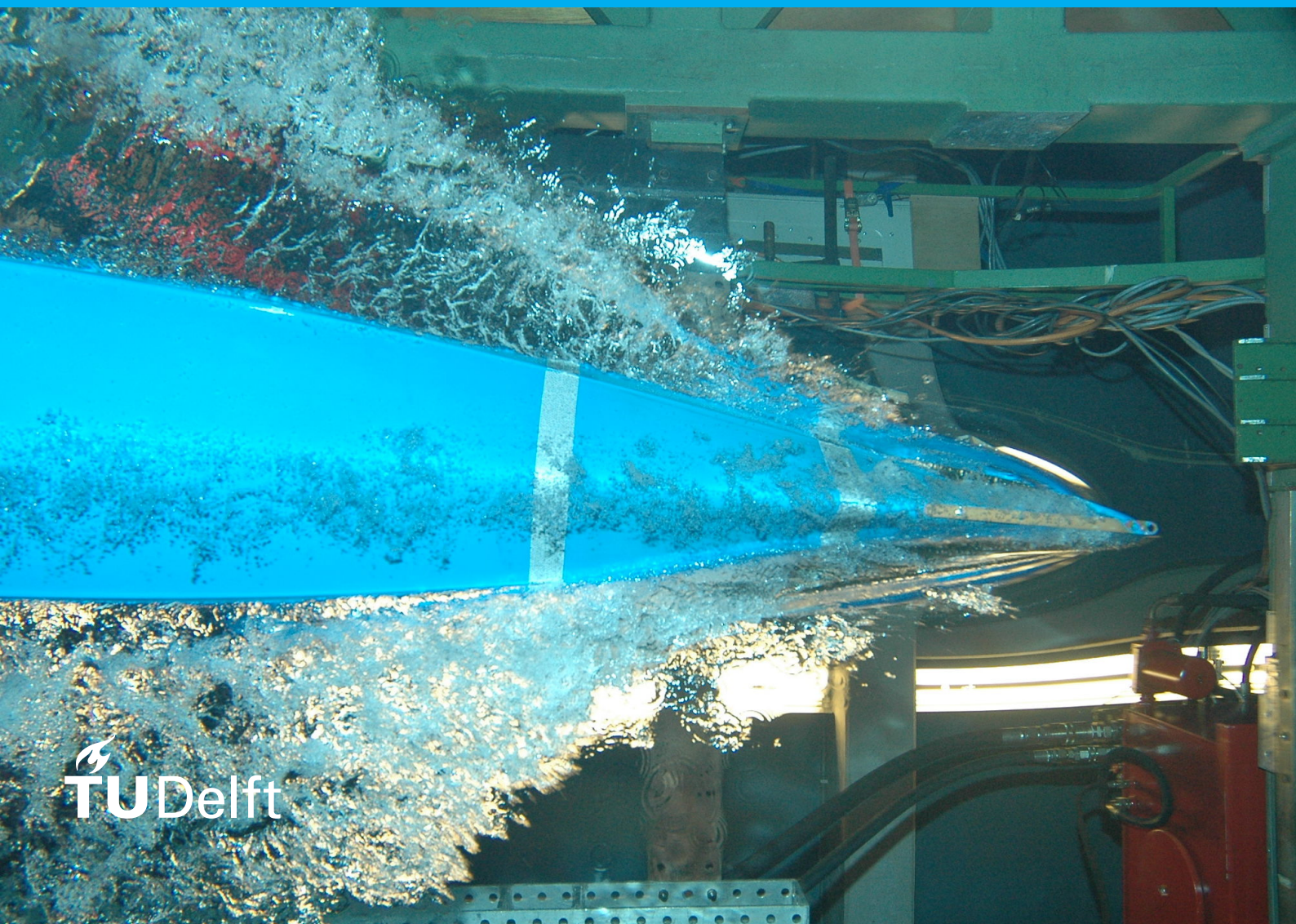
Applying Machine Learning to Learn System Dynamics Models for Urban Systems

Rukai Yin



TU Delft

Applying Machine Learning to Learn System Dynamics Models for Urban Systems

by

Rukai Yin

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 8, 2020 at 10:00 AM.

Student number:	4837371	
Project duration:	October, 2019 – July, 2020	
Thesis committee:	Dr. N. Yorke-Smith,	TU Delft, supervisor
	Dr. E. Isufi,	TU Delft
	Dr.ir. P.W. Heijnen,	TU Delft
	Arie Voorburg,	Arcadis

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

Doing a Master thesis project has been a challenging and interesting journey where I learned more than I have expected. Although spending half of the time staying at home because of the pandemic, it is one of the most valuable time in my life. I enjoy very much the past two years studying at the Delft University of Technology, in Delft, the Netherlands. This whole experience wouldn't have been possible if it weren't for the following people which I would like to thank.

I would like to express my gratitude for Dr Neil Yorke-Smith for being my daily supervisor and for his helpful insights throughout my research. I enjoyed his encouragement and innovative ideas on tackling many challenges. English is not my mother tongue, and it is a great challenge to finish this thesis report. Neil has offered a lot of help in this process. I would have ignored so many mistakes without him. I also want to thank Arie Voorburg for bringing the first idea of this work. Being an expert in urban innovations, he has provided great insights into cities which I would have never learned in Computer Science.

Working from home wouldn't be so smooth as it went without the accompany of Zhe Luo and Zhao Yin. I have been down and unproductive during the confinement, and it was their encouragement that kept me moving forward.

I would also like to thank Tómas Þorbjarnarson for providing great feedback and proofreading to this work. We have known each other since the beginning, and he has been a great teammate and a true friend.

And last but not least, I would like to thank my family and friends who supported me and gave me the motivation to complete this thesis.

Rukai Yin
Delft, June 2020

Abstract

System Dynamics (SD) is an approach to study the nonlinear behaviour of complex systems over time. SD models provide a high-level understanding of the system and aid in designing policies to achieve specific system behaviours. Conventional SD modelling requires an intensive amount of time, human resources and effort. Applying Machine Learning (ML) techniques benefits the modelling process in saving on resources. It also has the potential to provide insights into the system and prevent subjectiveness of the modeller. This work proposes two methodologies, EvoNN and EvoESN, to learn SD models automatically for the urban system from observations under different levels of prior knowledge. EvoNN solves the automated equation formulation task for a Causal Link Diagram (CLD) and annotates it with Shallow Neural Networks (SNNs) as surrogate equations. The annotated CLD can be further used in simulating the system behaviour. We provide experimental results on a real-world urban system in Amsterdam as well as the evaluation of the simulation results. The second methodology, EvoESN learns both the structure and the quantitative relations in the model without the prior knowledge about the structure. Trained using observation data, the EvoESN produces satisfactory results on the real-world urban system. We further incorporate the judgement from the domain expert to evaluate the learned model. Applied on a more complex system, EvoESN shows solid reliability and scalability to handle large datasets. Both EvoNN and EvoESN stand as promising supportive tools for SD modellers and remain robust even when lacking system observations.

Contents

1

Introduction

Modelling a complex system involves building conceptual models or mental models that provide a high-level understanding of the system. System Dynamics (SD) [19] is a well-developed and widely applied approach to study the nonlinear behaviour of complex systems over time. In SD modelling, Causal Loop Diagrams (CLDs) are one of the main tools to represent conceptual models. They often work as an intermediate model since they depict only qualitative behaviours of the system. A CLD can be converted to simulation models, for instance, a Stock and Flow Diagram (SFD), in which the nonlinear behaviour over time is simulated.

Conventional SD modelling involves several steps, each of which needs particular care to ensure the effectiveness of the process. Sterman [32] breaks down the SD modelling into five steps: problem articulation, formulation of dynamic hypothesis, formulation of a simulation model, testing, and policy design and evaluation. More importantly, SD modelling is an iterative process, meaning each step takes place more than once and may jump to any other steps. However, this iterative process requires the modeller to have comprehensive knowledge about the system and interact with various stakeholders as well. It takes a considerable amount of time, human resources and effort to obtain the resulted model.

The use and development of Machine Learning (ML) techniques have grown significantly over the years. Scholars, researchers and professionals have been studying on possible means aiding in accelerating the SD modelling process in different stages using such techniques. In addition to saving on resources, incorporating such methods potentially complements the subjectiveness of human modellers in the process. Chen and Jeng utilized the Recurrent Neural Network (RNN) to represent SD models [11] and learned a biological system model [24]. Together with Tu [13], they proposed an evolutionary method for policy design in which the task is to manipulate a set of variables such that the overall system behaviour fits the desired pattern. Drobek et al. [16] attempted to annotate the causal relations in the CLD with Neural Networks (NNs). Abdelbari and Shafi [2] proposed to learn conceptual models using the echo state network and further optimised the network in [4]. Although the proposed methodologies have reported promising results on various system models, there still lack applications in real-world cases, especially on urban systems.

Urban systems abstract the complexity of the city by inspecting human activities and how they influence each other. The literature shows evidence that people are incapable of assessing the effects of their actions in a complex system [29]. It would be interesting to build SD models to assist us in understanding the behaviour of urban systems. This work intends to visit ML techniques that aid in accelerating the SD modelling process and apply them on the dynamic modelling of an urban system. In particular, we explore various means for both qualitative and quantitative modelling under different levels of prior knowledge about the urban system. We take a real-world urban system in Amsterdam, the captical and the most populous city of the Netherlands, and apply the automated model learning methods on it.

The main research question of this work is:

How can machine learning techniques enhance both qualitative and quantitative dynamic modelling of an urban system?

In order to answer this question, the following sub-questions have been formulated:

- **RQ1:** *Can we model and simulate a dynamic urban system based on specialists' knowledge?*
- **RQ2:** *Can we learn a qualitative and quantitative urban model from the observations?*

The RQ1 takes advantages of the prior knowledge, a reference model of the urban system received from the domain expert, which makes it convenient to create a CLD to depict the system structure and behaviour and interpret the system qualitatively. The next step is to build the quantitative relations between system variables based on the CLD created, the process of which is called equation formulation. SD modellers used to formulate simple mathematical equations in this process which often requires extensive domain knowledge and intensive manual work.

To answer this question, and help the SD modeller accelerate the process, this work applies ML techniques to annotate the CLD and further simulate the real-world behaviour. EvoNN, an evolutionary shallow neural network, is designed for the purpose. A shallow neural network (SNN) has the ability to learn underlying features and simple relations from the data and to predict output values given the

input. An evolutionary algorithm (EA) is adopted to optimise the neural networks and fine-tune hyperparameters. We then show the use case of the EvoNN on the urban system model and evaluate the results.

The RQ2 removes the existence of any prior knowledge to the system and thus, both the model structure and the quantitative relations between system variables will be learned from observed data. We build on groundwork in the literature [1, 2, 4]. The authors show the possibility to learn CLD-like models using the Echo State Network (ESN). We adapt their work and introduce the EvoESN, an evolutionary echo state network, for automated model learning on urban systems. The model is encoded with the ESN, which is then trained to fit the behaviour of the system. The hyperparameters of the ESN is optimised with the help of the EA. Both qualitative and quantitative relations between system variables will be learned after the training. A series of experiments conducted on two cases show that the EvoESN produces satisfactory results in terms of the output error. It performs better and takes less time to solve, comparing to the non-optimised ESN. We further incorporate the judgement from the domain expert to evaluate the learned model.

The remainder of this thesis is organised as follows. In chapter 2 background to this work is provided. This is followed by an overview of related work in chapter 3, including the selection of the most suitable techniques for answering the research question. chapter 4 explains the technical details in answering the research question, including two case studies. Next, in chapter 5, the experimental setup is illustrated, and its results are evaluated and discussed in chapter 6. Finally, chapter 7 concludes this work and discusses future work.

2

Background

This section provides background knowledge of related techniques and algorithms used in this report.

2.1. System Dynamics

System Dynamics is a well-developed approach and dicipline developed to understand the behaviour of complex dynamic systems over time using mathematical models. It is first introduced by Jay Forrester in the 1950s [19]. Although SD is primarily developed for systems that can be assumed to be closed, it is often used to deal with complex real-world issues that are not fully closed or entirely open [30]. This work assumes all dynamic systems are closed which means that the actions of the system depend on the results from previous actions, not exogenous forces. It is important to make this assumption and limit the system boundaries as we try to learn the model structure from system observations which involves only endogenous variables. The reservoir computing in ESN takes advantages of this assumption as well.

SD defines, analyses, understands and solves issues through SD models which are simplified representations of issues or systems. SD models consist of variables and links between them. Among all types of SD models, we study two in this work: Causal Loop Diagrams and Stock and Flow Diagrams.

2.1.1. Causal Loop Diagrams

Causal Loop Diagrams provide means for model conceptualisation and model communication. They are often qualitative models that consist of variables, causal links between them and feedback loops. The polarities attached on the links indicate how a variable react to the change of its cause(s). An example of CLD is provided in Figure 2.1. The increase of the rabbit population will, ceteris paribus, cause an increase in the rabbit births and a decrease in the rabbit death.

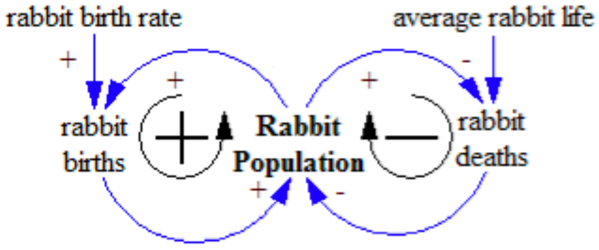


Figure 2.1: Example CLD, figure taken from [30].

2.1.2. Stock and Flow Diagrams

Stock-Flow Diagrams provide means for simulation purposes which typically consist of stocks, flows and causal links between them. The stock variable acts as a reservoir and accumulates flows over time. It is also considered as the output variable since its values are often of interest. An example of SFD is provided in Figure 2.2. The stock variable in this example is the rabbit population, and its value over time can be simulated.

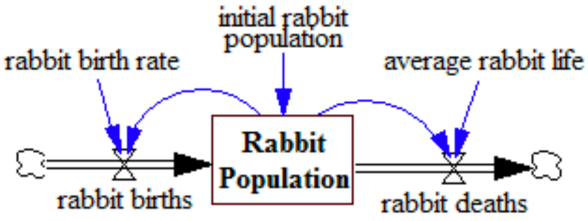


Figure 2.2: Example SFD, figure taken from [30].

2.2. Recurrent Neural Networks

Recurrent Neural Networks are a family of NNs that process sequential data. For more information, readers can refer to [31].

2.3. Echo State Networks

The Echo State Network is a novel variant of the RNN, introduced by Jaeger [23] in 2001. An independent work [28] also introduces the basic idea of echo state, but it emphasises continuous-time networks. Although in the real world, the system evolves in a continuous manner, the model we try to learn is discrete-time. Jaeger proposes ESNs based on discrete-time RNNs. The activation state $x(n)$ of an RNN is a function of the input history and thus can be understood as an "echo" of the input history. The perspective taken, according to Jaeger, is of mathematics and engineering, where an RNN is seen as a computational device for realising a dynamical system.

The basic architecture of the ESN is depicted in Figure 2.3, which consists of K input units, N internal units (dynamic reservoir) and L output units. The input layer is connected to the dynamic reservoir through the input weight matrix $W_{in} \in \mathbb{R}^{N\times K}$. Within the reservoir, neurons are connected to each other through the reservoir weight matrix $W \in \mathbb{R}^{N\times N}$. Both the input layer and the reservoir can connect with the output layer through the output weight matrix $W_{out} \in \mathbb{R}^{L\times(N+K)}$. The feedback connection from the output layer to the reservoir is through the feedback weight matrix $W_{back} \in \mathbb{R}^{N\times L}$ [23].

During training, all weight matrices except W_{out} remain unchanged which ensures that, if the network runs for a long time, its state will be uniquely captured by its input/output signals and given a new input signal, it can generate the suitable corresponding output one [25].

To ensure this property, one has to design W properly. A weight matrix W_0 is generated randomly $\in [-1, 1]$ with a given connectivity probability parameter ρ. W_0 divided by its maximum eigenvalue λ forms a weight matrix W_1 which is multiplied by the spectral radius $\alpha \in [0, 1]$. Equation 2.1 formalises above process:

$$W = \frac{W_0}{\lambda}\alpha. \tag{2.1}$$

W_{in} and W_{back} are initialised randomly with a scale value of δ. Once all weight matrices except the output one are initialised, the network reservoir is update using the following equation:

$$x(t+1) = (1-\gamma)x(t) + \gamma f(W_{in}u(t+1) + Wx(t) + W_{back}y(t)), \tag{2.2}$$

where $x(t+1)$ and $x(t)$ are the activation values of the reservoir's neurons at times $t+1$ and t, respectively, $u(t+1)$ the input signal at time $t+1$, $y(t)$ the network output at time t, $f(.)$ the activation function and γ the leaking rate [23]. This equation is applied for the number of times equal to the size of the training data set.

The reservoir's states, $x(t)$, are collected in a state matrix M. The output weights are then updated using the following equation

$$W_{out} = M^{-1}Y^{target}, \tag{2.3}$$

where M^{-1} is the morse-pseudo inverse for M and Y^{target} is the output for the training data set.

Finally, the network output $y(t)$ at time t is computed as

$$y(t) = f(x(t)W_{out}). \tag{2.4}$$

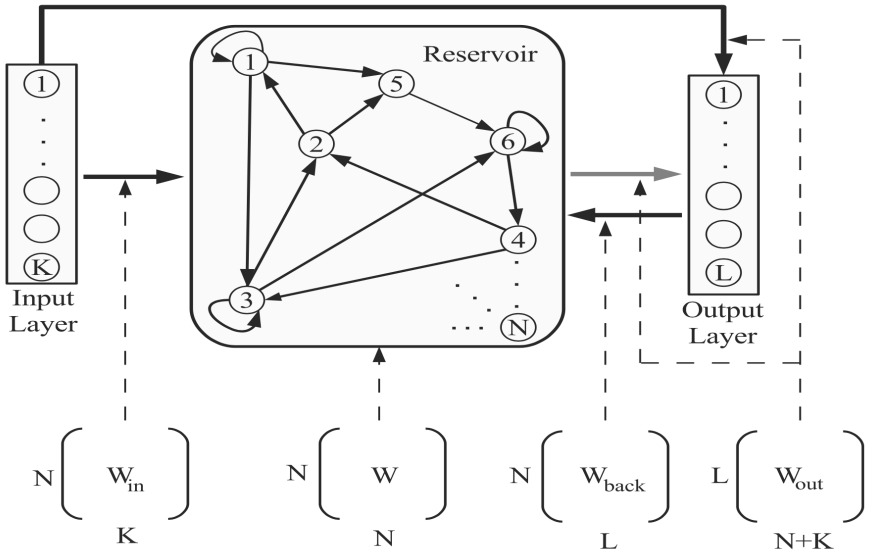


Figure 2.3: Standard ESN architecture. Figure taken from [4]

3

Literature Review

There are a few attempts to learn simulation models from system observations. Such models typically provide details and specify equations and parameters. Chen and Jeng have been working on this topic for a long time. They proposed a method to represent SD models with RNN, in 2002 [11]. Jeng et al. [24] in 2006 took advantage of the representation and learned a biological system model from time-series data. An SD model is represented by an RNN by encoding each variable as a hidden unit and each constant as a weight parameter. A genetic algorithm is then applied to train the network for the purpose of parameter calibration in which parameters are tuned so that the simulation behaviour fits the reference model. Finally, the trained network is converted back to SFD.

Chen and Jeng utilise the representation and focus on policy design and parameter optimisation in the following years. Together with Tu [13], in 2011, they proposed a policy design approach similar to [24] but capable of evolving the structure of the network simultaneously with parameter calibration powered by genetic algorithm. This approach still needs to convert the output network back to SFD, which is computation intensive.

Abdelbari and Shafi's team have been trying to learn a simulation model without the existence of the reference model. The behaviour of the system is the learning target. To this end, they have utilised EAs and ML algorithms. They proposed an approach in 2015 [5] and started with the Genetic Programming (GP) and the embedding reconstruction technique under full and partial observations of the system. In 2017, they proposed another GP-based method [3] to learn the complex dynamical system at large scale using the prior knowledge of variables dependencies. The GP-based learning method has proven its ability to learn the underlying structure of the system.

Abdelbari and Shafi also studied on encoding the simulation model with a NN in 2016 [1], 2017 [2] and 2018 [4]. Similar to Chen and Jeng's work, they chose a variant of RNN, the ESN [23] because of the similarities in the structure of an ESN and an SD model. The idea is to encode an ESN's reservoir network with a known number of nodes, equal to the number of variables in an SD model, and then train the ESN so that the network output fits the system's behaviour. However, the structure of the learned network might not be reliable since the optimisation lies in the output of the network instead of the structure, which is why the authors state their task as "learn causal loop diagram-like structures from observed data [2]". Combined with EAs, an evolutionary ESN is proposed which is able to learn both the model structure and the behaviour [1, 4].

Instead of learning simulation models, Drobek et al. focused on learning CLDs from system observation data. A CLD describes the causal relationships among variables and feedback loops. It can also guide the modeller in building simulation models. Drobek et al. [15] (2014) attempted to generate "appropriate" CLDs that are readable and informative for modellers. They used the Pearson product-moment correlation coefficient to analyse the dependency between two variables. The dependencies were then used to indicate the causal relationships among variables. Although they could not determine the causal relationship directly, using dependencies instead served the goal well. The proposed approach was performed under the full observation of the system in the business domain and required a business ontology and time-series data of all variables, which may lead to limitations when applying the approach to other domains.

Drobek et al. [16] (2015) took a step further and introduced the NN to perform automated equation formulation for a given CLD. Unlike other encoding methods where a simulation model is encoded with a single NN, this work trained a NN to approximate the behaviour of a single variable with the help of EAs. The learning outcome was then a set of learned NNs served as function surrogates that memorise the historical data and can predict the future development of variables. Taking a CLD and historical data of all variables as input, the proposed method was able to incorporate both the variables dependencies and their historical information. Results showed that the outcome of the proposed method was sensitive to the oscillation in variables behaviour and NN configurations. But the limitations remain. A prior CLD and time-series data of all variables are required.

Zhao [35] in 2019 proposed a platform for automated model conceptualisation which included structure generation and parameter calibration. The author took advantages of not only time-series data but additional information from the user to enhance the modelling processing. Such information was mainly the prior knowledge about the system and the problem. Allowing to interact with additional information, the model conceptualisation showed the ability to work under none, partial and full observations of the system. However, Zhao only ran experiments with a simple case taking into account several variables.

So far we have seen different techniques relating to structure generation and parameter calibration. To apply structure generation in the domain of urban systems, this work tries to adapt Drobek's methods

in [15, 16] since the paramilitary understanding of the system is obtained. To explore structure generation under full observations of the system but without the existence of a reference model, techniques proposed by Abdelbari and Shafi in [1, 2, 4] will be studied in this thesis since they are less complex compared to Chen's works [11, 13, 24] and provide experimental results on several case studies.

4

Approach

We break down the main research question and answer the sub-questions separately. This chapter approaches the solution in detail and provides methodologies and technologies used in the solution.

4.1. Annotating CLD Using Evolutionary Shallow Neural Network

Modellers study the system qualitatively through CLDs in which variables and causal links reveal the system's behaviour. The lacking of quantitatitive relations leads to a limited understanding of the system. To overcome the limitation of CLD and study the system quantitatively, we perform CLD annotation. The CLD annotation task is described as follows: given a set of variables S and a variable A from a closed system, where all the variables in S lead to direct causal links to A and there is no other variable in the same system that leads to a causal link to A and is not in S, build and train a simple NN to fit the causal relations between variables in S and A. In other words, we annotate the CLD by annotating all the incoming causal links for each variable unless there are none. The annotated CLD is then used to simulate the system, answering the RQ1.

The final annotated CLD consists of all variables and EvoNNs as the representation of the quantitative relation between them. The behaviour of the system is often of modellers' interest and it can be efficiently deduced and simulated given the initial state of the system. An overview of the process of CLD annotation is depicted in Figure 4.1. More details are provided below.

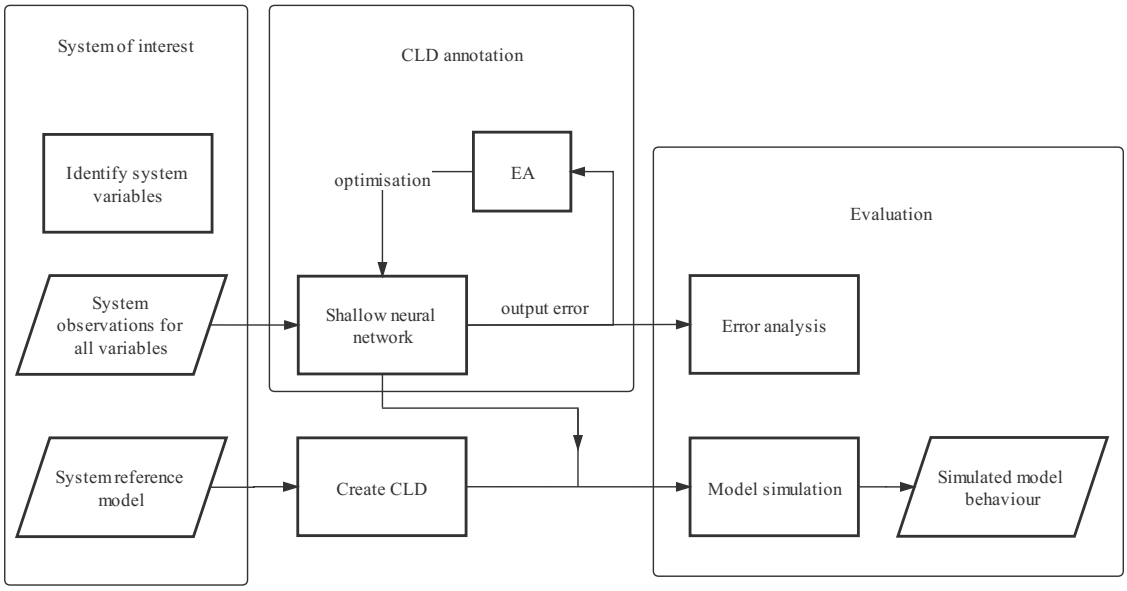


Figure 4.1: Overview of CLD annotation using EvoNN.

4.1.1. Shallow Neural Network

We design a SNN to replace the simple mathematical equation in the equation formulation. A basic structure of the SNN is depicted in Figure 4.2. The input layer consists of multiple neurons, the number of which is equal to the number of variables in S. The output layer consists of one neuron, corresponding to the variable A. A fully connected hidden layer lies between the input and the output layer. After training the network, the weight parameters are learned and able to preserve the underlying relations between the input neuron and the output neuron, which then serve as the annotation of the quantitative relations between S and A.

4.1.2. Optimisation with EA

Training a neural network can be just as hard as designing one. Fine-tuning hyperparameters plays a vital role in both processes. Its effectiveness affects the results of the neural network significantly [21]. We handle this process as an optimisation problem which is solved in the help of EAs. The set of parameters to be optimised in an shallow neural network is

$$S_{NN} = \{\eta, N\}, \eta \in \mathbb{R}^+, N \in \mathbb{Z}^+, \tag{4.1}$$

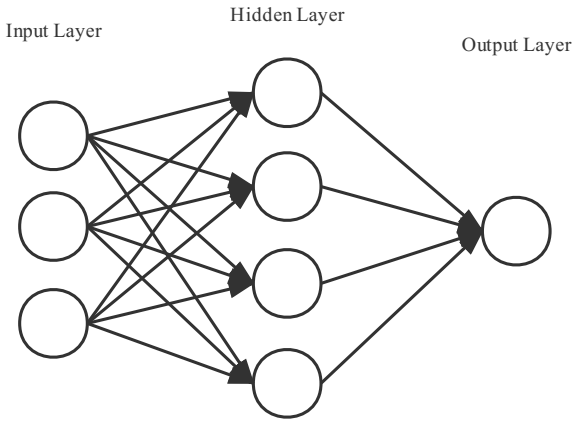


Figure 4.2: Structure of a shallow neural network.

where η is the learning rate of the neural network and N is the number of hidden units in the hidden layer. Varying the number of hidden layers may affect the performance of the network significantly, but it should be limited within a small range, e.g. one to three hidden layers. Doing so also verifies our assumption that we can annotate the CLD using simple and shallow neural network.

There are a few approaches to optimise hyperparameters: random search [7], Bayesian optimisation [8], evolutionary optimisation [8] and etc. They are all shown to obtain desired results in various cases. Evolutionary optimisation is a global optimisation method which is easy to understand and implement. It is also capable of producing fine results with limited training data available [1]. We choose to utilize the EA to optimise hyperparameters in the SNN.

EAs are population-based algorithms mimicking the biological world of natural selection and survival of the fittest. Starting from an initial state, the population evolves iteratively by applying selection and mutation operators and generates offsprings in each step. With the help of EAs, the task of CLD annotation becomes the task of optimising SNNs to fit the causal behaviours between variables in the system.

4.2. Learning Conceptual Models Using Evolutionary Echo State Network

The RQ2 describes a real-world scenario when prior knowledge about the system is no longer available. The modeller needs to gather all information and build the SD models from scratch. It often requires comprehensive knowledge for the modeller. We intend to simplify this process and learn conceptual models from system observations directly. Both the structure and the quantitative relations will be learned. We approach the task of learning conceptual models with an EvoESN which consists of three steps: modifying the ESN to be able to represent the conceptual model with its reservoir, optimising the hyperparameters of ESN using the EA, and evaluate the learned model and the performance of the EvoESN. An overview of the approach is described in Figure 4.3.

4.2.1. Modified ESN

The network topology shown in Figure 2.3 does not impose any condition on W and allows self-connections and cycles for internal units. CLDs and other mental models are typically represented as a graph of system variables which are connected directly to each other based on their causality. Such models generally include feedback loops which are comparable to cycles in a graph topology. These similarities make it possible to adapt ESNs with necessary modifications for learning mental models. In the work [1, 2, 4], ESNs are used to encode the system variables and proved to be reliable to simulate the system's behaviour after the training. We follow their work and modify the standard ESN to our ends.

First, we fix the number of reservoir neurons to the number of system variables. Abdelbari and Shafi [2] finds it is able to not only produce the target output's behaviour but also learn a sparse causal model

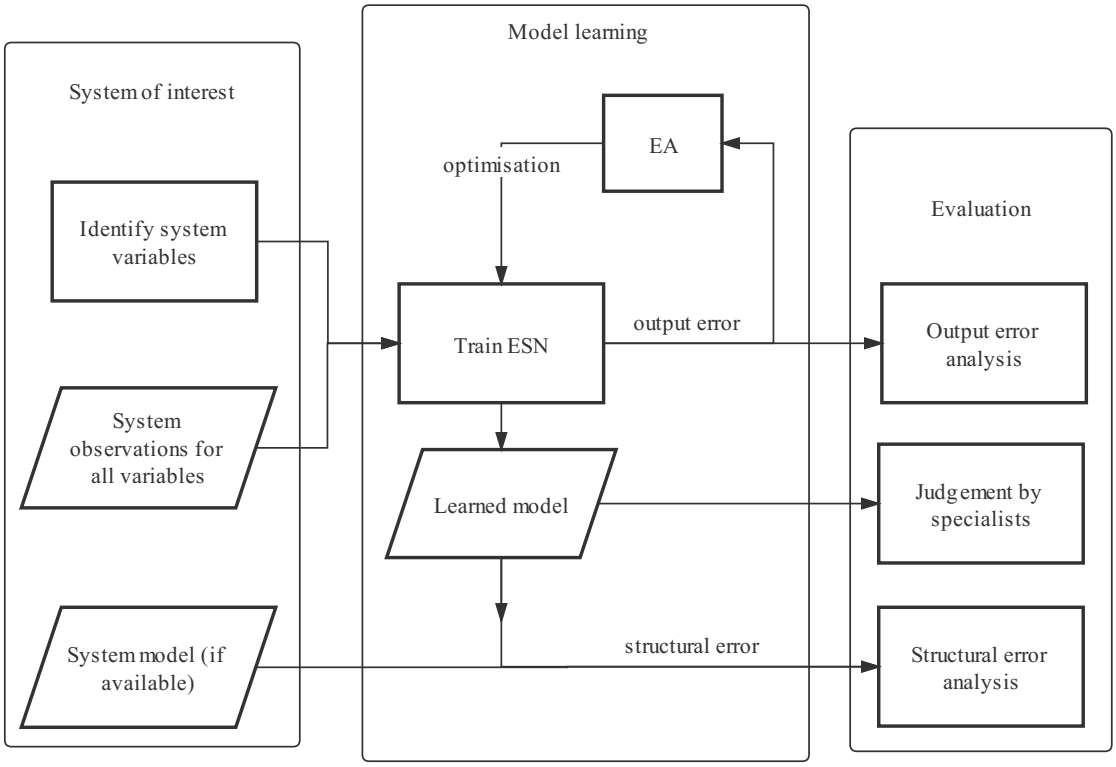


Figure 4.3: Overview of model learning using EvoESN.

that closely match the target model structurally. Second, we remove self-connections in the reservoir since self-feedback loops do not exit in SD models. For the reservoir weight matrix W, this means all diagonal values are set to 0. Finally, we remove the direct connections between the input layer and the output layer. These connections are considered to be optional and only marginally improve the network's performance, if at all, but with a significant increase to the network architecture and computation requirements [4]. The customised ESN after modification is shown in Figure 4.4.

Now the ESN is customised so that its reservoir is able to represent a conceptual model. There is still a flaw in its structure. A conceptual model is often shown and studied as a directed graph or a connected directed graph, to be precise. The current design of ESN is unable to guarantee that the reservoir will be a connected graph and thus may produce infeasible solutions. We overcome this impediment by manipulating the initialisation of the reservoir weight matrix W. After the initialisation, the weight matrix W is considered as a $0-1$ adjacency matrix to a graph. We randomly add at most two extra edges that connect unconnected vertices together to make it connected if W does not represent

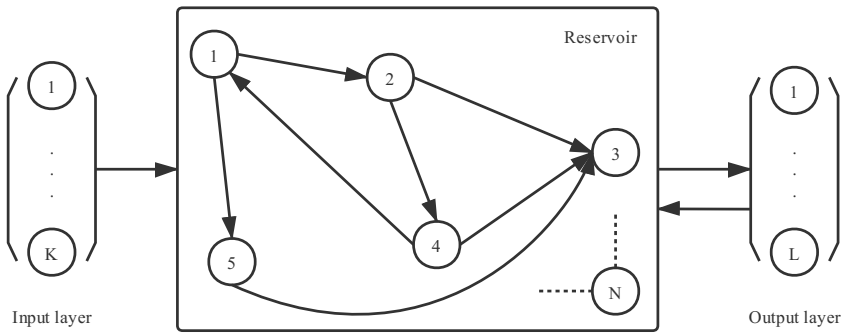


Figure 4.4: Customised ESN architecture.

a connected graph. The resulted matrix W can still be unconnected after the manipulation and thus leads to infeasible solutions which will be discarded eventually.

To represent a conceptual model, the reservoir weight matrix W tends to be sparse. It is generated randomly with some small probability, which makes it normal to produce infeasible solutions. With the manipulating of W, we preserve and add another possible structure of the learned model to the outcome, which increases the possibility to reach the desired solution.

4.2.2. Optimisation with EA

Like SNN, it also requires tuning hyperparameters in the training of ESN. The set of parameters to be optimised in an ESN is

$$S_{ESN} = \{\alpha, \gamma, \delta, \rho\}, \alpha, \gamma, \delta, \rho \in \mathbb{R}^+, \tag{4.2}$$

where α is the spectral radius of the reservoir weights matrix, γ is the leaking rate used in the update equation for the network states, δ is the scale of weights for both input weight and feedback weight, and ρ is the connectivity probability for initialising the reservoir weights matrix. The values of them are in a small range, e.g. $\alpha, \gamma, \delta, \rho \in (0, 1)$.

It is straightforward and intuitive to come up with a brute force way that tries all the combinations of these parameters. However, the number of combinations is infinite which makes the optimisation infeasible. It is still time-consuming even with a small amount of increment on the parameters. Similar to what we do in the previous section, we apply the EA to optimise the parameters in S_{ESN}. It varies the parameters in the given range and allows tiny changes on them. One of the advantages of doing so is to avoid local optimal and tend to find the global optimal or local optimal close to the global one.

Several works from the literature involve the optimisation on ESN through EAs. Ferreira and Ludermir [17], Ferreira et al. [18], Liu et al. [27] have tried to optimise different components of ESN with EAs and reported better network performance than those using default settings.

4.2.3. Model Similarity

There are no universal evaluation measurements or methods for the learned model. The output error of the ESN reflects how the trained ESN fits on the data. We introduce another measurement, the model similarity, to evaluate the structural error of the learned model, i.e. how far the learned model is away from the reference model. The reference model is created by the modellers and domain experts and we assume it to be the ground truth despite any flaw in it. To measure the structural distance between two conceptual models, we use the distance ratio (DR) from [26] which is also adopted in [4]. The structural error ψ of two conceptual models is then calculated as

$$\psi = \frac{\sum_{i=1}^{p} \sum_{j=1}^{p} |a_{ij} - bij|}{2(p^2 - p)}, \tag{4.3}$$

where a_{ij} and b_{ij} are elements from adjacency matrices representing the two models, respectively, and p is the number of variables in the model which holds the same for both models. ψ measures the average edge difference between two models over all possible edges. Obviously, $\psi \in [0, 1]$ and the smaller its value is, the closer two models are.

Figure 4.5 shows two models of small size. Both models consist of three variables and each model has two directed links none of which are the same. The structural error ψ of them is calculated as

$$\psi = \frac{4}{2 \times (3^2 - 3)} = \frac{1}{3}.$$

4.2.4. Judgement from Domain Experts

The EvoESN method aims at providing a supportive methodology for SD modellers and accelerating the modelling process by reducing human effort. Learning conceptual models automatically from the data, it has the potential to offer additional insights missed by the human modeller. There are mistakes in the learned models, as well, which are unavoidable and may be identified by the modeller. Therefore, we include domain experts' judgement on the learned model into the evaluation process. A domain expert judges a learned model from several perspectives. One can start with the dominating variables

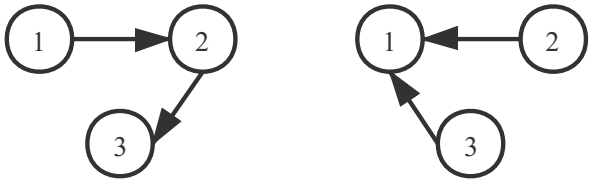


Figure 4.5: Example of model structural error.

and evaluate the way they affect others. Looking at the high level, it is worth evaluating whether the overall structure makes sense. The domain expert may also evaluate whether each causal link makes sense. By taking into account the judgement from domain experts, the proposed methodology becomes a combination of automated learning algorithms by the machine and judgement by a human.

5

Experiment

We have applied different methodologies to answer both RQs and we evaluate the correctness and effectiveness of proposed approaches with two case studies. The first one is an actual case study in Amsterdam, the Netherlands. We have collected as much data as possible from various sources (section 5.3). This case study aims at showing the feasibility of annotating CLDs and learning conceptual models from system observations. To further evaluate the generalisation, flexibility and scalability of proposed methodology EvoESN, we perform a second case study on a more complex model with enormous data available. Since it is not a real-world case, we generate the data from its simulation model, i.e. the SFD. More details regarding the case studies and experimental setup are explained in the Sections below.

We employ a $(\mu + \lambda)$ evolutionary algorithm [6] implemented with DEAP [14, 20] to optimise the network in both the EvoNN and the EvoESN. The $(\mu+\lambda)$ EA outperformes Genetic Algorithm (GA), Particle Swarm Optimisation (PSO) and (μ, λ) Covariance Matrix Adaptation Evolution Strategy (CMA-ES) in different optimisation senarios including low-dimensional cases like ours in section 5.2. In addition, it is well-supported by DEAP and simple for usage. The operator settings in this EA are specified below.

The initial population size is 100, and the maximum generation size is 15. This setting is to generate enough randomness in the initial population while preventing the algorithm from consuming too much memory. Crossover and mutation are known as the essential operators in the EA. A crossover rate that is too high may lead to premature convergence. A mutation rate can be too high to lose good solutions. In each generation, we set the crossover rate to 0.7 and the mutation rate to 0.3. This combination allows the offspring inherits most of the characteristics from the parents while maintaining genetic diversity.

Although most of the settings of the EA are same in both the EvoNN and the EvoESN, the fitness functions are not. The fitness function for EvoNN is

$$f_{EvoNN} = \epsilon, \tag{5.1}$$

where ϵ is the output error of the SNN. The fitness function for EvoESN is the weighted sum of the output error and the connectivity rate of ESN. It is described as

$$f_{EvoESN} = w_1 * \epsilon + w_2 * \rho, \tag{5.2}$$

where ϵ is the output error of the ESN, ρ is the connectivity rate and $w_1 + w_2 = 1$. We perform a sensitivity analysis for different weight combinations listed in Table 5.1 to show how weight combinations influence the outcome of EvoESN.

All methodologies including data preprocessing are implemented in Python 3. The experiments are run on a 6-core Intel i7 @ 2.6GHz and 16 GB RAM.

Table 5.1: Different weight combinations in fitness function for the EvoESN

Weight combination index	w_1 values	w_2 values
WC_1	0.1	0.9
WC_2	0.2	0.8
WC_3	0.3	0.7
WC_4	0.4	0.6
WC_5	0.5	0.5
WC_6	0.6	0.4
WC_7	0.7	0.3
WC_8	0.8	0.2
WC_9	0.9	0.1

5.1. ESN Setup

In a nutshell, two sets of experiments are carried out to verify different hypotheses and the generalisability of different ESN architectures in learning conceptual models. Each set of the experiment will run five times and generate an average result for the runtime. The best-learned model is selected from the one that produces the minimum output error.

5.1.1. Basic ESN

This setting is to test the hypothesis that without proper parameter optimisation, it does not only consume much more time but also produces a worse result to learn conceptual models using ESN. The task is to learn both the structure as well as weights for causal links, which provides a baseline for answering the RQ2. In this setting, we do not apply automated parameter optimisation on ESN. Instead, a brute force way is employed to search for the best combination of ESN parameters. The search space for each parameter is

$$\begin{aligned} &\alpha \in [0.1, 1], && step = 0.1; \\ &\gamma \in [0.1, 1], && step = 0.1; \\ &\delta \in \{10^i\}, i \in [-10, -5], && step = 1; \\ &\rho \in [0.05, 0.95], && step = 0.1; \end{aligned} \tag{5.3}$$

where $step$ is an increment value on the parameter in each iteration. The total size of the search space becomes 6000 and grows exponentially when reducing the step size.

5.1.2. EvoESN

Similar to the aforementioned basic ESN setting, the task in this setting is to learn both the structure as well as weights for causal links. The methodology We apply is the EvoESN described in section 4.2. The (μ + λ) EA is employed to optimise the parameters in the ESN.

5.2. Case Study

5.2.1. Parents Education Impact system

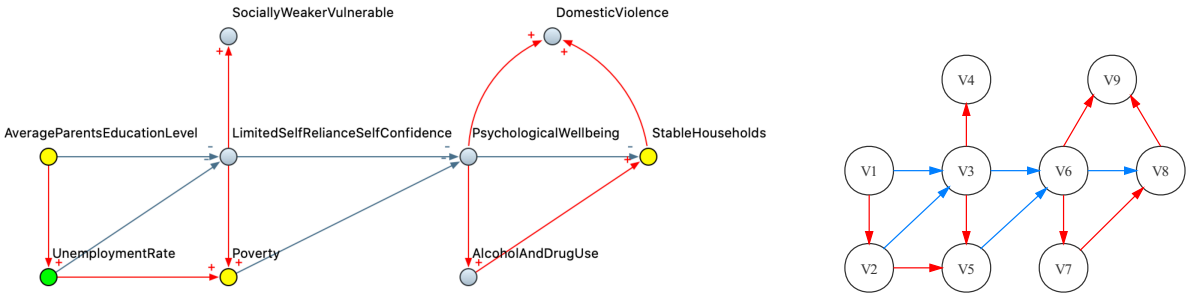
We received a reference model for the parents education impact (PEI) system from the domain expert and create a corresponding CLD depicted in Figure 5.1a. The CLD is built with AnyLogic[1], a simulation software. It contains 9 variables and 13 causal links. Thers is no feedback loop in this diagram. A simplified CLD is shown in Figure 5.1b for better visualisation, where links in red represent positive causal links and those in blue represent negative causal links.

We study the system in the city of Amsterdam, the capital of the Netherlands. The data is collected within the municipality area from 2004 to 2018. In Figure 5.1a, the colour of variables shows the availability of the data. The variable in green, the *unemployment rate*, indicates that the data is available for all years. Variables in yellow indicate that the data is available except for some years. Linear interpolation is applied to construct unknown data for missing years. Data is unavailable for the rest of the variables. We simulate the data for them based on their causal relations with variables whose data is known. More details about the data sources, interpolation and simulation are provided in section 5.3.

In the CLD annotation and the model learning methods, we train the NN using the leave-one-out cross-validation method. The network is trained using all data for a few iterations whose number is equal to the size of the training data. In each iteration, one data point is excluded in turn and serves as the test data. After training, the overall output error of the network is an average of the test error on each data point. Leave-one-out generalises the performance of the trained network on the whole data set and is commonly used when a limited amount of training data is available.

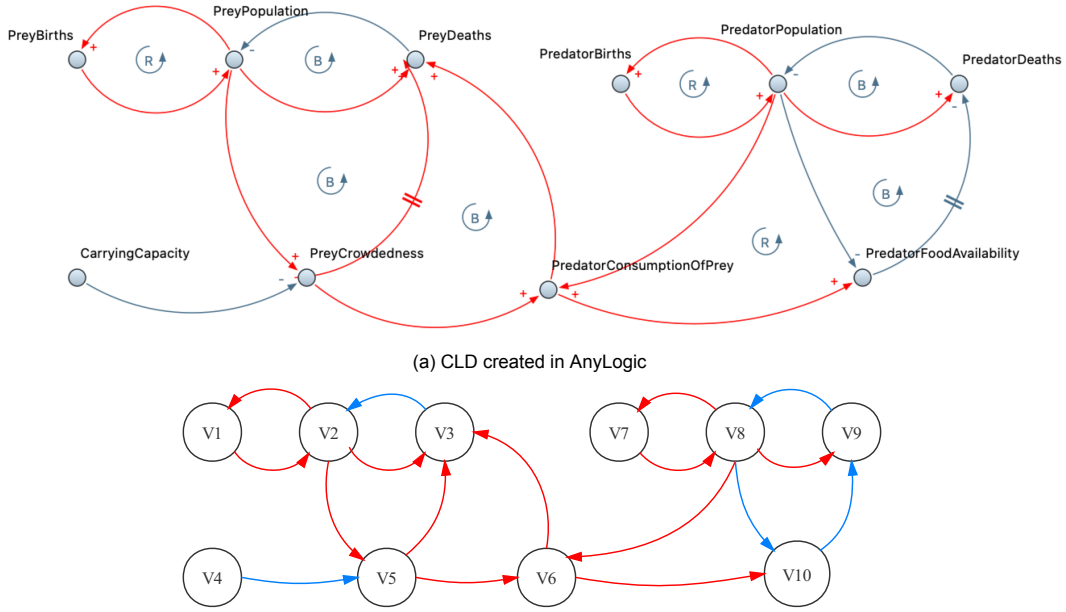
Abdelbari and Shafi [1, 2, 4] proposed to learn conceptual models with the ESN using system observations of output variables, which are stock variables in the SFD. We employ a different strategy to learn conceptual models since the prior knowledge about the system is not available in this case. Although [9] introduces an efficient way to transform a CLD to other SD models, it requires additional information about the system which is unreachable for us. The current CLD does not contain any feedback loops, which makes it even harder to identify the stock variables. Given that an ESN can potentially learn the information from the input and store it, we train the EvoESN on the data of all system variables and have it process all the information in its echo states. This setting aims at verifying the initial assumption which is we can learn conceptual models using EvoESN directly from system observations.

[1] https://www.anylogic.com



(a) CLD created in AnyLogic (b) Simplified CLD

Figure 5.1: CLD of Parents Education Impact system



(a) CLD created in AnyLogic

(b) Simplified CLD

Figure 5.2: CLD of Lotka-Volterra System

5.2.2. Lotka-Volterra System

The Lotka-Volterra (LV) system [22] is a well-developed ecological system describing how the population of the prey and the predator affect each other in an environment. Figure 5.2a shows the complete CLD created using AnyLogic[1]. The simplified CLD is shown in Figure 5.2b with the feedback loop sign removed. Similar to Figure 5.1b, links in red are positive causal links and those in blue are negative causal links. This model contains 10 varaibles including a constant varaible $V4$, 17 causal links and 8 feedback loops. The output variables, $V2$ and $V5$, are pre-defined. We only need the data of these two variables to train the EvoESN, and 100,000 sample data for each variable is generated from its simulation model.

5.3. Data for PEI

We study the PEI system in the municipality area of Amsterdam from 2004 to 2018. This section provides with details on how we collect and process the data.

5.3.1. Variables and Data Sources

From Statistics Netherlands (CBS)[2] the data for the *unemployment rate*[3] from 2003 to 2019 and the *average parents education level*[4] from 2011 to 2019 is available. The *average parents education level* is computed and measured by the education level (that is, the highest level of schooling attained) of parents who are living with their children. The education level is measured in three classes: low, secondary and high, and is represented by 1, 2 and 3 in the computation respectively. The reader can refer to the data source for more information on the education level classes.

The *poverty* is measured by the number of people under minimum wages of social benefits. The *stable households* variable is measured by the number of disorganized families, i.e. families where a single parent live with the children. We have found the data for them from 2004 to 2018 on *Wijk en Buurtkaart*[5] (in Dutch) from CBS.

We could not collect enough data for the rest of the variables within the limited time. A small part of the data for *domestic violence* is available from BBGA[6] dataset. It is not accepted in the experiment because of the extreme sparsity. Although the semantic meaning is clear, it is difficult to define the right indicators for variables *socially weaker* and *limited self-reliance and self-confidence*, leading to a lack of the data. The data sources for variables *psychological well-being* and *alcohol and drug use* remain unknown.

5.3.2. Data Interpolation

The collected data is not integral for the whole time, and we need to interpolate the data for missing years. Since the data shows a linear pattern over time, we employ the linear interpolation for simplicity. The results, as well as the original data, are shown inFigure 5.3. In each subfigure, the blue points indicate the originally collected data and the orange ones for interpolated data. There is no missing data for the *unemployment rate*.

5.3.3. Data Simulation

There remain variables without data available, and thus we need to generate the simulated data for them. In this process, the aim is to preserve the overall trend in the data, while the absolute values are irrelevant. The causal relations in the CLD indicate how variables change along with others and based on this foundation, the data is generated using the following ad hoc equations:

$$
\begin{aligned}
V_3 &= 100 - 0.5 * V_1 - 0.5 * V_2, \\
V_6 &= 100 - 0.5 * V_3 - 0.5 * V_5, \\
V_9 &= 1 + 0.5 * V_6 + 0.5 * V_8, \\
V_7 &= 1 + 0.5 * V_6, \\
V_4 &= 1 + 0.5 * V_3,
\end{aligned}
$$

where the data for V_1, V_2, V_5 and V_8 in Figure 5.1b are known. By applying the above linear equations, we assume linear causal relations between variables and maintain the trends over time in the generated data. An overview of the data for all variables is shown in Figure 6.2.

[2] https://opendata.cbs.nl/

[3] https://opendata.cbs.nl/statline/#/CBS/nl/dataset/84703NED/table?ts=1587027459596

[4] https://opendata.cbs.nl/statline/portal.html?_la=nl&_catalog=CBS&tableId=84337NED&_theme=348

[5] https://www.cbs.nl/nl-nl/reeksen/geografische-data

[6] https://data.amsterdam.nl/datasets/G5JpqNbhweXZSw/basisbestand-gebieden-amsterdam-bbga/

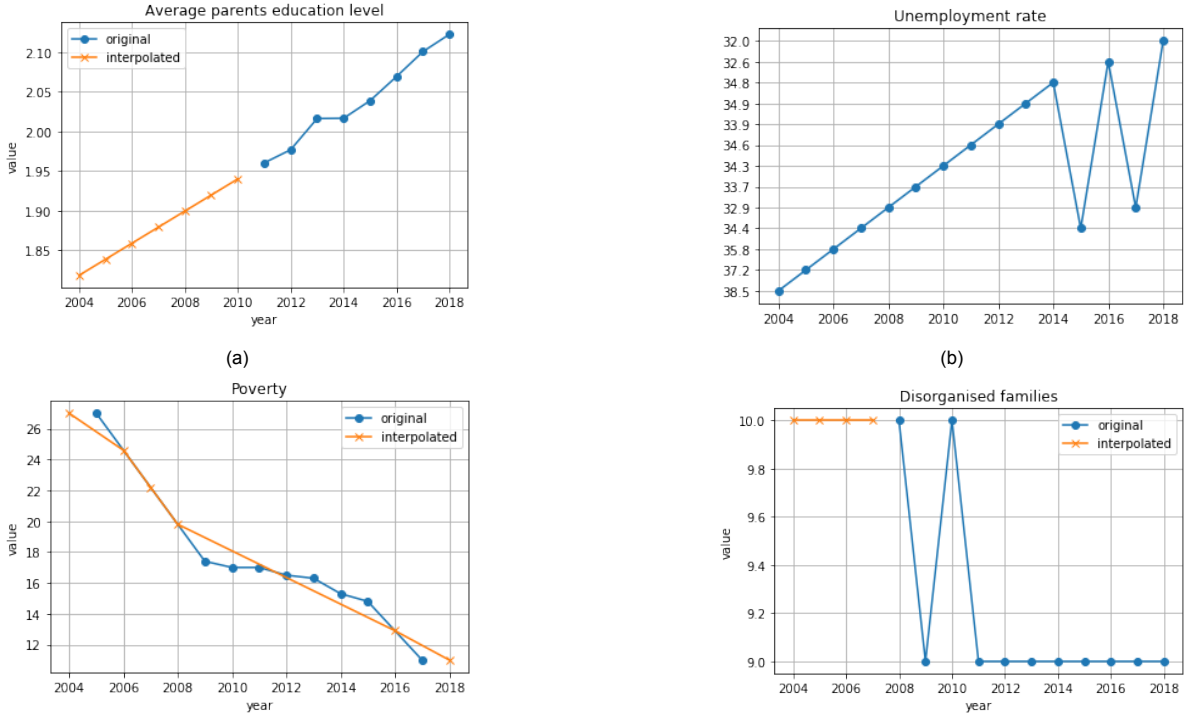


Figure 5.3: Available and interpolated data for PEI

6

Results

In this section, the results of our experiments are described and discussed. First verification and validation of the proposed methods are illustrated, followed by the results of EvoNN and EvoESN, respectively.

6.1. EvoNN

To validate the proposed method and evaluate the effectiveness of EvoNN on CLD annotation task, we observe the output error of the network. For verification purpose, we further show and discuss the simulation results. Finally, we visualise the EA convergence to evaluate its performance for each variable.

The case under study is the PEI system as it provides a rich set of causal relations and behaviour in the data. The PEI system does not contain constant variables. Besides, the data for all variables are available, which is another reason why we choose it over the LV system. The CLD annotation task focuses on the variables in the system instead of the whole system. We validate that the EvoNN works on one system, and we assume it works on all the systems WOLOG because it may differ from another one, but the causal relations are the same in structure.

6.1.1. Performance

We start with the EvoNN described in section 4.1, using the SNN with one hidden layer introduced in Figure 4.2.

Apart from the variable *average parents education level*, we annotate each variable in the CLD depicted in Figure 5.1. There are eight variables to study, and thus there are eight learned networks in total. In Figure 6.1 the error bar shows the average output error and the standard deviation of the error for each network. The vertical axis represents the error. The horizontal axis represents the name of the variables annotated. The output error is measured from leave-one-out cross-validation using the mean squared error (MSE), calculated as

$$\epsilon_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \tag{6.1}$$

where n is the total number of the sample of output data, vector Y represents the data points to be predicted, serving as the ground truth, and vector $\hat{Y}$ is the predicted data points.

The average error and standard deviation are relatively high for the *unemployment rate*, *self-reliance and self-confidence*, and *poverty*, indicating that the learned networks for them fail to represent the quantitative relations for them. The reason why the output error is high could be: the network structure is not complicated enough, the EA fails to find the optimal set of hyperparameters and the input data is of low quality and contains noise. We verify the performance of the learned networks by showing the simulation results generated by them.

We set the input data of *average parents education level* fixed and generate the data for the rest of the variables predicted by the learned networks. Figure 6.2 shows the comparison of the real data and the simulation data of each variable. In each figure, the horizontal axis is the year of the data, and the vertical axis shows the value of the data. For variables the *unemployment rate* in Figure 6.2b, *poverty* in Figure 6.2e and *disorganised families* in Figure 6.2h, the simulation data show an opposite trend of the real data, indicating the network for these variables fail to annotate the correct relations. The networks seem to produce close or similar simulation data to the real one for the rest variables.

The data for variables the *unemployment rate*, *poverty* and *disorganised families* are real-world data and might contain noise. It is the networks for these variables that produce high output error and unsatisfying results. Since neither the behaviour of the EA nor the quality of the input data can be guaranteed, it is reasonable to increase the complexity of the network to obtain better learning results. We then employ a SNN with two hidden layers and perform the same validation and verification experiments to test the assumption.

The structure of the SNN with two hidden layers is depicted in Figure 6.3. We apply the same methodologies as described in section 4.1 and run the same experiments as above.

We first perform the CLD annotation task with EvoNN using the two-hidden-layer SNN. The error bar is shown in Figure 6.4. Compared to Figure 6.1, both the average error and the standard deviation

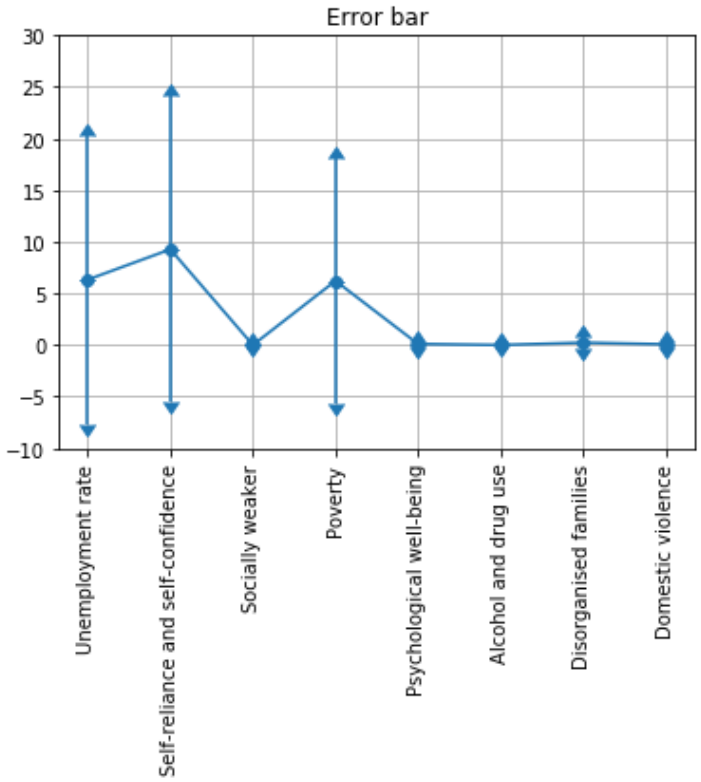


Figure 6.1: The average output error and standard deviation for each network using one-hidden-layer SNN

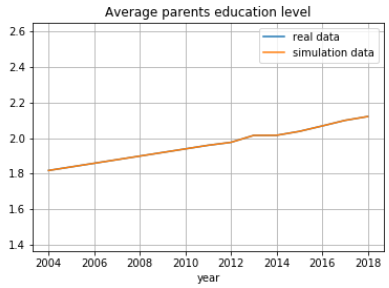
for variables the *unemployment rate*, *self-reliance and self-confidence*, and *poverty* have reduced and fallen into an acceptable range.

The simulation results predicted by the learned network is shown in Figure 6.5. Fixing the input data of the variable *average parents education level*, the EvoNN produces similar behaviour to the real data for all variables, confirming that the proposed methodology is feasible in solve the CLD annotation task and has been correctly implemented. The overall linear behaviour in the simulation data is the result of the linear one in the input data. Despite the fluctuation in the data (e.g. Figure 6.5b), the EvoNN is able to catch the overall trend and predict similar results.

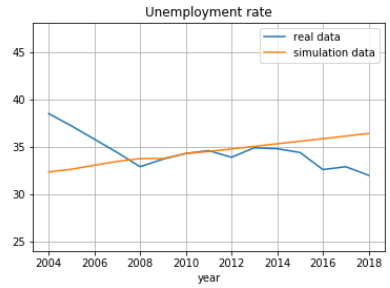
Compared to the one-hidden-layer SNN the two-hidden-layer one can deal with more complicated situations and generate better results in such cases. The structures of both SNNs are uncomplicated and comparable, despite which the two-hidden-layer SNN also shows the ability to resist the noise in the data, which is an essential characteristic in dealing with real-world data. The noise comes from various sources and is unavoidable in many cases. It can be intrinsic to the data. It may add noise to the data in collecting it by inappropriate means or calculations. The reference model is subject to some extent. When generating data using the reference model, it injects the noise as well. Based on the overall performance and the simulation results, the proposed methodology, the EvoNN, shows the reliability to handle the CLD annotation task and replace the traditional equation formulation with neural networks in the SD modelling.

6.1.2. EA Convergence

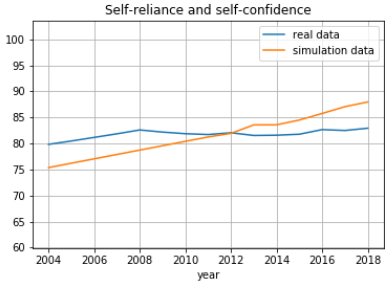
EA is applied to optimise the hyperparameters in the above SNN and is proven to be reliable in the optimisation. To further study the performance of how the EA works with the two-hidden-layer SNN, we visualising the convergence of the EA for each variables in Figure 6.6. It shows that the fluctuation in the data slows down the convergence of the EA, which corroborates the previous finding that it is more difficult for the SNN when the fluctuation exists in the data. Judging from the overall performance of all EAs, the fitness converges closely to 0 within a few generations nevertheless.



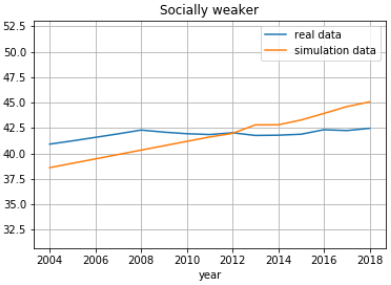
(a)



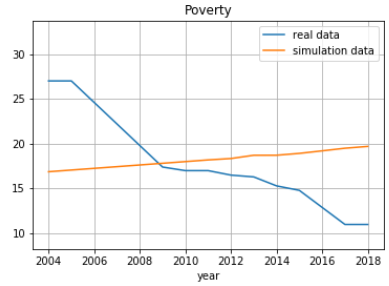
(b)



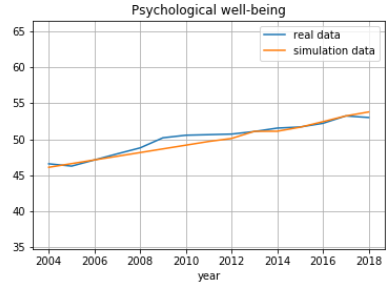
(c)



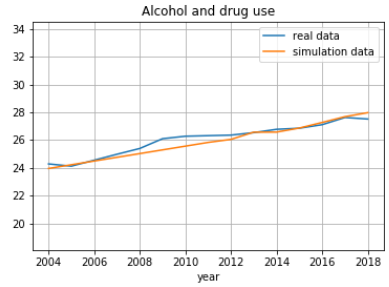
(d)



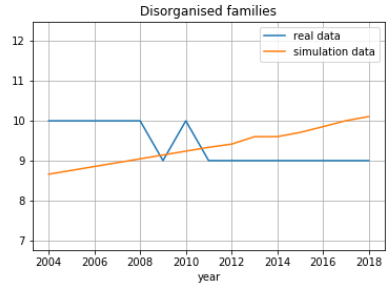
(e)



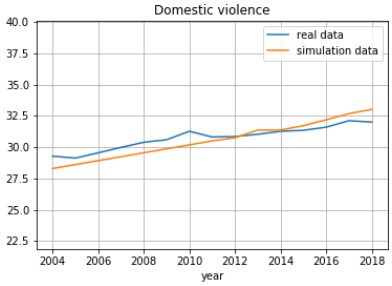
(f)



(g)



(h)



(i)

Figure 6.2: Comparison of real data with simulation data generated from one-hidden-layer SNN for PEI system

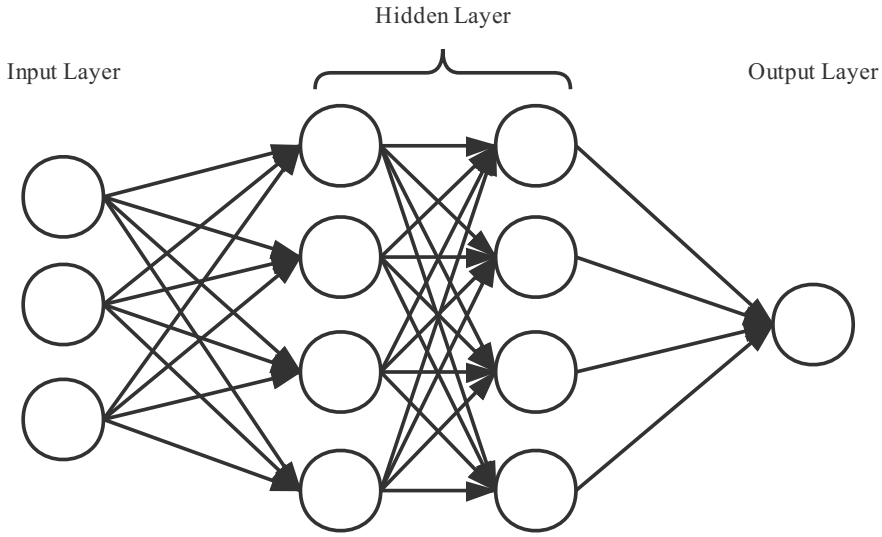


Figure 6.3: Structure of a shallow neural network with two hidden layers

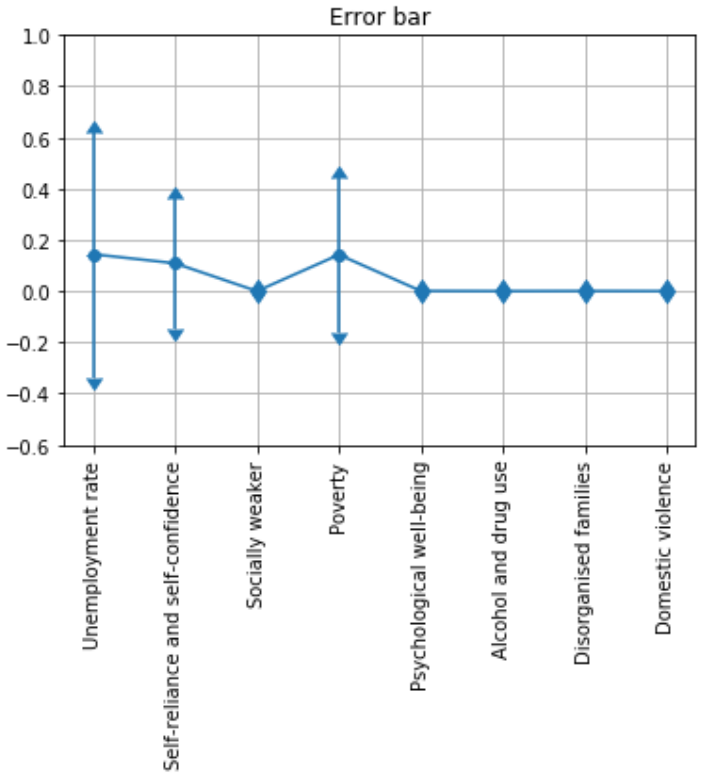
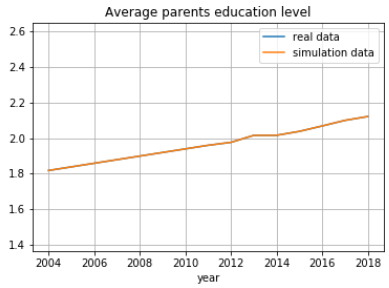
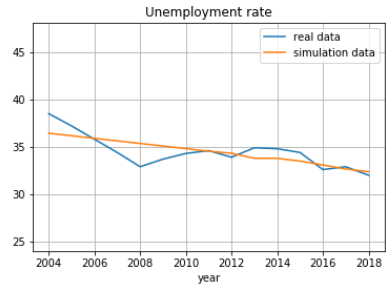


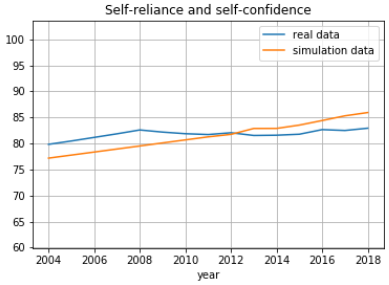
Figure 6.4: The average output error and standard deviation for each network using two-hidden-layer SNN



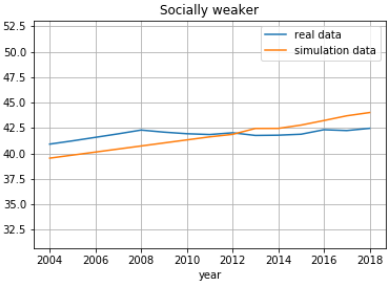
(a)



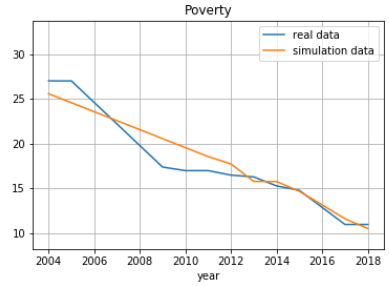
(b)



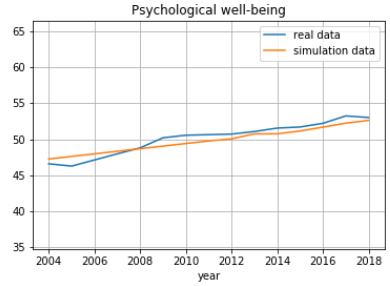
(c)



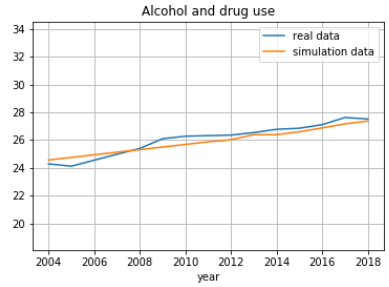
(d)



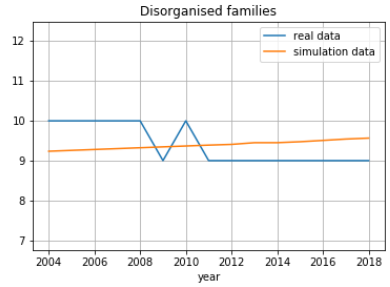
(e)



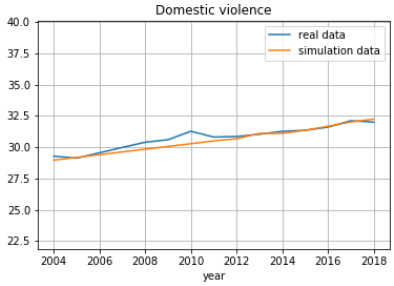
(f)



(g)

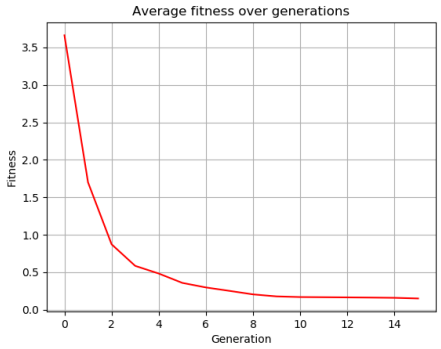


(h)

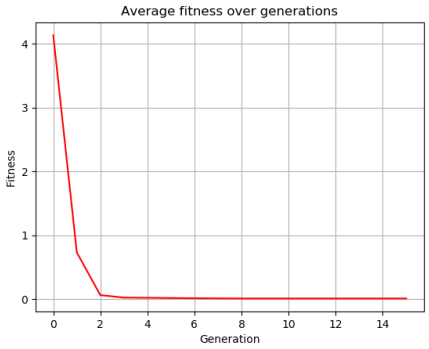


(i)

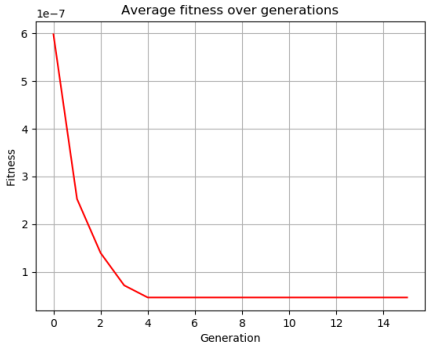
Figure 6.5: Comparison of real data with simulation data generated from two-hidden-layer SNN for PEI system



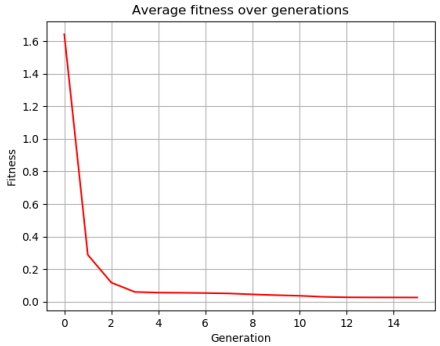
(a) Convergence of EA for *unemployment rate*



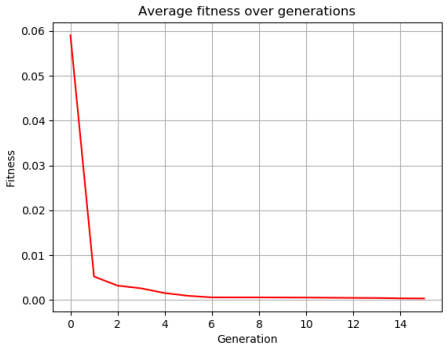
(b) Convergence of EA for *self-reliance and self-confidence*



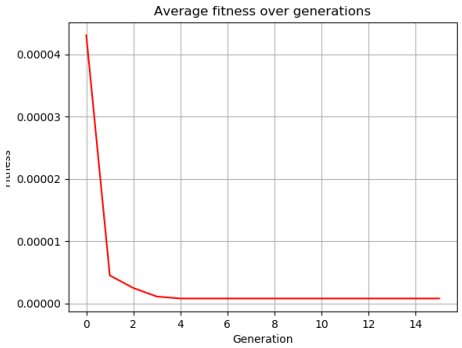
(c) Convergence of EA for *socially weaker*



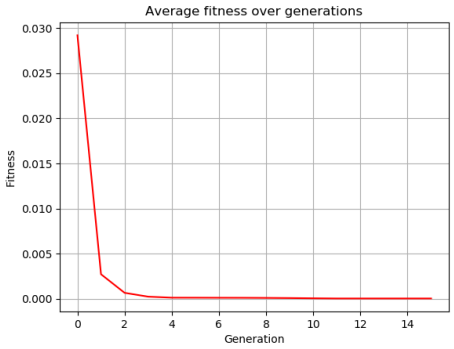
(d) Convergence of EA for *poverty*



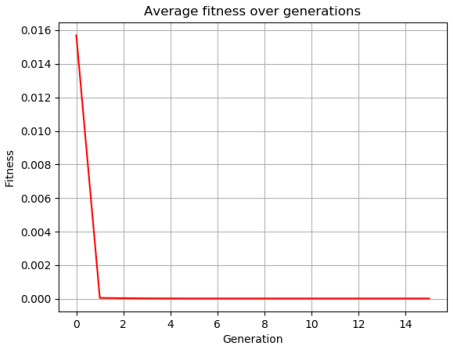
(e) Convergence of EA for *psychological well-being*



(f) Convergence of EA for *alcohol and drug use*



(g) Convergence of EA for *disorganised families*



(h) Convergence of EA for *domestic violence*

Figure 6.6: Convergence of EAs for each variabl in PEI system

Table 6.1: Mean output errors and structural errors for EvoESN using different fitness settings in EA (bold best for each system)

	PEI		LV	
Weight index	Output Error	Structural Error	Output Error	Structural Error
WC_1	0.1281981	0.1389	1.2170139	0.1333
WC_2	0.1281897	0.1458	1.2170265	0.1500
WC_3	0.1281902	0.1389	1.2170170	0.1333
WC_4	**0.1281848**	0.1250	1.2170173	0.1444
WC_5	0.1281933	0.1389	1.2169687	0.1333
WC_6	0.1281919	0.1458	1.2170170	0.1389
WC_7	0.1281920	0.1528	**1.2142639**	0.1500
WC_8	0.1281874	0.1250	1.2267988	0.1556
WC_9	0.1281919	0.1389	1.2170615	0.1500

6.2. EvoESN

We have shown the experiment results for EvoNN, and now we study EvoESN. This section provides experiment results obtained in both case studies (section 5.2) for different ESN settings discussed in section 5.1.

First, we perform a sensitivity analysis on the weight combinations in Equation 5.2. This experiment is to evaluate the influence of the fitness function on the output error of EvoESN. The best weight combination that leads to minimum output error is then determined and used for later experiments. The validation of EvoESN, as well as the evaluation of the result, is performed on both case studies. To verify the hypothesis that the optimised ESN can produce better results and consume fewer resources than the basic ESN, we compare the output errors and resulted models. In addition to the evaluation by the output error, the best-learned model using EvoESN is assessed by a domain expert.

We further verify EvoESN by showing its capability to learn the quantitative relations between system variables. Conducting the simulation experiment on both case studies, we show that the EvoESN is reliable to learn different system models and able to reproduce the system behaviours represented by key variables.

After the verification and validation, as well as the evaluation on the result, we finally run a scalability test using the LV system with different scales of input size.

6.2.1. Sensitivity Analysis

The results of the sensitivity analysis for various weight combinations of the fitness function in EvoESN is shown in Table 6.1. Overall the EvoESN produces higher output errors on the LV than the PEI. For both cases, the difference between output errors is too small to be distinguishable. WC_4 has generated the lowest output error for the PEI, while it is WC_7 for the LV. The best-learned models are different in structure, judging from the structural error. However, the weight setting in the EA fitness function does not significantly affect the output error of the EvoESN. This finding slightly differs from what was reported in [4], where the output errors are higher, and the weight setting has a significant influence on the output error. A possible reason is because of different EAs. Although the sensitivity analysis shows the correlation between output errors and weight settings is weak, we will adopt WC_4 on the PEI and WC_7 on the LV in the following experiments.

6.2.2. Performance

Figure 6.7 shows the best-learned models for the basic ESN and the EvoESN. Both models for PEI are sparse and distinguishable in the structure. In Figure 6.7c, $V1$ and $V8$ are dominating as they are not affected by any other variables. Figure 6.7e shows $V1$, as well as $V7$, are dominating variables, while $V1$ is the dominating variable in the original CLD. Both ESNs have identified $V1$ and added more information as well. From the error measurements shown in Table 6.2, the basic ESN produces an

Table 6.2: Error measurements for best-learned models using difference ESN variants for PEI and LV systems

	Output error	Structural error
Basic ESN on PEI	inf	0.1250
EvoESN on PEI	0.1282	0.1250
Basic ESN on LV	0.6287	0.4778
EvoESN on LV	1.2143	0.1500

Table 6.3: Link statistics for best-learned models compared to original CLDs (opposite polarity is correct causal link with opposite polarity)

	Correct	Opposite polarity	Missing	Additional
Basic ESN on PEI	0	3	10	6
EvoESN on PEI	2	0	11	7
Basic ESN on LV	7	8	2	68
EvoESN on LV	0	3	14	7

infinite output error, indicating it is beyond the capability to learn the system behaviour. Although the structural error is the same as that of EvoESN, the best-learned model by the basic ESN is not reliable. On the other hand, EvoESN produces a low output error.

Results for the LV is depicted on the right side of Figure 6.7. Compared to Figure 6.7f, the best-learned model from the basic ESN preserves a high degree of redundancy. The main reason is parameters in the basic ESN have not been fully optimised by the brute force search, and thus the reservoir requires more complexity to fit the target system behaviour, leaving redundancies in the network. Although sparse in structure, the EvoESN best-learned model produces a comparable output error and a lower structure error. The conclusion draw from the LV results remains the same as that from the PEI: EvoESN performs better in optimising parameters and producing the desired outcome.

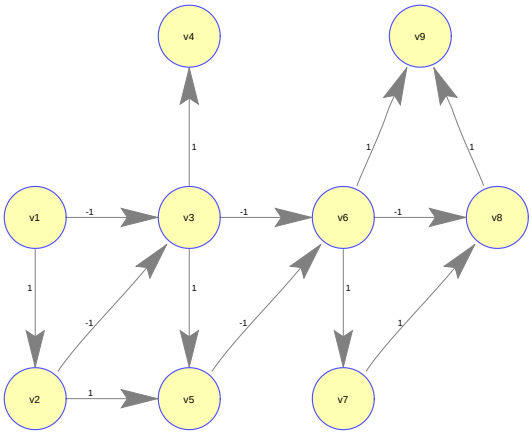
In addition to error measurements, we show the results of link statistics for each learned model in Table 6.3. This table only serves as a supportive tool and provides with a structural comparison between the original CLD and the learned model. There are 13 links in the PEI CLD and 17 ones in the LV CLD, and only a small number of links are learned correctly by the ESN. Since the ESN is learning towards the system behaviour instead of the system structure, it is acceptable for ESNs to miss the original structure and add new information. 68 additional links are learned by the basic ESN on LV, contributing largely to the high structural error of 0.4778.

ESN overall performs better on the PEI than the LV even though much more data is available in the latter case. A possible explanation is that the data of all variables in the PEI is used in training the ESN while only two variables are used in the LV case. During training, the ESN is able to extract more information of the system from more variables, and the implicit causal relations between variables can be identified as well.

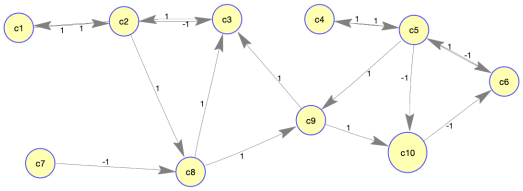
The runtime time results for each ESNs are shown in Table 6.4. Compared to the EvoESN, it takes longer time to optimise and train the basic ESN. Together with the previous error analysis, this result confirms our hypothesis that EvoESN is less time-consuming and produces better results than the basic ESN.

Table 6.4: Runtime in second for different ESNs

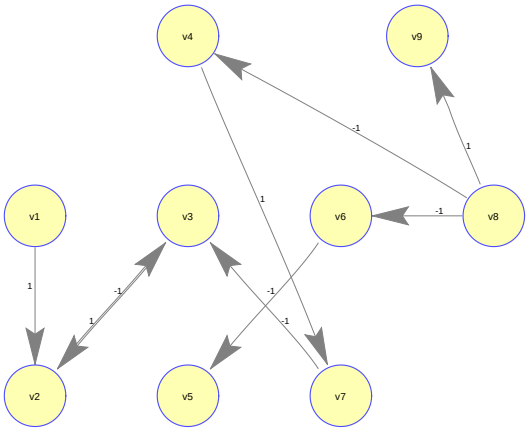
	Basic ESN	EvoESN
PEI	14	7
LV	1590	123



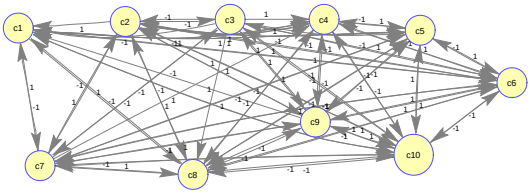
(a) Case PEI original CLD



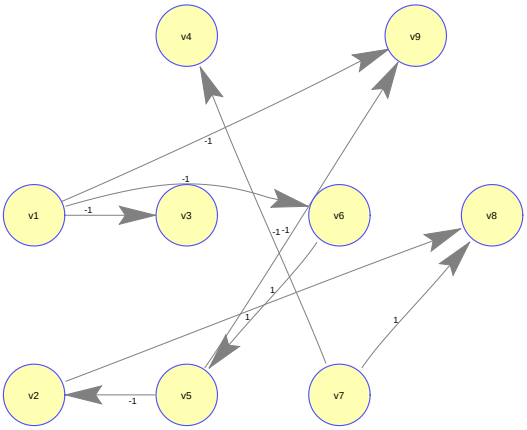
(b) Case LV original CLD



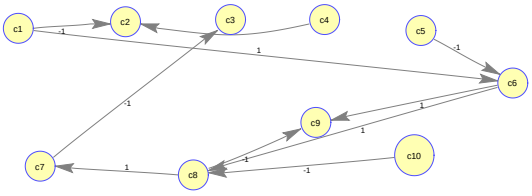
(c) Case PEI best-learned model using basic ESN



(d) Case LV best-learned model using basic ESN



(e) Case PEI best-learned model using EvoESN



(f) Case LV best-learned model using EvoESN

Figure 6.7: Best-learned models using different ESN variants for PEI and LV systems (original CLDs for better comparison)

Table 6.5: EvoESN runtime on different sizes of sample data. EA converge indicates on which generation it converges.

Data size	EA runtime (s)	EA converge (# generation)	ESN runtime (s)
10	0.4	12	0.1
100	1.0	6	0.2
1000	2.2	14	0.2
10000	14.6	13	0.6
100000	123.0	7	2.4
1000000	1029.6	9	19.8
10000000	11651.0	-	194.8

6.2.3. Judgement from the Domain Expert

We present the best-learned model using EvoESN to the domain expert for further evaluation. One can evaluate the model from different perspectives and decide whether it provides with new insights to the system. The assessment of the best-learned model using EvoESN (Figure 6.7e) on the PEI is, and we quote,

"In general, the learned structure makes sense. $V1$*, the average parents education level, and* $V7$*, the alcohol and drug use, becomes the dominating variable, which is reasonable, although* $V7$ *is not a dominating variable in the original CLD. Looking at individual variables,* $V1$ *no longer connects* $V2$*, the unemployment rate, which may be inappropriate."*

6.2.4. Simulation

This methodology, EvoESN, is proposed to tackle the automated model learning task. Training with system observations, it learns both the model structure and the quantitative relations between variables. So far, we have presented its capability to learn the model structure. We now illustrate the simulation performance. The simulation task is to verify EvoESN has learned the quantitative relations and can reproduce the same system behaviour by generating outputs from the best-learned model.

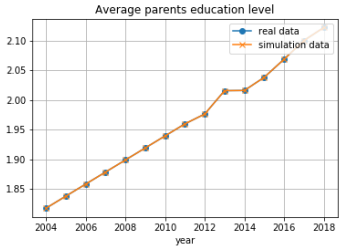
The outputs from the best-learned model using EvoESN for the PEI system is depicted in Figure 6.8. The results for the LV system is depicted in Figure 6.9. From both figures, we observe that the simulation data overlap with the real data in both cases. This result agrees with that in [4], confirming that the best-learned model using EvoESN can preserve the underlying relations between variables. Together with the experimental results of EvoESN, we show that EvoESN learns both the model structure and the quantitative relations from system observations.

6.2.5. Runtime Analysis

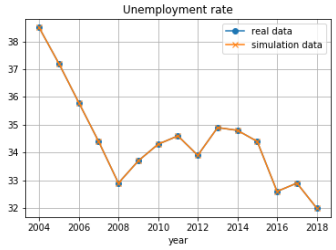
So far, we have shown the performance, as well as the evaluation on the results, of EvoESN. It would be interesting to inspect how the algorithm works on input data of massive volume. We now run the EvoESN on different size of input data to perform the runtime analysis. The task is to investigate the scalability of the algorithm on different input sizes.

Table 6.5 shows the runtime results of EvoESNs on different size of sample data for LV system. An enormous amount of data is available in this case, and thus we opt for it to perform the scalability test. The relation between the input size and the EA convergence is unclear. It converges within 15 generations in general, but for extremely huge input, the EA fails to converge within this range. One of the reasons why the converging speed being unstable could be the randomness in the initial population and the mutation in the offspring. The runtime of EA and ESN scales almost linearly as the input size grows, which is expected. The computation in the EA and ESN training is mainly the matrix multiplication, resulting in the linear runtime complexity of EvoESN.

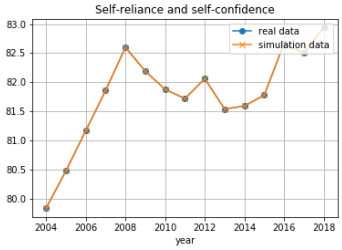
In practice, we do not expect to study an urban system for an extremely long period, meaning that the data size tends to be small in term of time. Therefore, the proposed EvoESN is considered to be a fast solution to automated model learning.



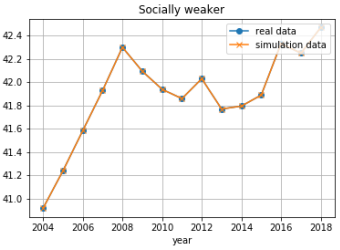
(a)



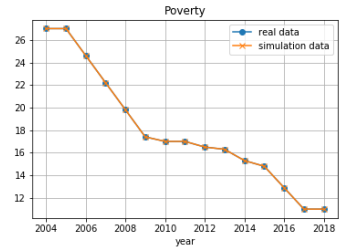
(b)



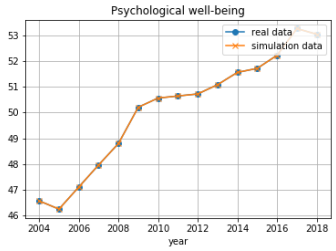
(c)



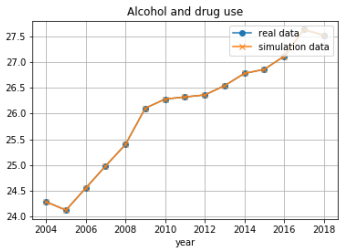
(d)



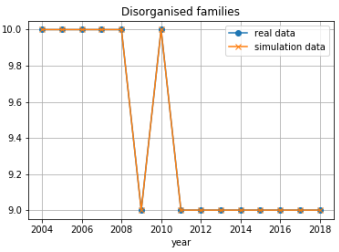
(e)



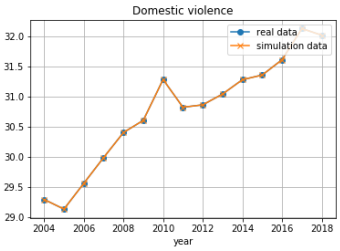
(f)



(g)

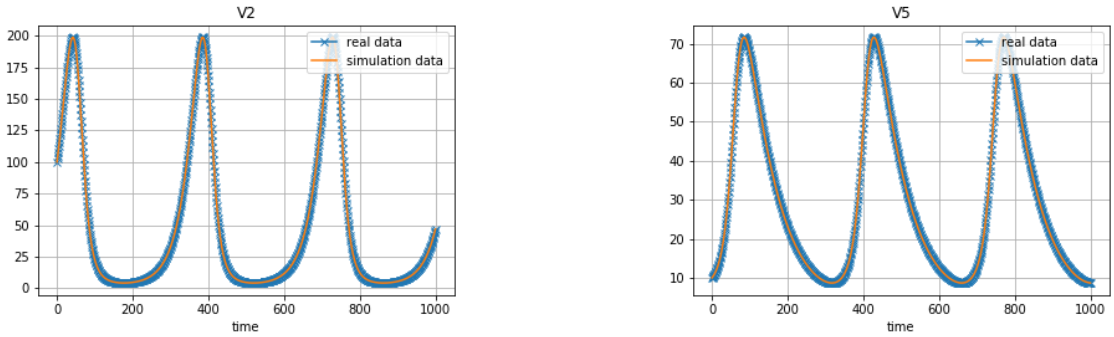


(h)



(i)

Figure 6.8: Comparison of real data with simulation data generated from best-learn model using EvnESN on PEI system



(a) (b)

Figure 6.9: Comparison of real data with simulation data generated from best-learn model using EvnESN on LV system (showing the first 1,000 data points for better visualisation)

7

Conclusion

Traditional SD modelling involves massive human effort and requires a comprehensive understanding of the system under study. This work has proposed automated model learning methodologies under different levels of prior knowledge about the urban system. Based on a PEI model received from the domain expert, we perform a case study on the city of Amsterdam and provide answers to the research questions by solving two tasks: annotating the CLD and simulate the model using EvoNN, and learning both structures and quantitative relations using EvoESN.

We tailor an optimised SNN to learn the quantitative relations between system variables. By adding another hidden layer, the SNN shows the ability to learn more complex situations, and better fit the system behaviour. Combined with a $(\mu + \lambda)$ EA for the hyperparameter optimisation, the SNN can further evolve for different learning targets. The evolutionary SNN, also known as the EvoNN, has proved the feasibility to learn the quantitative relations given the prior knowledge about the system structure.

The second proposed methodology, the EvoESN, is developed based on the groundwork of Abdelbari and Shafi [1, 2, 4]. Adapting their work, we can learn the conceptual model and yielding similar resulted model for both a real-world system and a well-developed model from the literature. A self-evolved customised ESN is devised for the purpose. We manipulate the initialisation of the reservoir weight matrix to prevent infeasible solutions as much as possible. The $(\mu + \lambda)$ EA is employed to optimise the hyperparameters in the ESN, and it adopts the fitness function used in [4] which takes into account both the output error and the connectivity probability. Varying the weight combination in the fitness was reported to affect the result significantly. However, we have found that it only leads to a slight difference in the results. Instead of learning towards the output variables this work has the EvoESN learn the system observations of all variables in case of unavailable output variables. Doing so allows the ESN to maintain much more information in the reservoir. Experiment results have shown the feasibility for the EvoESN to learn both the structure and quantitative relations of conceptual models from the observations.

Overall we can answer the main research question and conclude that it is feasible to enhance both qualitative and quantitative modelling of an urban system using ML techniques. We provide an answer to the RQ1 by solving the CLD annotation task and simulate the model using EvoNN on a real-world urban system in Amsterdam. The second proposed methodology, EvoESN, answers the RQ2 and solves the automated model learning problem on the same system. Thus, based on this work, SD modellers can benefit from automated dynamic system modelling. Considering that a real-world urban system may be complicated in structure but lacking in data, the proposed methodologies, served as supportive tools, can produce satisfactory results within a limited time, reducing much less modelling time and human effort in the process. Furthermore, the proposed methodologies remain robust even when lacking system observations.

In general, there are still limitations in the methodologies and more work can be done to improve them. The PEI system is the only real-world case in our experiments, and we could not collect all data. One can continue collecting the data to replace the simulated one and study on more complex urban systems. Furthermore, the best-learned model using EvoESN is evaluated by one domain expert. It would be interesting to incorporate with several domain experts in the assessment.

Multiple ways might change and improve the results for EvoNN and EvoESN. One could, for example, use different EA variations to optimise hyperparameters. The fitness function is one of the essential operators in the EA and often problem-dependent. Abdelbari and Shafi [4] use an additional penalty term in the fitness function to prevent infeasible solutions. One could research on designing a better fitness function to potentially achieve a closer solution to the global optimal. Besides, a complicated urban system may require constraints on variables and simulation results. In such cases, the fitness function must be formulated carefully to achieve fast EA convergence on the global optimal.

We have shown in this work the possibility to apply ML on automated SD model generation. It would be interesting to explore more topics in the SD domain. For example, Chen and Jeng [10, 12] apply ML and EA in policy design for SD models. Yücel and Barlas [33, 34] work on pattern-based parameter search.

Bibliography

[1] Hassan Abdelbari and Kamran Shafi. Optimising a constrained echo state network using evolutionary algorithms for learning mental models of complex dynamical systems. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 4735–4742. IEEE, 2016.

[2] Hassan Abdelbari and Kamran Shafi. A computational intelligence-based method to 'learn' causal loop diagram-like structures from observed data. *System Dynamics Review*, 33(1):3–33, 2017. doi: 10.1002/sdr.1567. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sdr.1567.

[3] Hassan Abdelbari and Kamran Shafi. A genetic programming ensemble method for learning dynamical system models. In *Proceedings of the 8th International Conference on Computer Modeling and Simulation*, ICCMS '17, page 47–51, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348164. doi: 10.1145/3036331.3036336. URL https://doi.org/10.1145/3036331.3036336.

[4] Hassan Abdelbari and Kamran Shafi. Learning structures of conceptual models from observed dynamics using evolutionary echo state networks. *Journal of Artificial Intelligence and Soft Computing Research*, 8(2):133 – 154, 2018. URL https://content.sciendo.com/view/journals/jaiscr/8/2/article-p133.xml.

[5] Hassan Abdelbari, Sondoss Elsawah, and Kamran Shafi. Model learning using genetic programming under full and partial system information conditions.

[6] Zaid Alrashdi and Mohammad Sayyafzadeh. (μ+ λ) evolution strategy algorithm in well placement, trajectory, control and joint optimisation. *Journal of Petroleum Science and Engineering*, 177: 1042–1058, 2019.

[7] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

[8] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[9] Thomas Binder, Andreas Vox, Salim Belyazid, Hordur Haraldsson, and Mats Svensson. Developing system dynamics models from causal loop diagrams. In *Proceedings of the 22nd International Conference of the System Dynamic Society*, pages 1–21. Citeseer, 2004.

[10] Yao-Tsung Chen. *The Use of SDM□RNN Transformation for System Dynamics Model Construction and Policies Design*. PhD thesis, National Sun Yat□sen University, 2001.

[11] Yao-Tsung Chen and Bingchiang Jeng. Yet another representation for system dynamics models, and its advantages. In *Proceeding of the 20th International Conference of the System Dynamics Society*. Citeseer, 2002.

[12] Yao-Tsung Chen and Bingchiang Jeng. Policy design to fitting desired behaviour pattern for system dynamics models. 01 2004.

[13] Yao-Tsung Chen, Yi-Ming Tu, and Bingchiang Jeng. A machine learning approach to policy optimization in system dynamics models. *Systems Research and Behavioral Science*, 28(4):369–390, 2011.

[14] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: A python framework for evolutionary algorithms. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 85–92, 2012.

[15] Marc Drobek, Wasif Gilani, and Danielle Soban. A data driven and tool supported cld creation approach. In *The 32nd International Conference of the System Dynamics Society, Delft*, pages 1–20, 2014.

[16] Marc Drobek, Wasif Gilani, Thomas Molka, and Danielle Soban. Automated equation formulation for causal loop diagrams. In *International Conference on Business Information Systems*, pages 38–49. Springer, 2015.

[17] Aida A Ferreira and Teresa B Ludermir. Comparing evolutionary methods for reservoir computing pre-training. In *The 2011 International Joint Conference on Neural Networks*, pages 283–290. IEEE, 2011.

[18] Aida A Ferreira, Teresa B Ludermir, and Ronaldo RB De Aquino. An approach to reservoir computing design and training. *Expert systems with applications*, 40(10):4172–4182, 2013.

[19] Jay W Forrester. *Industrial Dynamics*. M.I.T. Press: Cambridge, MA., 1961.

[20] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, 13(1):2171–2175, 2012.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[22] Stefan N Groesser and Martin Schaffernicht. Mental models of dynamic systems: taking stock and looking ahead. *System dynamics review*, 28(1):46–68, 2012.

[23] Herbert Jaeger. The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148, 01 2001.

[24] Bingchiang Jeng, Jian xun Chen, and Ting peng Liang. Applying data mining to learn system dynamics in a biological model. *Expert Systems with Applications*, 30(1):50 – 58, 2006. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2005.09.068. URL http://www.sciencedirect.com/science/article/pii/S0957417405002393. Intelligent Bioinformatics Systems.

[25] Danil Koryakin, Johannes Lohmann, and Martin V. Butz. Balanced echo state networks. *Neural Networks*, 36:35 – 45, 2012. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2012.08.008. URL http://www.sciencedirect.com/science/article/pii/S0893608012002213.

[26] Kim Langfield-Smith and Andrew Wirth. Measuring differences between cognitive maps. *Journal of the Operational Research Society*, 43(12):1135–1150, 1992.

[27] Da Liu, Jilong Wang, and Hui Wang. Short-term wind speed forecasting based on spectral clustering and optimised echo state networks. *Renewable Energy*, 78:599–608, 2015.

[28] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14 (11):2531–2560, 2002.

[29] Mark Paich and John D Sterman. Boom, bust, and failures to learn in experimental markets. *Management Science*, 39(12):1439–1458, 1993.

[30] Erik Pruyt. *Small System Dynamics Models for Big Issues: Triple Jump towards Real-World Dynamic Complexity*. 01 2013. ISBN 9789461861955.

[31] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[32] John Sterman. Business dynamics, system thinking and modeling for a complex world. *http://lst-iiep.iiep-unesco.org/cgi-bin/wwwi32.exe/[in=epidoc1.in]/?t2000=013598/(100)*, 19, 01 2000.

[33] Gönenç Yücel and Yaman Barlas. Pattern-based system design/optimization. In *Proceedings of the 25th international conference of the system dynamics society, Boston*, 2007.

[34] Gönenç Yücel and Yaman Barlas. Automated parameter specification in dynamic feedback models based on behavior pattern features. *System Dynamics Review*, 27(2):195–215, 2011.

[35] Wang Zhao. Automated model conceptualization and interactive modeling environment: A software prototype. Master's thesis, The University of Bergen, 2019.