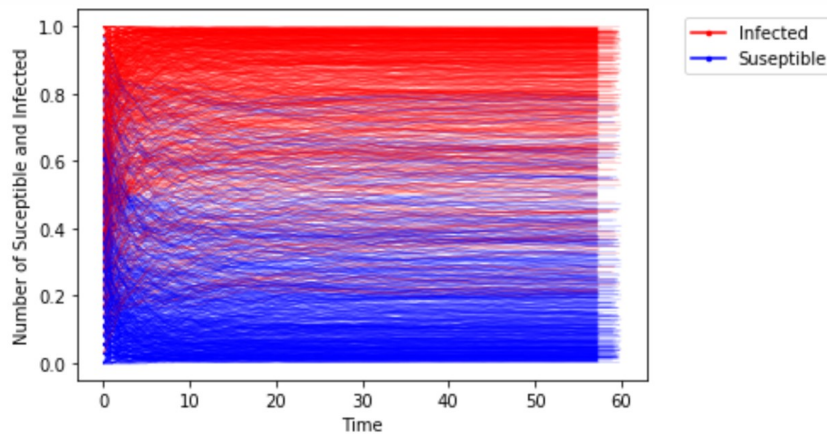


Figure 3.7: Training Data

infected respectively. "s proportion" and "i proportion" are that number normalized to a proportion. For example, on row 2 the columns for "simTime" = 0, "SUM" = 36, "s proportion" = 0.64, "i proportion" = 0.36. This means at 0 seconds, there has been 36 agents spawned, 64% of them are in susceptible status, 36% of them are infected.

3.6 shows only 1.6 seconds of data for 1 day in the scenario. To efficiently visualize s and i proportion for all 800 days of training data over each day's whole duration, the data has been plotted on 3.7 "s proportion" or proportion of susceptible agents are plotted in blue, "i proportion" or infected agents are plotted in red. The starting values are all different. This is because of two reasons. We run the scenarios for 200 rounds, each round contains 4 consecutive days. Each round the proportions are initialized differently, resulting different initial proportions. Each day takes the final proportions of the previous day, as their initial proportions, resulting a later day's initial "s proportion" to be lower and "i proportion" to be higher. We can see susceptible proportions are decreasing, while infected proportions are increasing. It is however hard to what type of decline it is.

We can further visualize the input data with the histograms in 3.8. number of occurrences vs proportions for susceptible and infected separately. We can see for most input data points, susceptible proportions are mostly near 0, infected proportions mostly near 1. This agrees with the coloration of blue is most concentrated at the bottom part, and color red is most concentrated on top in 3.7

Training and Validation Loss

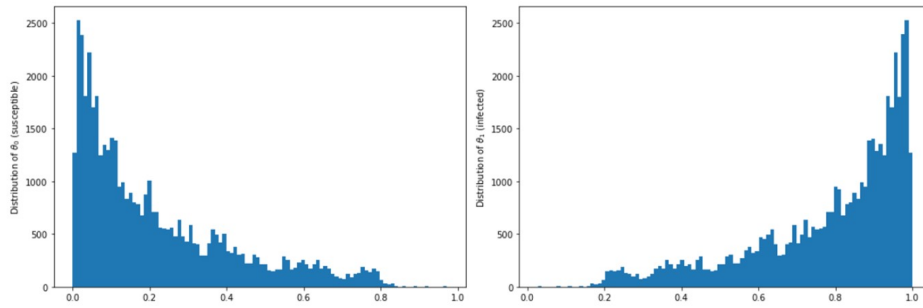


Figure 3.8: Train Platform Training Data Distribution

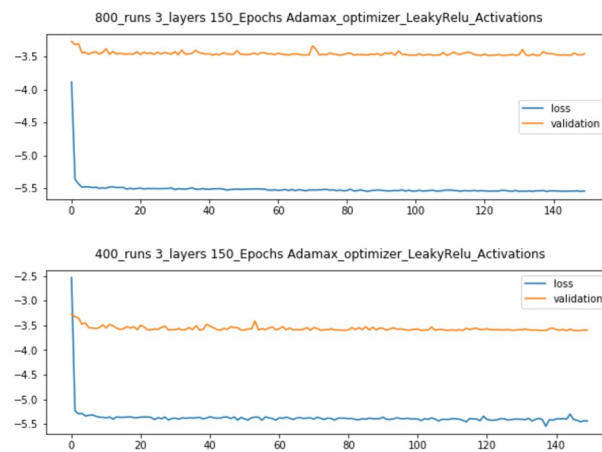


Figure 3.9: Training and Validation Losses with 800 data files versus 400 files

The final training loss is -5.54 and final validation loss is -3.46, which are considered very well. The losses converged within 10, therefore the computational effort is really low. In fact, with a smaller batch of training data, the final training loss is -5.44 and final validation loss -3.59. Both are plotted in 3.9

Test Data

We generated 100 groups of 4 days testing data, totalling 400 files. The initial "infecte-dRate" determining infected proportions was chosen to be 0.2.

True vs Approximated SDE Paths

We construct the approximate SDE Paths using the drift and diffusivity functions from the network, and compare it with the paths from Vadere, also known as test data in 3.1.4. Both start with the same initial conditions. Since 100 groups of 4 days have been generated as test data, we will also generate 100 approximated SDE paths. We plot both of them on the same plots.

3.10 shows the true SDE paths in black and approximated paths in blue for proportions of susceptible. 3.11 is for the corresponding proportions of infected. Once again we can immediately see although the black lines are decreasing, it is decreasing in a stair like fashion, almost having a discrete appearance instead of a continuous look. Suspecting that was because we connected a group of 4 consecutive days together. The new day, despite continuing with the same proportions as the previous day, do not spawn the same amount of agents in the beginning of the day resulting in the group counts looking different. On one hand it also helps us see exactly where one day ends and another day starts. 3.1.5 Attempts to get a continuous looking graph from Vadere by simulating one long day instead of four shorter days.

Discussion

On the other hand, there may also be a degree of realism since a station may shut down and reopen and have different type of traffic from open to close. The disruption between days may also result in the neural network learning the spikes, therefore resulting some approximated paths going to unexpected places. However, the approximated SDE path's disruptive look seem to calm down on day 3 and 4 instead of day 1 and 2. This is possibly because on day 1 and 2 the disruptive look of the true path appear dominant. By day 3 and 4, the dominant trait of the path is that people are continuously being infected, not fluctuations. If the interest is solely on a low error rate and initial and final proportions, then the model serves the purpose rather well.

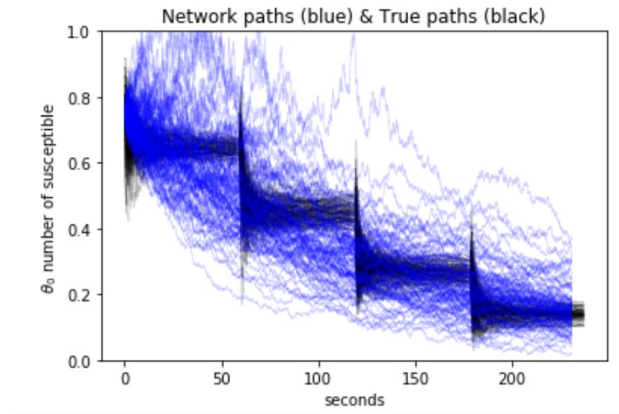


Figure 3.10: Train Platform Susceptible Rate

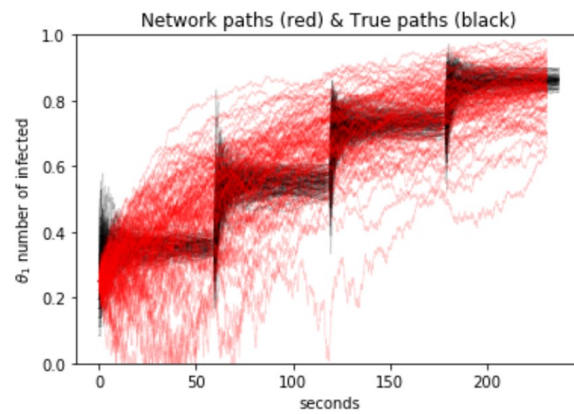


Figure 3.11: Train Platform Infected Rate

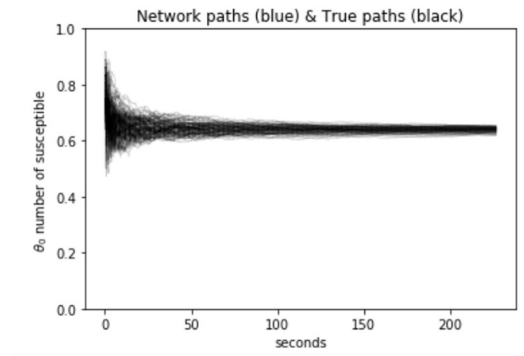


Figure 3.12: SDE Path for 0.2 Initial Infected Rate

3.1.5 Train Platform Long day

Suspecting having separate days and the way group proportions are calculated may result in the data looking stair like. We rerun the train platform scenario for 100 rounds, instead of 4 separate days in each round, we merge all 4 days into one longer day. Looking at the scenario in this manner may offer a different perspective

How we achieve this is to edit some parameters on the Vadere scenario. For "finish-Time" set to 2400 seconds instead of 600 seconds, since we are joining 4 days to one day that is 4 times as long. For each of the 10 sources, we set "endTime" from 559 to 2236, meaning the sources will keep on spawning agents until 2236 Vadere seconds. Since the development is rather modular, we can change attributes of the scenario without changing other parts of the attributes. To generate training and testing data, we keep number of iteration as 100 but change number of days to 1. So we will get 100 files for each instead of 400 files.

Before training the neural network we can immediately see from the testing data in 3.12 and training data in 3.13 that the infection spread slower compare to the case of creating 4 separate days. We take a further look at the all the figures which suggest that infection spreads quickest at the beginning of the day than through out the day. One suspicion may be that the phenomenon occurs due to agents are closer together at spawning. However, the train platform scenario spawns 36 agents every 20 seconds in the same manner; therefore, even if infection spreads faster upon spawning, it would occur multiple times instead of at the very beginning. We have carefully chosen the spawning frequency and the sources dimensions to avoid unnecessary over crowding

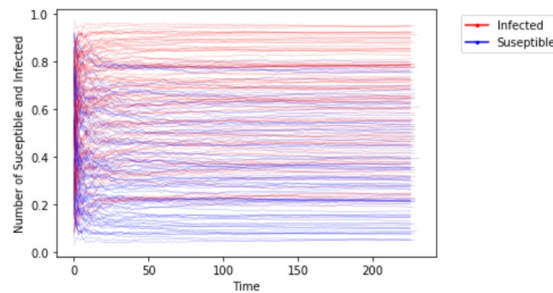


Figure 3.13: Train Platform One Long Day Training Data

of agents upon being spawned. We can still deduce that for a more open scenario like a train platform plus waiting area, even at a high infection likelihood of 0.04 the disease does not spread quickly. If we intend to have agents linger around more in the scenario, what we can do is to create more targets for agents to stop and wait at. The real life inspiration would be there are seats in the train waiting area and snack shops in the train station.

From the testing data in 3.12 and training data in 3.13 we can see, having one longer day is not the same as four shorter days. Most of the infection occurs during the initial spawning period. As the scenario continues, not as much people are getting infected.

Errors and Results

Due to having one longer day instead of 4 shorter days, we have generated 100 files of training data and 100 files of testing data. The training and validation error are shown as in 3.14. By looking at the numerical value of the losses, as well as the Network vs SDE Paths graphs, generating more training and testing data is not a priority or perhaps not even a necessity.

Even though so far it seems most infections took place at the beginning of the scenario, then the proportions of different groups stay constant through out the rest of the scenario. What is of concern is that, the scenario generates a set amount of agents every few seconds in the exact same manner, so if all agents infected each other at the first spawning wave of 0 second, then they should do so again at 20, 40, etc seconds since at those time, same amount of agents are being spawned.

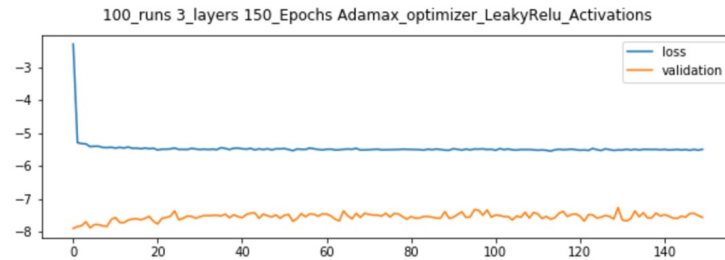


Figure 3.14: Training and Validation Error for a Longer Day Train Platform Scenario

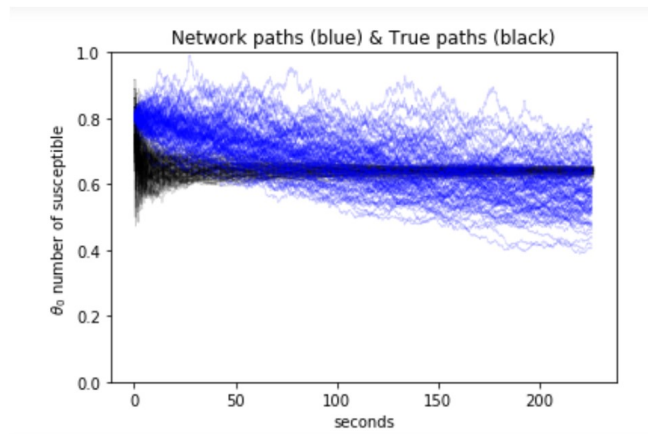


Figure 3.15: Results Train One Long Day Susceptible Rate

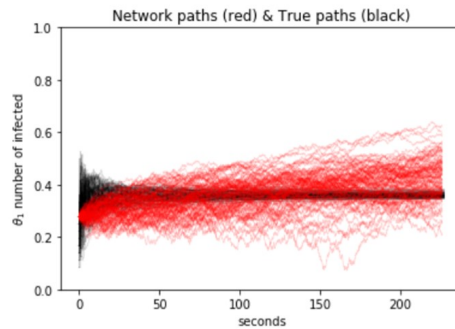


Figure 3.16: Results Train One Long Day Infected Rate

3.1.6 How NN learns the data

The results in 3.15 also leaves one pause and ponder how the NN learns the data. The training data's shape is virtually the same as the test data in 3.15, after an initial sudden drop, it stays flat for the rest of the simulation. The NN path however, it does not follow the shape of the true path. It does not stay flat neither, even though a flat horizontal line is the shape of the SDE for the longest times. On the other hand, in 3.10 and 3.11 having stair like features, and such a shockwave look in the SDE data, also make the network more chaotic the bigger peaks that are very big.

3.2 Train Platform Scenario with New Group Counting Method

We are very excited to resolve the problems occurred in the train platform scenario, which also serves as a progress in making the model more suitable for significantly more scenarios. Due to the limitations on time, we were not able to regenerate data for all scenarios. However, we did re do the train scenario, keeping all other parameters the same while only changing how groups are counted.

Keeping the same conditions as the train platform scenario, such as infectionLikelihood; while only changing the way we count the groups. We visit the train platform scenario again. The number of days stay as 4, while 100 iterations have been done to generate training data, totally 400 days or 400 files.

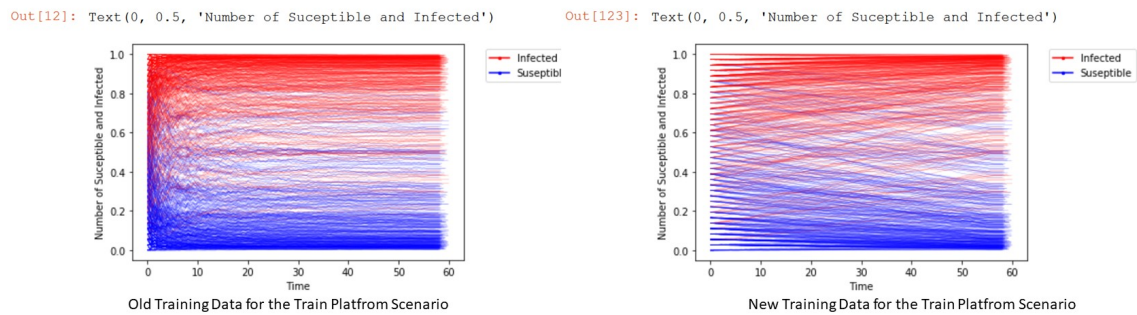


Figure 3.17: Comparison

Instead of only count the proportions of the agents that has already been spawned, we consider both the agents that have been spawned and have yet to be spawned. We can immediately see in 3.17 that the new training data is much smoother, there is not as much noise. Also the spread of disease no longer appear exponential at first, constant later on, but a more consistent spread. The training data at time = 0, on the vertical axis seem to have many uniform gaps looking like hair brush gaps. This may be due to the training data is generated with random seeds in the range of 0.1 to 0.9 for initial proportions. Due to only a small number being spawned at the beginning yet being divided by the entire population of the scenario, the noise becomes extremely small leaving some gaps appearing at time = 0. However, at the end of the simulation as time approaching 60, there are no such wide gaps. This also shows that despite very similar starting conditions, small perturbation causes the results to be different. We can also do a sanity check. The two figures shown in 3.17 despite different appearance at time = 0, the lines go to the same places at end time. As they should since by the end of simulation all agents have been spawned, so at the end of the simulation the group proportions will be the same regardless of which one of the two counting methods we are using.

Our training and validatio loss are very low at -9.9 and -8.5 each. 3.18

The SDE vs Network Paths also appear to be much better. The network paths is highly confident with no lines fluctuation violently as the previous case, as well as not deviating far from the group. The network path agree with each other as well as with the true SDE paths. The SDE paths took unique looking aesthetics almost like the fins of the

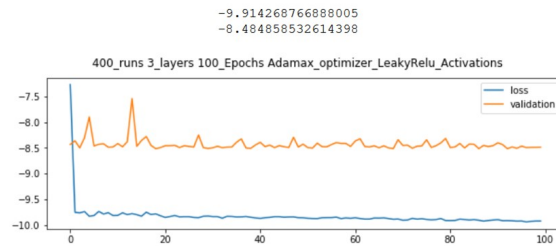


Figure 3.18: Training and Validation Loss after New Group Counting Method

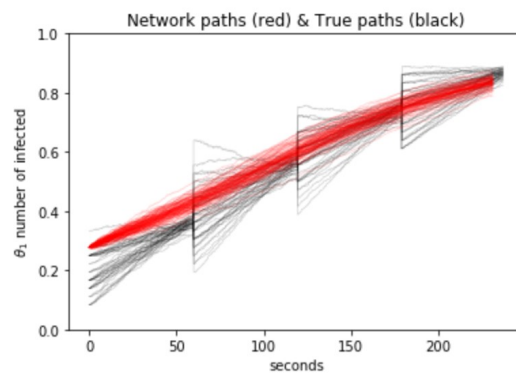


Figure 3.19: Infection Rate Over Time Train Platform New Group Count Method

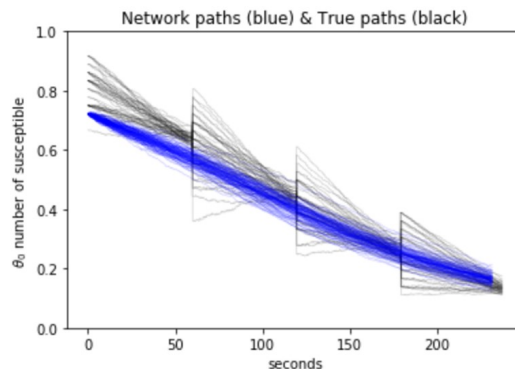


Figure 3.20: Susceptable Rate Over Time Train Platform New Group Count Method

fish, this is again due to every new day despite continueing the previous days infacte-dRate, due to random initialization the inital conditions still deviate. The SDE paths converge quickly despite different starting conditions, suggesting for a bigger scenario like the train platform, microscopic deviations matter not as much. The network is about to learn the true path and almost agreeing with the mean of the true paths, which is exceptional.

3.3 Bottleneck Scenario

This is a scenario that has been doing in the past, we will take the exact scenario without making any changes. All changes made are either in the data generation process

3.3.1 What was done before this thesis

The Bottleneck scenario is a scenario which pedestrians exit through a door. The topography is a lot simpler than the train platform and classroom scenario in which there is only one source, one target, seldom amount of objects and obstacles forming a bottleneck. The details are on [3.21](#)

The number of days was set to 4, each day runs on roughly 30 seconds in the scenario, resulting in 4 days being plotted as 12 seconds considering the time reduction factor of 10.

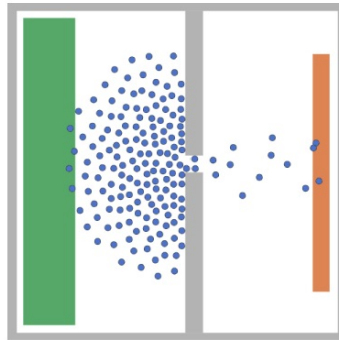


Figure 3.21: Bottleneck Scenario

The area is 20m x 20m. The finishtime attribute was set to 125 Vadere seconds, which should give enough time for all agents to exit the scenario. The target's absorbing property was set to be true. InfectionLikelihood was set to 0.01

3.3.2 Observing the previous results

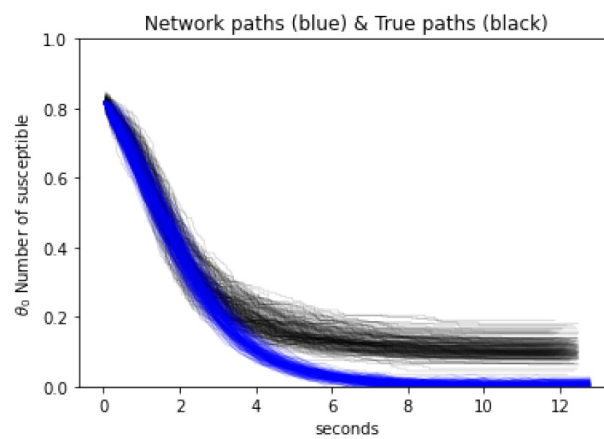


Figure 3.22: Bottleneck Susceptible Rate

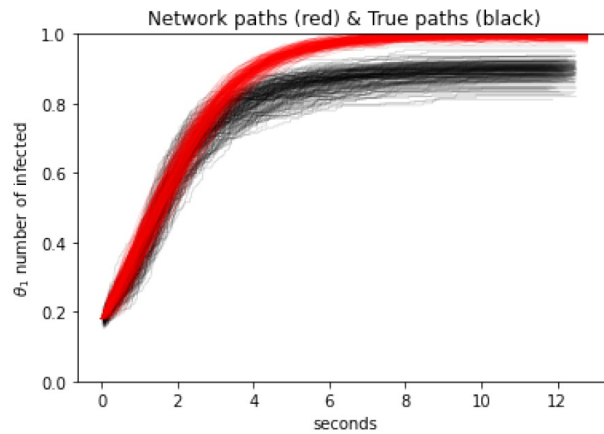
Figure 3.18: Network approximated (red) and true (black) SDE paths for θ_1

Figure 3.23: Bottleneck Infected Rate

We can immediately see that the true path labeled in black converges to roughly 0.8 for infected, 0.2 for corresponding susceptible, which do not agree with what the neural network learned, which is converging to 1 and 0 for S and I. As shown in 3.23 and 3.22

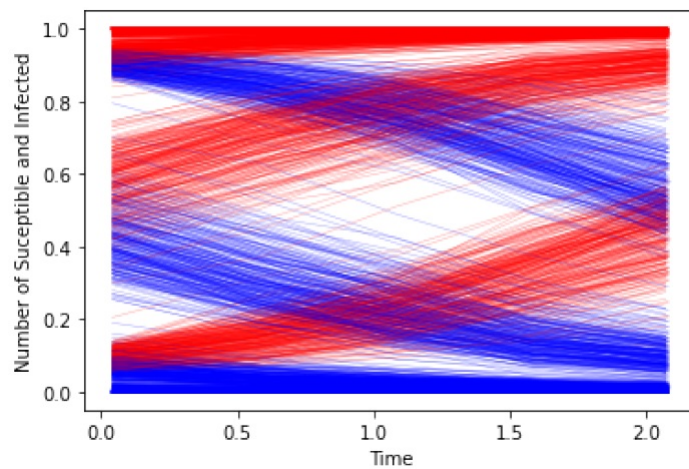


Figure 3.24: Bottleneck Training Data

There may be many reasons why what the network learned path does not agree with the path generated by Vadere. One hypothesis being discussed was that the target is too close to the source and absorbing was set to true, so agents left the scenario before they

were able to infect each other. The first step is to take the scenario and regenerate all the training and test data and check the results.

3.3.3 Bottleneck Scenario Rerun Results

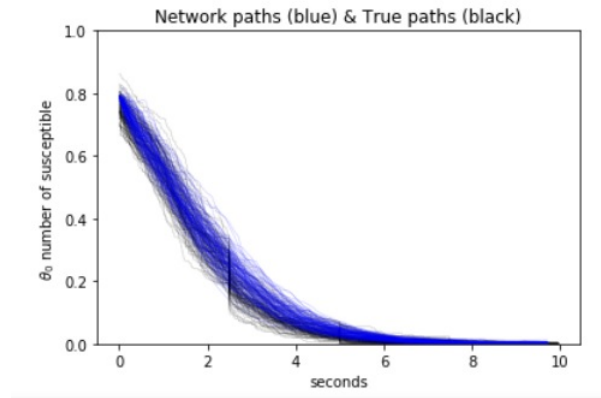


Figure 3.25: Bottleneck Rerun Susceptible Rate

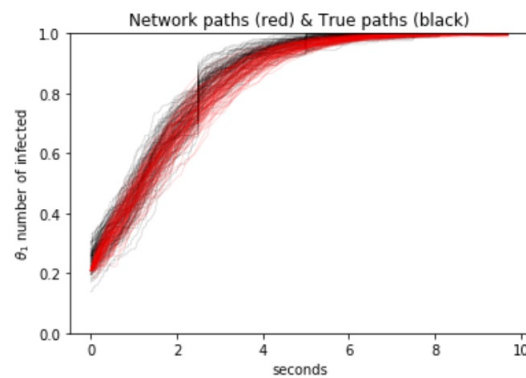


Figure 3.26: Bottleneck Rerun Infected Rate

Let us first look at the result of path generated by the network learn from training data in blue and red, versus path generated using test data.

Bottleneck Scenario Rerun Discussion

Nothing was changed during the scenario building phase. The data generation phase, in the previous thesis the next iterations' initial parameters were controlled using "numberofSusceptible" "numberofInfected" "numberofRecovered"; however, here it is con-

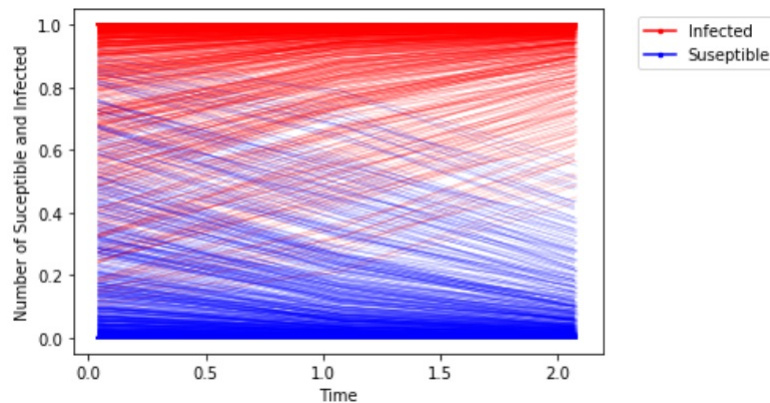


Figure 3.27: Bottleneck Rerun Training Data

trolled by "infectedRate" while the numbers are all set to 0. which may also decrease the probability of typos.

At the rerun the network path agrees the testing path. On figure 3.26 and 3.25, while the rerun itself cannot explain the previous phenomenon, it can ensure the network is adequate in learning this scenario. Basically other than the scenario itself, rest of the process such as data generation and learning the data are done exactly the same way as in the other scenarios

We can have a look at the rest of the experiment regarding the Bottleneck Scenario. The training data is being shown in 3.27, Which is quite nice a suggesting the dynamics of a disease spreading resulted from a crowd gathering

3.27 shows the new training data, compare to the previous training data, it has more consistency. However, the previous network was trained "correctly" in learning all agents will be infected, so the training data being more sparse does not necessarily impact the neural network's ability to learn. The bigger concern was the testing data not converging to all infected, that is most likely due to some setting in either the scenario building phase or the data generation phase.

Using the same hyperparameters defined in the previous section (to be added), we train the neural network with the training data in 3.27. The final training loss turned out to be -4.31 and final validation loss turned out to be -3.12, which means the training is considered very well and validation is considered good. Both losses converge in around 5 epochs, which means the computation time needed to train the network is not high, note that near 100th epoch the losses started to get even lower, this does not necessarily

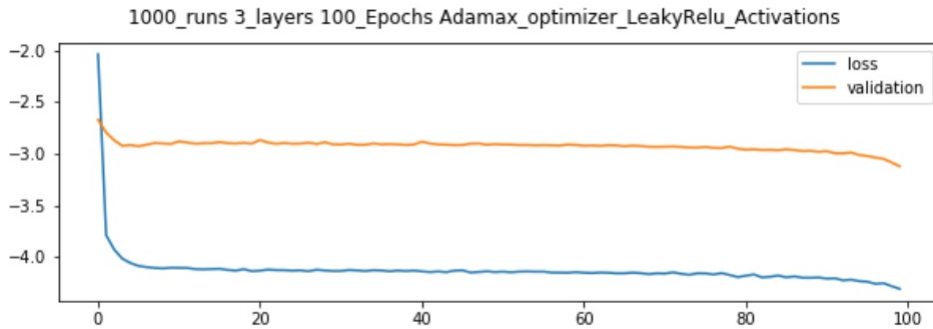


Figure 3.28: Bottleneck Rerun Losses

mean we have to keep training the network since in this case we may only need the loss value to be low enough for our purpose, versus the convergent value is only something we take if we cannot even get a low value. If the value is already low enough for our tolerance, then we can stop the training. The network was trained for 100 epochs with 1,000 time snapshot data. The losses vs epoch are plotted in figure 3.28

The test data is of initial susceptible rate of 0.8 and infected rate of 0.2 A different initial test value can be chosen and regenerated. In conclusion, on the scenario has been switched from train platform to bottleneck, no changes in data generation and neural network process were made, this suggest the procedure is not exclusive to one scenario but useful for various scenarios.

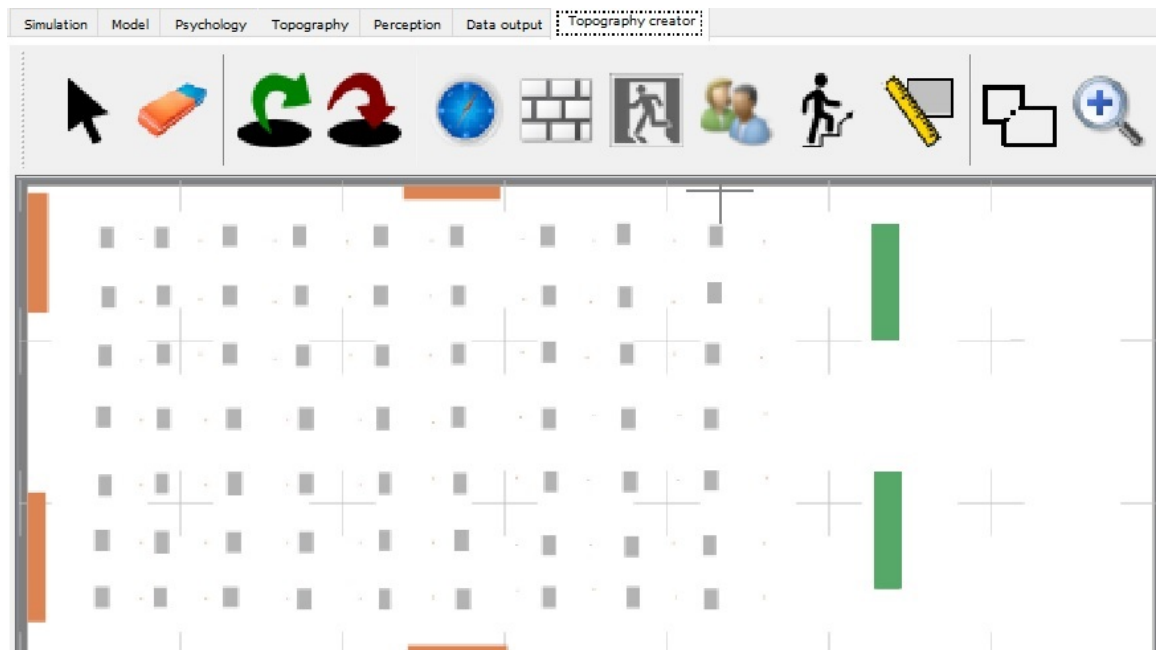


Figure 3.29: A classroom scenario with 63 agents

3.4 Classroom Scenario

```
Text(0, 0.5, 'Number of Suceptible and Infected')
```

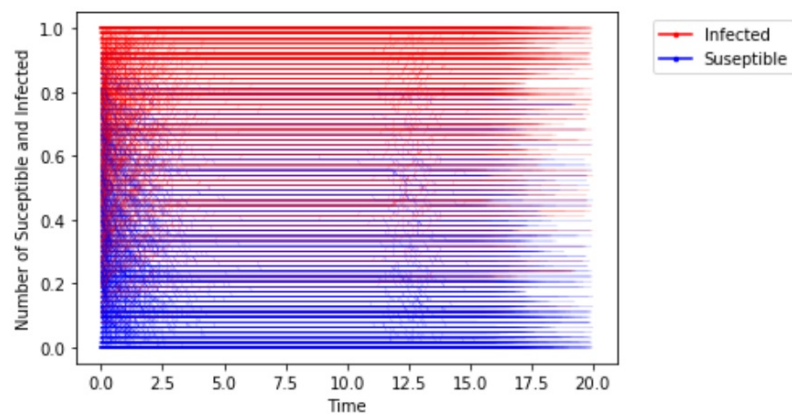


Figure 3.30: Classroom Training Data

The example of the classroom of 63 people, the simulation was set to run for 200 seconds, although most agents exited the scenario by 185 seconds. It may be helpful to get a sense

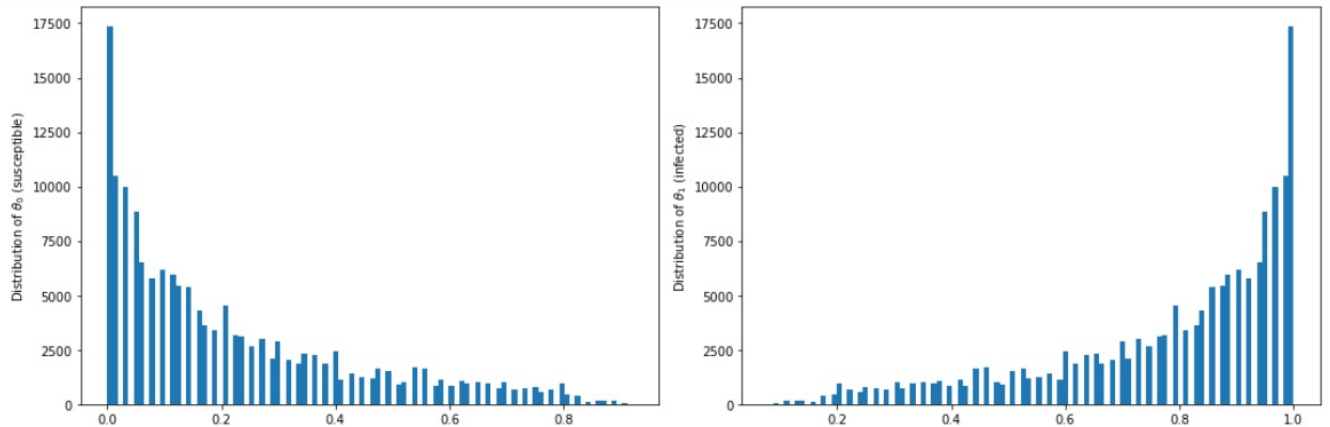


Figure 3.31: Classroom Training Data Distribution

of the data by first view the group counts via some spreadsheet that can read csv, so one can first be familiar with for one iteration the agents spend in a scenario, what is their status over time.

To create the training data, we run the scenario for 4 separate days. Each day takes the resulting infectedRate as the input for the next day's initial infectedRate. We do this for 250 times creating 1000 files for training named "group counts". It is not easy to open 1000 csv files separately therefore we look at them together on 3.30 it shows the fraction of people infected and suseptible over time. The time on the horizontal time is equal to Vadere Time divide by the time reduction factor.

We can also view the distribution of input data points. The histogram below is number of occurrences versus rate for 0 (susceptible) and 1 (infected) as in 3.31

3.4.1 Testing Data

For the testing data, we also set the number of days to 4, and number of iteration of the outerloop to 100. Creating 400 files for testing. While for the training data contain a whole

3.4.2 Traning Validation and Loss

Our final trainig loss is -1.236, final validation loss is around -1.7, note higher training loss than validation loss. With the loss information alone we should not yet conclude it is acceptable or not. Such judgement is better realized after having a look at true vs approximated paths.

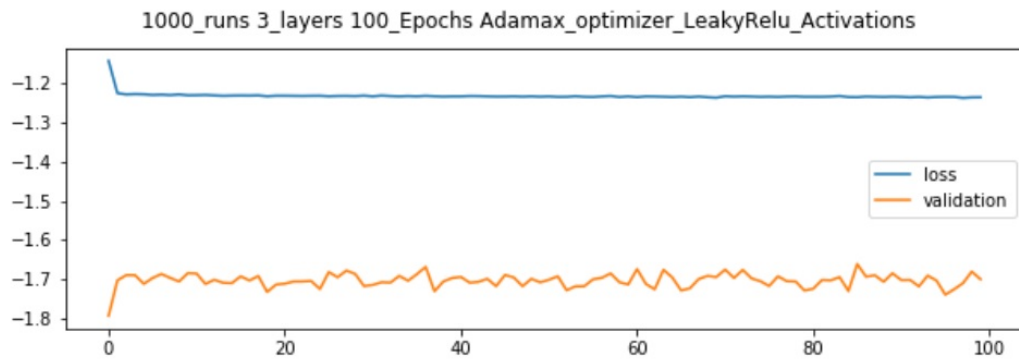
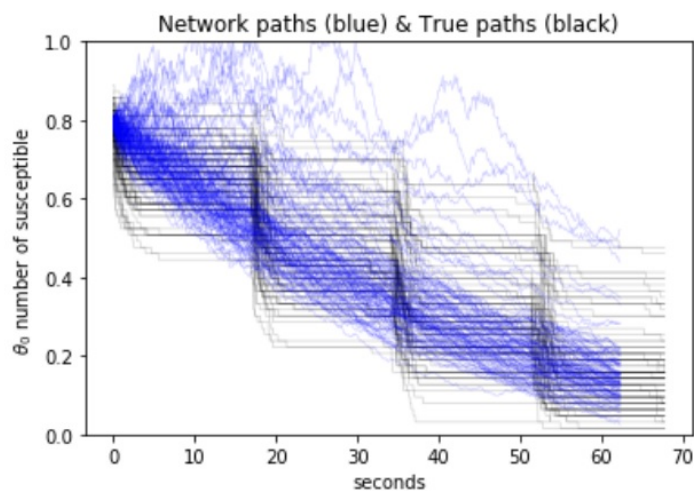
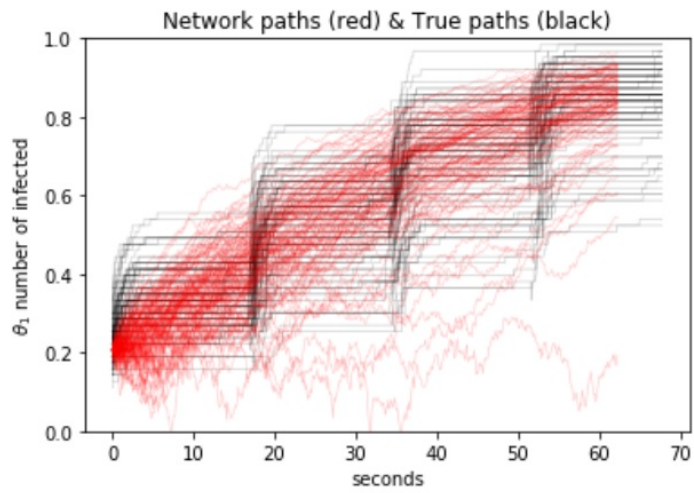


Figure 3.32: Classroom Training and Validation Loss

3.4.3 True vs Approximated SDE Paths and Discussion



above graph shows the number of susceptible vs time in seconds, each day in the scenario is about 180 seconds, with a time reduction factor of 10 result in roughly 18 seconds. We run the scenario for 4 days, therefore the horizontal axis has a length of roughly 70. We can see the true paths (black), and Network paths(blue) overlaps quite well, offering a good approximation of how many people are infected before and after each day in the class scenario. So if one is looking for a path independent initial vs final macro state this offers a good approximation. For micro data, how each agent infect one another, the network does not necessary represent the true dynamics well, despite again getting the correct numbers. This may be due to the noise being chosen.



Since the classroom is also a scenario that spawns agents over time instead of all at once. The remedy of restructuring group counts would have also very likely worked here.

4 Conclusions

In conclusion, when it comes to infection spread due to pedestrian proximity to each other. There is more risk in the entrances being a bottleneck and hotspot for spreading the disease, versus open spaces and arranged seating with distances, are less likely to be the main contributor to disease spreading.

The established system joining agent based modelling, SDE, and neural networks showed competence in learning the stochastic behavior of infectious disease in simulated environments

Our main contributions to the project are two things. We designed scenarios that show no compromise and put the system under heavy tests. We tested a scenario that contains 7 times more people, simulation time 10 times longer in each scenario, while only requiring half the amount of training data. The bottleneck scenario was first tested with 250 training data, as time progresses we start to reduce the amount of training data while see no compromise in results. The low training and validation loss suggest the amount of training data can be reduced even further. Based on the feedbacks from all aspects of the experiment, it is possible to design scenarios bigger with more people, longer in duration, and more complex.

4.1 What can be improved in the future

Include recovery Rate in the studies, the current experiment contains only susceptible and infected rates. It is fine if the focus point is if people went to one class and get out, if people visit the train station once to take a train, what is going to happen. If in reality people do not recover from the interested disease in a few minutes, few hours, or a day. For example, a student being infected with the disease would not recover while still sitting in the same class. For future studies, it would be nice to design scenarios which one can fit into recovery.

Include social distance, mask usage, in the scenarios for comparative studies. The scenarios considers a set infection likelihood, however, in train stations and classrooms people use masks and keep social distance factor, these factors can be included without changing the infection likelihood to query into their effects.

Joining multiple scenarios. So far our model is based on the same group visiting the same scenario recursively. In reality, people may take a train, then take a bus, visit a classroom, and go home. We can call one scenario, send the results of the scenario to a different scenario instead of calling the same scenario again.

Instead of the infection model is based agents proximity to each other. It does not cover the case of what if an agent leaves contaminated substance on a table, and the next agent sat on the same table. A model which can consider something follows the path of something else, there is infection. or a more complex one if something stays at some spot for an amount of time, it leaves say a cloud of infection to whoever passes it.

References

Bibliography

- [1] Benedikt Kleinmeier, Benedikt Znnchen, Marion Gdel, and Gerta Kster. Vadere: An open-source simulation framework to promote interdisciplinary understanding. *Collective Dynamics*, 4(6), 2019.
- [2] Michael J. Seitz and Gerta Koster. Natural discretization of pedestrian movement in continuous space. *Phys. Rev. E*, 86:046108, Oct 2012.
- [3] James A Anderson. *An introduction to neural networks*. MIT press, 1995.
- [4] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.
- [5] Felix Dietrich. Learning effective stochastic differential equations from microscopic simulations: combining stochastic numerics and deep learning, 2021.
- [6] D. J. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Review*, 43(3):525546, Jan. 2001.
- [7] Munich Central Station Map, Deutsche Bahn AG, <https://www.bahnhof.de/bahnhof-de/bahnhof/M-C3-BCnchen-Hbf-1023144>
- [8] Long distance vehicle lexicon. data.deutschebahn.com Deutsche Bahn AG. June 2021.
- [9] Azmat A. Identification of Stochastic Differential Equations with Artificial Neural Networks. September 2021

Bibliography