RODRIGO FRANÇA DAGUANO

Automatic calibration of traffic microsimulations with artificial neural networks

São Paulo

2019

RODRIGO FRANÇA DAGUANO

Automatic calibration of traffic microsimulations with artificial neural networks

Dissertation presented at the Polytechnic School, Universidade de São Paulo to obtain the degree of Master of Science

Supervisor: Prof. Dr. Leopoldo Rideki Yoshioka

São Paulo

2019

RODRIGO FRANÇA DAGUANO

Automatic calibration of traffic microsimulations with artificial neural networks

Corrected Version

Dissertation presented at the Polytechnic School, Universidade de São Paulo to obtain the degree of Master of Science

Area of concentration: Electronic Systems Engineering

Supervisor: Prof. Dr. Leopoldo Rideki Yoshioka

São Paulo
2019

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Daguano, Rodrigo França
Automatic calibration of traffic microsimulations with artificial neural networks / R. F. Daguano -- versão corr. -- São Paulo, 2019.
86 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Redes neurais 2.Simulação de sistemas 3.Sistemas inteligentes de transporte I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

Name: DAGUANO, Rodrigo França

Title: Automatic calibration of traffic microsimulations with artificial neural networks

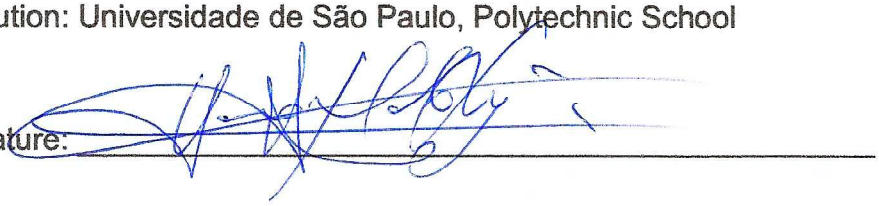
Dissertation presented at the Polytechnic School, Universidade de São Paulo to obtain the degree of Master of Science

Approved on: September 12th, 2019

Examination committee

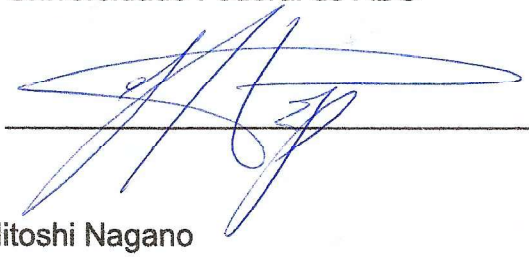
Prof. Dr. Leopoldo Rideki Yoshioka

Institution: Universidade de São Paulo, Polytechnic School

Signature: 

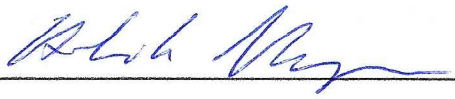
Prof. Dr. Humberto de Paiva Junior

Institution: Universidade Federal do ABC

Signature: 

Prof. Dr. Hitoshi Nagano

Institution: Fundação Getúlio Vargas, São Paulo School of Business Administration

Signature: 

In loving memory of my grandparents Nazaré, Libertina, Manoel and Benedito.

ACKNOWLEDGEMENTS

I would like to thank my parents and my sister, Sidney, Clemir and Juliana, for their unconditional support during this degree program.

I would also like to thank my post-graduation colleagues Leonardo Gomes and Lucas Koleff for their valuable friendship, discussions, jokes and laughs during the life at the university.

I would like to thank my supervisor Dr. Leopoldo Yoshioka for his guidance in the academic and professional fields during this research. Professors Dr. Marcio Lobo and Dr. João Justo were also part of project Mobility, another research project that was conducted in the same period of this degree, and they provided valuable help with the academic matters of the Polytechnic School.

Finally, I would like to thank Dr. Cláudio Marte and PhD student Olímpio Barros, from the Traffic Engineering department of the Polytechnic School, for their important collaboration in this research.

RESUMO

No campo de Transporte Inteligente, o planejamento urbano se beneficia de ferramentas computacionais de simulação de tráfego. Nas microssimulações, um passo importante é a calibração, que envolve o ajuste de parâmetros de entrada da simulação, de forma que as métricas de saída correspondam àquilo observado no mundo real. Trata-se de um processo iterativo e demorado, sendo tradicionalmente feito manualmente por um engenheiro de tráfego. Esta pesquisa propõe uma metodologia para calibrar automaticamente as simulações de tráfego. Inicialmente, executa-se um grande número de simulações e constrói-se um extenso banco de dados de casos. Depois, utiliza-se esse banco de dados para treinar redes neurais capazes de estimar as entradas do simulador que levam a determinados resultados, ou seja, dos cenários de estudo desejados. Este processo corresponde efetivamente à calibração da microssimulação de uma rede de transportes operando sob determinada condição. Experimentos de validação calibraram as configurações de roteamento e volume de tráfego nas simulações, sendo que nestes experimentos foi verificada uma alta correlação, acima de 80%, entre as estimativas das redes neurais e os valores desejados para as variáveis de entrada do simulador. Os experimentos demonstraram a possibilidade de automatizar e tornar escalável o processo de calibração. A avaliação das redes neurais forneceu as métricas individuais de cada variável de entrada do simulador, como volumes de veículos e decisões de rota, permitindo ao usuário escolher ou ignorar as estimativas com mau desempenho e alternativamente calibrá-las manualmente. Finalmente, dois experimentos investigaram a calibração de parâmetros comportamentais dos motoristas, um tipo de variável mais abstrata; Para esse tipo de variável, foram observados resultados com acurácia aceitável para cerca de metade dos parâmetros estimados. Neste caso, caberá ao usuário a escolha de ignorar as variáveis de pior desempenho e usar somente aquelas de desempenho satisfatório como um ponto de partida para refinamentos manuais. A metodologia proposta mostrou ser capaz de estimar com acurácia suficiente uma parte significativa dos parâmetros de calibração, o que reduzirá o esforço do engenheiro de tráfego na etapa de calibração e permitirá que se dedique ao seu trabalho de análise de cenários.

Palavras-Chave: Rede Neural Artificial. Simulação de tráfego. Vissim. Automatização de calibração. Modelo de regressão. Transporte Inteligente.

ABSTRACT

In the topic of Smart Transportation, urban planning took great advantage from computational simulation tools for traffic. In microsimulations, one important step is calibration, which is accomplished by tuning the values of simulation inputs, in order to match its internal metrics with those from real-world data. The process is iterative, time-consuming and is traditionally done manually by a traffic engineer. This research proposes a methodology to automatically calibrate traffic simulations. Initially a large number of simulations are run to create an extensive dataset of examples. Then, the dataset is used for training Artificial Neural Networks that are capable of estimating the simulation inputs that deliver the target output metrics, thus calibrating the simulations upon request of specific scenarios. Validation experiments were conducted to calibrate the routing and flow setup of simulations, and in these experiments it has been verified a high correlation, above 80%, between the estimates from the Neural Networks and the desired values for the input variables of the simulator, therefore validating the proposed methodology and the capabilities of automation and scalability of the calibration process. The evaluation of the Neural Networks also delivers the metrics for each individual input variable, such as vehicle volumes and route decisions, thus allowing the user to choose or ignore the estimates for those variables with poor performance and instead proceed to manual calibration. Finally, two experiments investigated the calibration capabilities of driving behavior parameters, a more abstract type of variable. For this type, results were observed with acceptable accuracy for about half of the parameters. In this case, it is the user's choice to ignore the variables with the worst performances and use those with acceptable performances as a starting point for refinement. The proposed methodology has been shown to be capable of estimating with sufficient accuracy a significant part of the calibration parameters, thus reducing the workload of a traffic engineer and allowing more dedication to the work of scenario analysis.

Keywords: Artificial Neural Network. Traffic simulation. Vissim. Calibration automation. Regression model. Smart Transportation.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1. Introduction

1.1. Context

Smart Cities have become a topic of interest as an approach to tackle urban problems with new technologies (PECAR; PAPA, 2017). Furthermore, there is special focus on the advancements of Information and Communication Technologies (ICT) and one possible division of the Smart City concept from the literature is into the branches of Smart Health, Smart Energy, Smart Governance and Smart Transportation, among others (MOHANTY; CHOPPALI; KOUGIANOS, 2016).

In Smart Transportation, one of the main goals is to improve urban mobility by applying ICT to the transportation infrastructure, multimodal transport integration schemes, vehicles, and to the procedures through which traffic authorities perform their planning and operational tasks. In the sense of enabling authorities to perform these tasks more efficiently, Smart Transportation implements Intelligent Transport Systems (ITS), an overlapping concept that also aligns with making updated traffic information available to the users of the road network. Smart Transportation uses ITS and considers roads critical, because they are generally regarded as inefficient in comparison to their potential of use (XIONG et al., 2012).

The tasks of planning new transportation infrastructure and estimating the impact of modifications to current road networks have specially benefited from simulation software (CHU et al., 2003; FANG; ELEFTERIADOU, 2005; HOLLANDER; LIU, 2008), which can be used for the comparison between two simulations in a project proposal. In this comparison, one simulation displays the current traffic conditions in the region of interest, and another simulation highlights the benefits of the new proposed infrastructure (future scenario) by improving the desired performance metrics of that traffic network, as shown in Figure 1.

Furthermore, traffic simulations are usually divided into macroscopic, microscopic and mesoscopic simulations. Macroscopic simulations assess the aggregate characteristics from vehicle volumes on road networks over large geographical areas. Microscopic simulations focus on the interaction between individual vehicles and the physical infrastructure, within areas that are smaller and

with high level of detail. Mesoscopic simulations fit as an intermediate case that analyzes small groups of vehicles, but under the assumption that their elements are homogeneous (VILARINHO, 2008; YIN; QIU, 2013). This research focuses on microscopic simulations.

For traffic simulations an important step is calibration, which is the process of matching the metrics from the modeled road network (e.g. average speeds, car queue lengths and average travel times) to the metrics observed in the real world, as shown in Figure 2. In the specific case of microsimulations (another name for microscopic simulations), which cover limited areas, this is accomplished by tuning the simulation inputs (e.g. number and types of vehicles that enter the simulation area and their characteristics) until the metrics match with small error in comparison to those observed in the real world. Then, the simulation is considered accurate and calibrated (HENCLEWOOD et al., 2017; RRECAJ; BOMBOL, 2015; TETTAMANTI et al., 2015).

The area of the map is limited and is it is built as a network of nodes and link segments. The configured vehicle volumes are inserted in the edges of the map, whereas the metrics are evaluated both edge-to-edge and internally in the map. In the second case, the metrics are the results from the manifestation of the vehicle behaviors and interactions, outside of the direct control from the user, who can only configure fixed behaviors to the vehicles that enter the edges of the map.

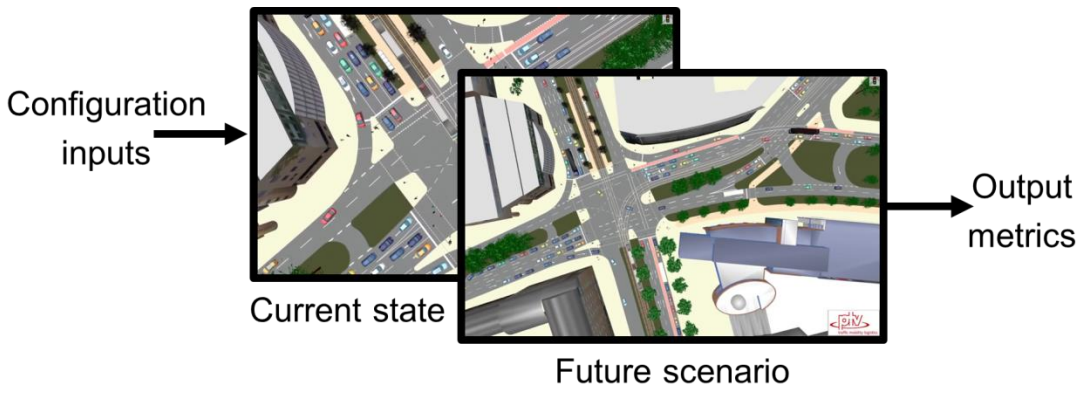


Figure 1 –Comparison between traffic simulations in current and proposed scenarios for a road network.

Source: Author.

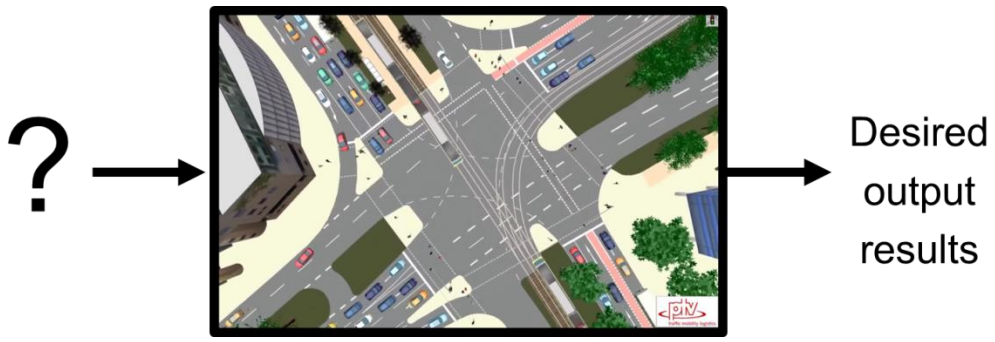


Figure 2 – Calibration as the search for the inputs that deliver desired output results.
Source: Author

1.2. Motivation

Traditionally, calibration has been done manually and iteratively by a traffic engineer. Also, smaller road segments that have less influence on the network dynamics are usually omitted for simplicity. Even so, the process is often very time-consuming and repetitive, as criticized by Chu et al. (2003) and Hollander and Liu (2008).

By contrast, the availability of computational power is an incentive to develop tools that automate the repetitive, yet analytical task of calibration (AGHABAYK et al., 2013; HENCLEWOOD et al., 2017; MA et al., 2015; PARK; QI, 2005; SHAFIEI; GU; SABERI, 2018). For some types of problems, one possible strategy is to run a large number of diversified and low-cost simulations at first, and then filter out a collection of the most successfully calibrated simulations (by chance).

Rrecaj and Bombol (2015) identified that for VISSIM, a commercially available traffic simulator, there is a convergence point related to optimization techniques being used for automatic calibration, with Genetic Algorithms (GA) being the most commonly used. Bethonico (2016), Tettamanti et al. (2015) and Yu and Fan (2017) also used GA to solve calibration as an optimization problem. GAs are heuristic techniques and therefore deliver solutions that are expected to perform better as the iterations run, but are not guaranteed to be optimal. Furthermore, even if the same road network is being calibrated, but in a different traffic scenario, the optimization

procedure needs to start from the beginning to deliver a new solution. The exception would be to use the previous results as a starting point.

On the other hand, Machine Learning techniques use large amounts of collected data to create models to classify or estimate variables. They have been used for calibration of traffic simulations and can alternatively use the results from the large number of simulations that are conducted in the heuristic optimization algorithms (SHAFIEI; GU; SABERI, 2018; OTKOVIC; TOLLAZZI; SRAML, 2013). Even if a large number of simulations need to be set and run to allow a more efficient learning process for the Machine Learning tools, one hypothesis is that those would not be more computationally expensive than the large number of simulations otherwise necessary for the optimization methods. Another advantage that this research seeks is the possibility of data reuse in the training of Artificial Neural Networks (ANN), specific computing systems that are used for supervised machine learning.

1.3. Objectives

- **General objective**: This research proposes a methodology to train Artificial Neural Networks that can automatically calibrate traffic microsimulations.
- **Specific objectives**:
  - Automate computing tasks of the traffic simulator to deliver a large number of simulation results.
  - Identify the most commonly used calibration methods and the research gap.
  - Develop Artificial Neural Networks that reuse information from the history of simulations for the calibration of the simulator and validate the proposed methodology.

1.4. Dissertation structure

The main text of this dissertation is organized as follows. Chapter 1 is an introduction that gives an overview of the research context, motivation, objectives and organization of this document.

Chapter 2 gives an overview of the key concepts from traffic simulators that are necessary to proceed with the research, as it is interdisciplinary and about Electrical Engineering and Computer Science methods being proposed for Traffic Engineering applications.

Chapter 3 describes and discusses the literature review related to calibration of traffic simulation and machine learning techniques applied to traffic problems.

Chapter 4 formally states the problem, the research question and proposes a methodology as a solution.

Chapter 5 describes the methods and setups to validate the proposed solution.

Chapter 6 presents experimental results and discussions.

Finally, chapter 7 describes the conclusions of this research and suggestions for further development.

2. Key concepts of traffic microsimulation

2.1. Traffic simulator

The proposed methodology performs the automatic calibration of traffic microsimulations. Microsimulation is one type in the broader scope of traffic simulations that includes macro, meso and micro. The main characteristic of microsimulations is that the elements (e.g. cars, buses and pedestrians) are modeled individually and that their interactions with one another and with the physical infrastructure are calculated at each simulation step (LLANQUE AYALA, 2013; BETHONICO, 2016).

A traffic simulator is a computer program that receives as inputs a collection of simulation settings composed by four elements: 1) The simulation configuration, 2) the virtual infrastructure (road network and attributes), 3) the description of the vehicles and the way in which they should navigate in the network and 4) the parameters that adjust the behavioral models used in the vehicles, pedestrians and other traffic elements.

The simulator delivers as outputs all the calculated properties for all vehicles in the simulation, such as positions, speeds and accelerations; which are all identifiable and traceable. However, it is considered unreasonable that the simulation be completely deterministic (HOLLANDER; LIU, 2008), and therefore the simulation adds an element of randomness to the calculations. The simulator allows the creation of measurement devices for vehicle counts, car queue counts and travel times, among other results. In practical terms, these aggregate data are considered the simulator outputs and mathematically treated as statistical results.

One possible interpretation of the traffic simulator is that of a mathematical function that receives all settings and parameter inputs, including the duration of the simulation; and delivers the states of the road network at each simulation step, alongside the output metrics that were set by the user and refer to the whole simulation, or to intervals within the simulation duration (e.g. measure vehicle counts on a road only within a specific time interval). The representation of a traffic simulator can be made by a black-box diagram, as shown in Figure 3

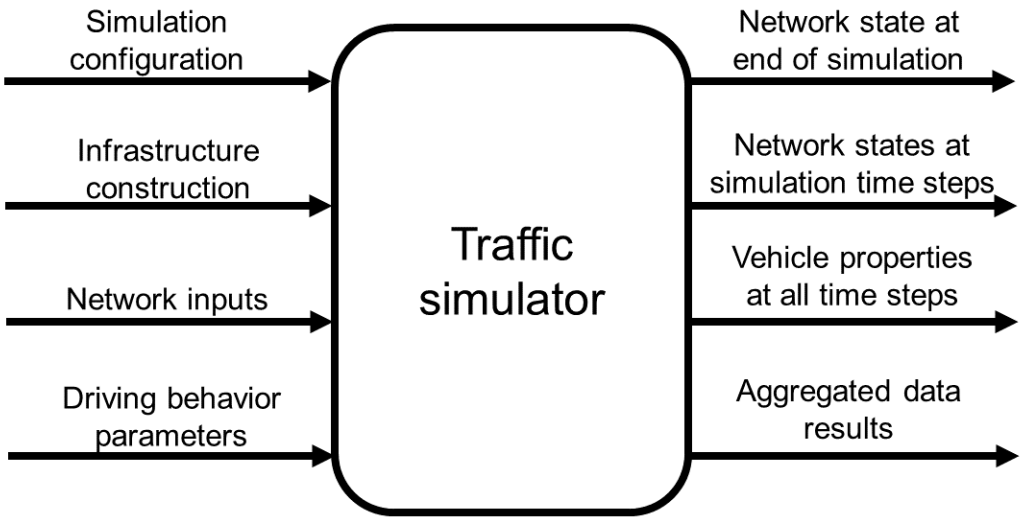


Figure 3 – The simulation abstraction as a black box.
Source: Author.

2.2. Characteristics of the PTV Vissim simulator

PTV Vissim is the simulator used in the experiments. It is a commercially available microsimulator developed by the German company PTV Planung Transport Verkehr AG and is widely used both in the academia and in the industry. This simulator allows the user to build the road networks with a Graphical User Interface (GUI) and to observe the dynamic movement of the vehicles on a map while the simulation steps are running. One key functionality of the simulator license that was available for this research is that there is a Component Object Model (COM) interface enabled (PLANUNG TRANSPORT VERKEHR AG, 2018). The COM interface allows the loading scripts to execute programmed tasks in the simulator in an automated way. This enables the automatic execution of a large number of simulations that can be used in this research for an automatic calibration procedure. An example of the user interface screen of the Vissim software is presented in Figure 4.

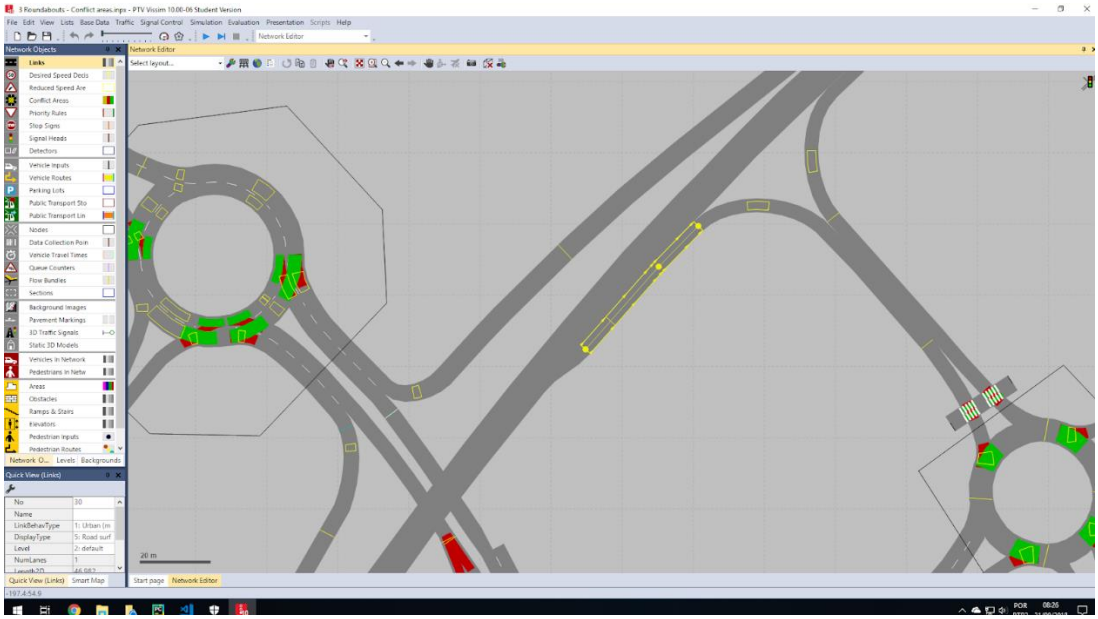


Figure 4 – The main screen of PTV Vissim.

Source: Author.

Based on the works of Llanque Ayala (2013), Bethonico (2016) and Vilarinho (2008), for a specific case of Vissim, the simulator inputs can be classified in the following four groups:

1. **Simulation configuration**, e.g. simulation duration, step duration, statistical model for generation of vehicles and routing algorithms. The step duration will be assumed constant for the experiments of this research. The simulation duration is set by the Vissim user as the simulation scenario is being built. The generation of vehicle volumes for a defined input follows an exponential distribution by default. The routing algorithm is set at the default allocation model if the routing decisions are not being experimented or calibrated.
2. **Infrastructure construction**, which includes the road network links, ramps, traffic lights (but not their timing settings, as further explained) and measuring entities. A new user to Vissim would identify these as the map that the software simulates and these items are often fixed infrastructure (hence the configurable traffic light timing settings are considered in another group). For the experiments, they are all considered constant throughout the methodology procedure, because the experiments start from already built traffic simulations that the user wants to calibrate.

3. **Network inputs**. In Vissim, the network map starts empty at the beginning of the simulation and receives the vehicles that enter the simulation. The network inputs refer mostly to the configuration of these vehicles, such as the vehicle volume distributions that enter the edges of the map, their proportional composition among the different types of vehicles (e.g. buses, cars and motorcycles) and the distribution of route decisions, i.e. the proportion of vehicles that decide to take one route or another at each road fork in the map.
4. **Driving behavior parameters**. For a single vehicle entity in Vissim, there are two models that are considered to define its behavior in this research: the car-following and the lane-change models (BETHONICO, 2016). The car-following model dictates the acceleration and braking behaviors of the car within a lane, i.e. its longitudinal movement along the lane axis, according to the movement of the vehicle in front of it. The lane-change model relies on a decision algorithm dependent on the movement of surrounding vehicles to determine if the vehicle changes lane, i.e. its transversal movement along the lane axis (OLIVEIRA; CYBIS, 2008).

On the output side, the simulator calculates all applicable properties to all vehicle entities in the simulation (e.g. instantaneous speed, acceleration and route decision) at every simulation step. It is possible to access all result and log files from a given simulation, but in practical terms, aggregate data are generated from the measuring devices that are set in the simulation and these are considered the proper simulation outputs (RRECAJ; BOMBOL, 2015).

The simulation outputs can be computed from any interval within the simulation duration. However, because Vissim simulation networks start empty (i.e. with no vehicles in circulation), an initial time interval is usually considered transient and discarded from the simulation results, such as the initial 5 to 15 minutes (MARTE et al., 2017a).

From the COM interface and through script commands, all simulation parameters can be accessed and edited, including the construction of the road network. It is possible to write scripts to perform the full cycle of creating simulations, creating the road networks, loading all vehicle inputs and their characteristics, setting the

measurement devices, running the simulations and saving the results (PLANUNG TRANSPORT VERKEHR AG, 2018).

However, the proposed methodology in this research is intended to support traffic engineers in their experimentation and analysis work with the traffic microsimulator. Therefore, the methodology starts with built simulation scenarios, as explained in the solution proposition in Chapter 4.

3. Literature review

3.1. On the selected parameters for calibration

In the references, where traffic engineers document the calibration process of a traffic microsimulation, such as in the theses from Llanque Ayala (2013) and Miller (2009), the simulation configuration and the infrastructure construction are mostly considered constant. There is a group of authors who focus on the calibration of driving behavior parameters (AGHABAYK et al., 2013; LLANQUE AYALA, 2013; OTKOVIC; TOMMAZZI; SRAML, 2013; MILLER, 2009; VILARINHO, 2008) related to the car-following and lane-changing models, and usually in smaller road networks (e.g. an intersection, roundabout or highway section).

By contrast, Chu et al. (2003) criticized such calibrated models as incomplete and proposed a calibration procedure for the routing behavior of a larger and more complex network and by utilizing an Origin-Destination (O-D) matrix as reference. This routing calibration is proposed to be an additional step after the driving behavior calibration. Additionally, Tettamanti et al. (2015) used a Genetic Algorithm to calibrate the traffic volume demand on a group of roundabouts, and therefore their variables of interest were in the category of network inputs.

An insight from the literature review is that there is no single set of parameters to be calibrated according to all authors. Llanque Ayala (2013) and Vilarinho (2008) respectively investigated the selection of parameters and an analysis of parametric sensitivity of the simulations. The conclusion is that the subset of parameters (and the other subset that can be left on the program defaults) depends on a case-by-case basis. Furthermore, Punzo, Montanino and Ciuffo (2014) reinforce the idea that a subset of parameters might be sufficient to consider the calibration of a traffic microsimulation adequate.

Chu et al. (2003) observed that more limited traffic simulations (e.g. a single intersection or roundabout) are likely to be more sensitive to driving behavior parameters than larger and more complex networks, in which case network inputs such as routing decisions, vehicle compositions and volumes become more relevant.

3.2. On the calibration procedure

Rrecaj and Bombol (2015) identified that calibration is widely considered an optimization problem. Furthermore, there is a convergence point in regard to calibration in Vissim, with the use of Genetic Algorithms being the most common and identified as the state of the art. As an optimization problem, calibration is the iterative process of reaching the maximum value of an objective function or alternatively the minimum value of a cost function.

Hollander and Liu (2008) conducted a survey on calibration techniques where various cost functions are used, and they defend that, as a first approach to calibration, quadratic errors are the most appropriate form of measuring costs to be minimized. They argue that:

- Positive and negative errors should not cancel each other.
- The quadratic measurements place more penalty on larger errors.
- Smaller statistical errors may be tolerable in microsimulations due to the stochastic nature of traffic.

The works of Llanque Ayala (2013), Vilarinho (2008), Chu et al. (2003) and Miranda (2018) are examples where the iterations of calibration were not automated. For a given calibration set, i.e. a set of determined values for all simulation parameters, the cost function is computed; the steps are tracked and the engineer chooses new calibration sets with the help of sensitivity analysis of the cost function and with some room for guessing, directed by the familiarity with the traffic simulator due to previous experiences. Both Chu et al. (2003) and Hollander and Liu (2008) were critics to the repetitive nature of the calibration task.

Henclewood et al. (2017) propose a calibration procedure that relies on automation of simulations and a Monte Carlo approach. The procedure runs a large number of simulations with random parameters and measures squared errors; 1000 simulations were performed to extract 93 and 34 calibrated models for noon and evening periods in their location of testing, thus showing the calibration differences between different time periods at the same location. Their hypothesis is that disaggregate individual vehicle data ought to be obtainable with the expansion of

vehicle-based data collection technologies. However, by comparing results of disaggregate data, the authors came to the conclusion that their solution is not scalable for larger networks and real-time applications.

On the other hand, Bethonico (2016), Tettamanti et al. (2015) and Aghabayk et al. (2013) automated the calibration procedure by using Genetic Algorithms to solve an optimization problem. The Genetic Algorithm is a search heuristic that iterates through groups of calibration sets called generations; inspired by evolutionary theory, the new generations are created from combinations of the most successful calibration sets from the previous generation, with an added element of randomness. Success is determined by maximizing a fitness function (or alternatively minimizing a cost function) and, in practical terms, the algorithm delivers a successful generation of calibration parameters after a large arbitrary number of iterations. However, there is no mathematical guarantee that the algorithm eventually delivers the optimal calibration set, i.e., the set which absolutely minimizes the cost function.

Shafiei, Gu and Saberi (2018) use machine learning techniques to calibrate simulations at a mesoscopic level. In their research, the subject of calibration is a Dynamic Traffic Assigment (DTA) model and machine learning is used to cluster archived data from the traffic authorities to identify the normal daily traffic dynamics, as reference for the DTA. Furthermore, as mesoscopic simulations deal with the abstraction of homogeneous groups of vehicles, clustering is also used to identify groups of roads for the DTA model, implicitly from the data (e.g. roads of 2, 3, or 4 lanes and the desired speeds of 80 or 100 km/h).

Otkovic, Tollazzi and Sraml (2013) proposed a calibration methodology for Vissim microsimulations on a roundabout scenario that uses Artificial Neural Networks (ANN). Within machine learning tools, the ANN is a supervised learning model that is trained with the input-output pairs of the desired phenomenon, which can be either a classification or regression problem. Once trained, the ANN should model the transfer function between inputs and outputs in a generalized way. In their methodology. ANN models are trained and compared to best fit the direct transfer function of the Vissim microsimulator as a first step, i.e., the relationship between simulator inputs and generated outputs is modeled by ANNs and described as a

simulator prediction function. This prediction function delivers simulator output estimates directly and is less time-consuming than running a full VISSIM simulation.

Furthermore, the calibration methodology of Otkovic, Tollazzi and Sraml (2013) iterates through calibration sets by utilizing the ANN estimates to score results more quickly (replacing Vissim as a first approach). Their ideas share similarities with the calibration methodology proposed in this research, as both use Vissim simulation results to generate examples for ANN training.

However, key differences are that this research does not intend to model the direct transfer function of a traffic microsimulator, but to use machine learning to identify other patterns in traffic calibration, as it will be further explained in the formal methodology proposal in Chapter 4.

3.3. On Machine Learning techniques for traffic problems

Machine Learning is a field of computer science that uses statistical methods to allow computer programs to learn a task, as opposed to being directly programmed. The task is learned from a variety of methods applied to example data, such as pattern recognition, regression or clustering (SAMUEL, 1959; KOZA et al., 1996).

Problems of interest of machine learning include classification of sample data and regression of transfer functions between input and output data of an observable phenomenon. Both regression and classification problems have appeared in the literature of traffic engineering, and machine learning has been proposed as a solution. One machine learning technique of interest to this research is the Artificial Neural Network (ANN), a class of computing models that are inspired by biological neurons and are used for machine learning of regression and classification tasks (VAN GERVEN; BOHTE, 2018). The subclass of deep learning ANNs are used for pattern recognition and considered by some authors to be the state of the art in machine learning (CIRESAN; MEIER; SCHMIDHUBER, 2012). Other varieties include convolutional ANNs, recurrent ANNs, and the simplest ANN model is the feedforward neural network (SCHMIDHUBER, 2015), a more detailed explanation of the model used in this research for the proof of concept is in Chapter 5.

Zhou, Qu and Li (2017) proposed a recurrent ANN model to predict traffic oscillations in the car-following model of microsimulations. The recurrent ANN is a type of network designed to be trained from time-series sample data and deliver predictions of future data. This model is used to replace the car-following models that are usually sets of equations for calculating the interaction between cars in a same road lane (BETHONICO, 2016). The recurrent ANN had a better performance than the classical models in predicting the trajectory of subsequent vehicles; the better than the classical models, the farther the subsequent vehicles were from the reference vehicle.

Conversely to the driving behavior parameters, Tang et al. (2016) used a type of network called fuzzy ANN to estimate travel times in the real world from road loop detectors, thus the ANN implicitly computes a traffic simulation and delivers the travel time outputs, if we consider the loop counts and speeds as the inputs of this hypothetical simulation.

In the context of Smart Transportation and not necessarily traffic simulation calibration; Alkheder, Taamneh and Taamneh (2017) used ANNs to classify the severity of traffic accidents from the data filled in the police reports. The goal of the ANN was to estimate how severe a new accident is from the data that is provided in the emergency call, and therefore the traffic authorities are better informed to send appropriate help to the location. In an analogous way and also in the context of Smart Transportation mapping, Ngwangwa et al. (2010) built a classifier of road defects with ANNs based on the pattern recognition of road accelerometers.

As another example of ANN use outside of traffic calibration, Chen et al. (2018) used a combination of Genetic Algorithms and ANNs to create a system to predict rear-end collisions. Instead of training a single ANN to predict the collision from car-following patterns that are fed from the Vehicle-to-Infrastructure (V2I), Vehicle-to-Vehicle (V2V) and Global Positioning System (GPS) infrastructures; their methodology generates collections of ANN that compete in a Genetic Algorithm to deliver near-optimal ANN models. Their proposed model was more accurate than kinematics equations in the proof of concept.

Deka and Quddus (2014) used the pattern recognition properties of ANN to propose an accident mapping tool. They argue that police reports are inaccurate in their scenario of study, and an ANN model was trained to estimate the relationship between the given coordinates of the accident and the matching coordinates of the location on an actual road and with improved accuracy.

Finally, Zhang and El Kamel (2018) proposed a novel driving model that is solely based on an ANN learning vehicle trajectories from sets of examples, thus the resulting ANN modeled the transfer function of the driving behavior model. They also implemented a traffic simulator to demonstrate how to replace the classical equation-based car following models by the ANN transfer function. Their conclusions were positive in regard to the novel model being able to reproduce a broader variety of vehicle behaviors, i.e., calibration between the novel model and real-world data was more accurate under different scenarios.

This research focuses on the use of feedforward Neural Networks to build a regression model of the calibration function, with emphasis on the experimentation of different topologies and optimizer strategies.

4. Research proposal

4.1. Research gap

Smart Cities have been highlighted as one of the current technological trends in urban scenarios, and Smart Transportation has been identified as one of its key elements (XIONG et al., 2012). Furthermore, Chu et al. (2003), Fang and Elefteriadou (2005), and Hollander and Liu (2008) have shown evidence of the advantages of traffic simulators for Smart Transportation planning. Additionally, traffic simulators are virtual environments that allow experimentation of research in traffic engineering, e.g. Zhang and El Kamel (2018) propose a new behavior model to replace the car-following model that is widely used in traffic microsimulations.

Calibration has been identified as one of the challenging steps of the preparation of traffic simulation scenarios due to it being repetitive and time-consuming (CHU et al., 2003). Furthermore, automation and the interpretation of calibration as an optimization problem has led to positive results in regard to improving accuracy, reliability and reproducibility of the calibration task (RRECAJ; BOMBOL, 2015). Genetic algorithms were identified as the most commonly used optimization methods for calibration, and while their results are promising, one drawback is that a large number of simulations are performed throughout the iterations for the heuristic to achieve a small set of highly specific solutions to the single scenario under calibration. In case of new traffic simulations using the same input and output variables (with some constraints that will be further explained), the calibration process by optimization needs to start over from the beginning.

Figure 5 shows an example of a Genetic Algorithm. The hypothesis inspired by the theory of evolution is that successful candidates are created by a combination of successful candidates from the previous generation that survived the called fitness function, which evaluates their performance.

Each iteration includes the recombination of the competing candidates into new candidates, highlighted in green, and the following discard of candidates that failed the evaluation of the fitness functions, marked with a red X. In (a), A, B, C, D and E represent a first generation of candidates for the optimal parameter set and upon a round of evaluation through the fitness function; C and D are discarded because their

performances are the two worst. In (b), A, B and E were left from the previous step to recombine alongside mutations α and β for a new set of candidates that still competes with A, B and E. In this round, B and E are discarded because of their inferior performance.

The process continues iteratively in (c), where A, (A+B+α) and (B+E+β) left from the previous step recombine to generate (A+B+E+γ) and in this round, A and (B+E+β) are discarded. In (d), the survivor candidates (A+B+α) and (A+B+E+γ) recombine to generate (A+B+E+δ). Finally, the algorithm ends when, upon evaluation and discard of the candidates with the worst performances (A+B+E+γ) and (A+B+E+δ); (A+B+α) is left as the only candidate and considered close to optimal in this heuristic method.

There is a possibility of using evaluation results from the intermediate candidates as a starting point for a new calibration, but the reuse of data was not mentioned as a key element in any of the references from the literature review.

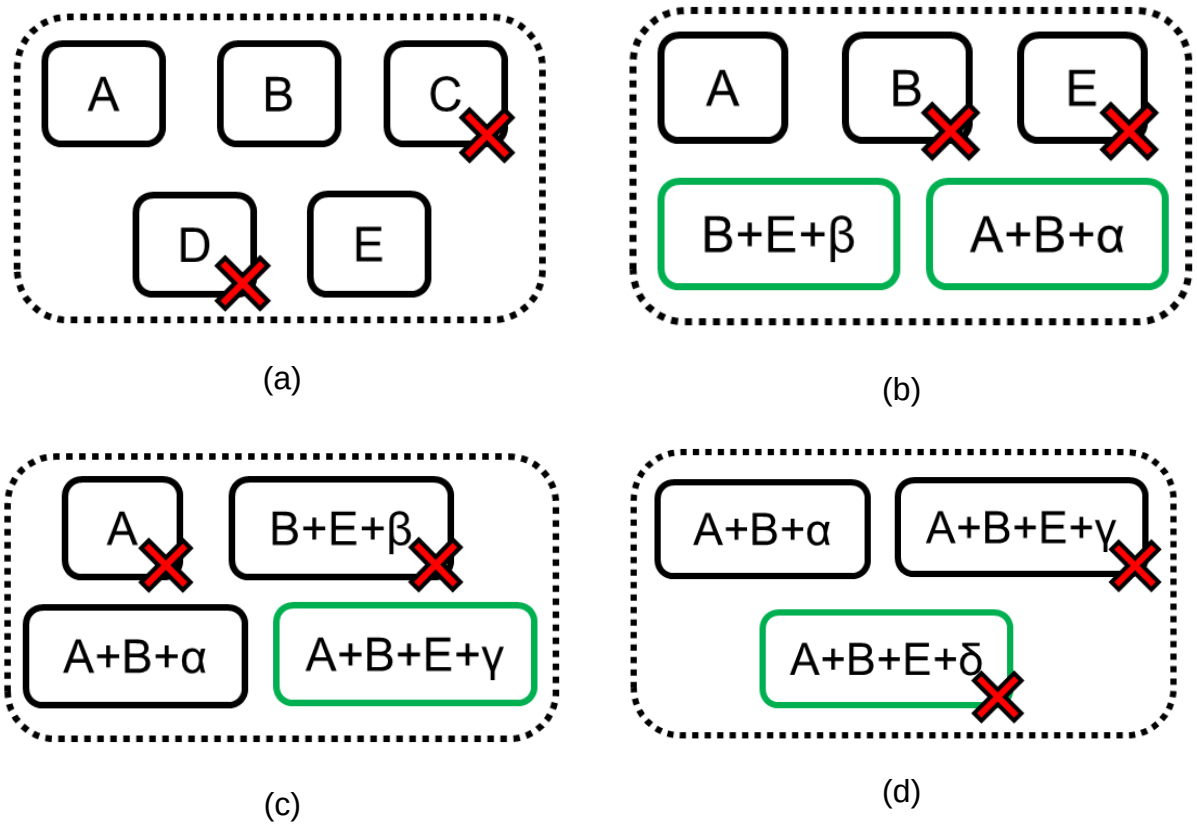


Figure 5 – Genetic Algorithm iterations process.

Source: Author.

Aghabayk et al. (2013) proposed a methodology for Vissim calibration that relies on parallelism to improve computing performance. They worked directly with the company that developed the Vissim software, PTV AG; and their research's underlying assumption was that traffic microsimulations run on Computing Processing Units (CPU) in most cases, which imposes restrictions in scalability for larger networks and methodologies that require large numbers of simulations. Moreover, Xu et al. (2014) and Vu and Tan (2017) conducted research on the use of Graphical Processing Units (GPU) to improve performance of mesoscopic simulations with focus on scalability of network sizes. Their research indicates that CPU is the major trend in time-stepped simulations across the simulation software alternatives, thus GPU-based simulators are still under research and not considered widely available.

Given that a bottleneck has been identified in scalability due to the computing architecture mostly used for traffic microsimulators, and that the currently used

calibration methods need a large number of simulations to be performed anyway. In this way, one possible research opportunity is in the direction of data reuse and the development of generalist procedures to calibrate multiple traffic microsimulations at once. Furthermore, as opposed to calibrating a single simulation through the optimization approach and restarting the process for every new simulation, the modeling of a generalist calibration function for the simulation of a road network map under different traffic scenarios has been identified as a research gap.

4.2. Research question

From the identified opportunities in the previous item, an elaboration of the research question is:

- *How to create automatic calibration models that are reusable across similar traffic simulations?*

From the research question, it is important to clarify what are simulations that are considered similar as a first step. The solution proposal to the question is a methodology, and the validation experiments are developed by using PTV Vissim as the traffic simulator of choice. Referring back to Chapter 2, Vissim (and other simulators) performs the simulation function from the set of all the simulator inputs (e.g. driving behavior parameters and network inputs) to the set of all simulation outputs (e.g. simulation states at each step and output aggregate metrics).

The proposed categories of simulation inputs are simulation configuration, infrastructure construction, network inputs and driving behavior parameters. In the scope of this research, similar traffic simulations share the same values and settings for simulation configuration and infrastructure construction, which are in practical terms simulations over the same road network map and with the same duration. Similar simulations also share the listing of network inputs and driving behavior parameters. However, the values of those variables are not fixed. Figure 6 highlights the fixed and variable inputs for PTV Vissim alongside the aggregated data results, the only outputs used for calibration in practical terms.

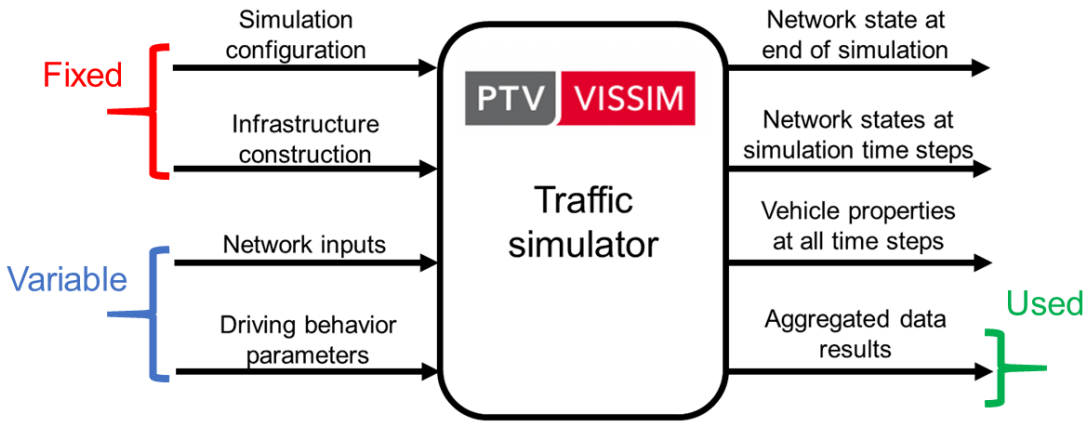


Figure 6 – Fixed and variable inputs for similar simulations.
Source: Author.

The idea behind the research is to create automatic calibration models that can be used for various simulations of the same map, but under different traffic conditions (e.g. the same avenue or intersection; but under various traffic demands, varying vehicle compositions, varying routing decisions and various driver profiles), such as exemplified in Figure 7.

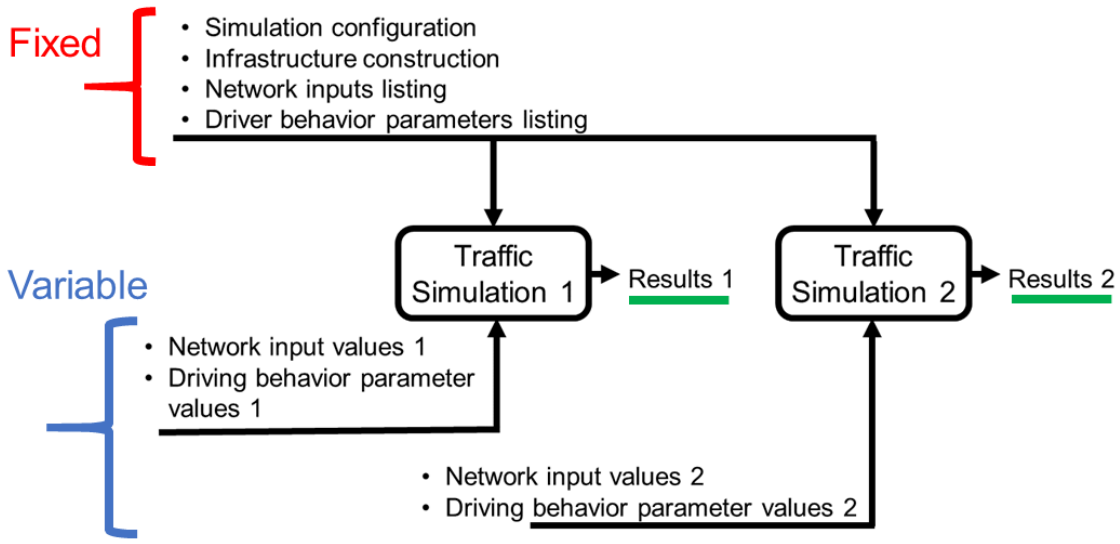


Figure 7 – Relationship between similar traffic simulations.
Source: Author.

4.3. Statement of the research proposal

This research proposes a methodology to create automatic calibration models that are reusable to calibrate traffic microsimulations that are considered similar among themselves. The goal of the methodology is to perform all tasks necessary to deliver a reusable automatic calibration model from an initially uncalibrated traffic microsimulation setup. This calibration model is created through means of supervised learning, a branch of techniques in Machine Learning and, particularly, Artificial Neural Networks (ANN or simply Neural Networks for short, NN) are used in the validation experiments of this research.

The Neural Networks perform the regression of a non-linear function with multiple inputs and outputs, and the learning of this behavior is conducted by mirroring a dataset of examples of input-output pairs. One assumption is that the examples in the training dataset are sufficient to represent the behavior of the function to be learned, across a wide range of its possible input and output values (HAYKIN, 2009). Once the Neural Network is trained and receives new inputs, it should deliver outputs according to the behavior that it learned in the training phase.

The methodology is, therefore, divided into two parts: the creation of the training dataset with sufficient input-output examples of calibration, and the training of a Neural Network to perform such task with adequate accuracy. A novel approach is to use a large number of traffic simulation results to build the dataset of what is defined as the calibration function, which reverses the input/output order of the traffic simulator, i.e. the simulation outputs are used as inputs of the calibration function, and the simulation inputs are the desired outputs of the modeled calibration function.

This relationship between the simulation and calibration functions is represented by diagram shown in Figure 8. According to this approach, values of input parameters should be mapped into values of aggregate outputs and vice-versa. As it will be further described in the experiments, the range of values for both input parameters and aggregate outputs are well-known for the applications of traffic simulations and traffic engineers who use the proposed methodology can spot inconsistencies after the calibration model delivered the estimations of values for calibration.

Examples for identifying inconsistencies are that vehicle traffic volumes should be positive numbers limited by the maximum capacity of the road segment, travel times should be positive and range from seconds to hours; the ratio of route decisions at a road fork for one side should range between 0% to 100%, and average speeds should be positive numbers and limited by what is physically possible on the road segments.

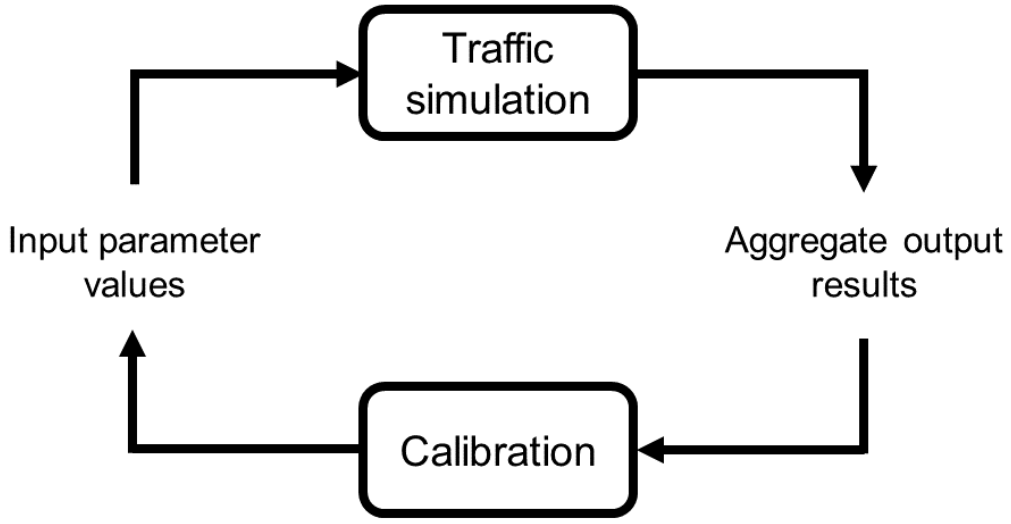


Figure 8 – Relationship between simulation and calibration functions.
Source: Author.

Finally, as described by Hattab and Motelica-Heino (2014) in their research with Neural Networks for regression in a different application, there is a problem in modeling an inverse function because of the probable non-uniqueness of the input-output pairs, i.e., different input parameter values can lead to identical values of aggregate outputs, in such a way that tracing the reverse function has a high chance of error. Measures to minimize the effects of the non-uniqueness issue include verifying inconsistencies in the values of the variables and evaluating the Neural Network error across a large number of testing examples.

In the implementation, the software describes the performance metrics for each variable to be calibrated; hence the traffic engineer can choose to ignore some of the suggestions from the calibration model. The proposed methodology is explained by the list of tasks below.

Start with: Simulation road network.

1. **Dataset creation**
   1.1. Generate random values for inputs to be calibrated.
   1.2. Run multiple simulations.

1.3. Exchange simulation inputs and outputs to create the dataset of the calibration function.

2. Neural Network training

2.1. Separate training and testing datasets.

2.2. Train neural network as a regression model.

2.3. Evaluate which inputs are estimated correctly.

End with: Regression model for automatic calibration.

The global input is a traffic microsimulation of interest, with the simulation configuration and infrastructure construction inputs set; and with the network inputs and driving behavior parameters for calibration defined, but not necessarily with the values set. The methodology starts by creating random sets of values for the variables intended to be calibrated and running multiple traffic simulations.

So far, the methodology resembles the optimization methods from the literature review, in regard to the large number of simulations being performed from randomly-generated sets of inputs. This process in both cases is automated by using the COM interface to launch the simulation jobs in the software, but the difference is that in this methodology the collection of simulation results is used to train a machine learning tool, as opposed to being filtered out in order to create new sets of more specialized simulations (e.g. new generations of simulations in the case of genetic algorithms).

The regression model for automatic calibration is created with an Artificial Neural Network, which is trained with a selection of the dataset and evaluated with the remaining examples in order to eliminate bias, i.e., it is possible that a Neural Network be overly trained and memorizes the training examples with small error, which is undesirable and not the network learning the relationships between inputs and outputs of the dataset.

Once the calibration model has been created, new similar simulations can be calibrated by applying the calibration function directly (thus by deployment of the ANN model), using the desired simulation outputs, i.e. the calibration reference, as the inputs of the calibration tool.

Finally, the proposed methodology is intended to automate the calibration process that would be traditionally performed by the traffic engineer manually or with some degree of repetitiveness. Furthermore, it focuses on the calibration of the simulation inputs that were identified as of interest in the literature review, and advances the identified state of the art methods by making the calibration model reusable for other simulations, within the constraints that define them as similar.

For a completely new simulation, which can be interpreted as a new map location, the methodology must be followed from the beginning, but with it a new group of similar simulations becomes available for the reuse of the newly created calibration model.

5. Methodology implementation

5.1. Experimental parts to validate the methodology

According to the proposed methodology, the validation experiments can be divided into two parts: building a training dataset that represents the calibration function and creating the neural network models. Additionally, all steps can be automated in a software implementation. The first part uses traffic simulation software that are commonly found in the traffic engineering references of the literature review, being an implication that some widely used alternatives are proprietary. By contrast, the literature of machine learning algorithms is mostly open-source and the experimental setups use the collaborative libraries that are generally well-documented.

5.2. Traffic microsimulation software

In the first part of the methodology, the traffic microsimulation software is used to build a dataset that represents the traffic simulation model. The PTV Vissim, which is a proprietary traffic microsimulator from PTV Planung Transport Verkehr AG (2018), was chosen from the software alternatives mentioned in the literature review. One of the reasons for this choice was because the current state-of-the-art calibration methods with Genetic Algorithms have been conducted with simulations on Vissim, as referenced in the survey of Rrecaj and Bombol, 2015. Another reason was because of the availability of a software license with COM interface, enabling the automation through scripts.

Figure 9 presents an overview of a loaded road network in Vissim. Even though the COM interface allows commands for the creation of entire network maps and scenarios, traffic engineers usually recreate real-world networks with drawing tools and the reference map layer in the background. The same editing alternatives apply to the various physical and abstract entities in the map (e.g. vehicles, traffic lights, route decisions and driving behavior parameters), as they can be either input by the user through the button panel or directly edited with script commands. From the perspective of the user, the button panels are usually preferred, whereas the

proposed methodology takes advantage of the COM interface for automation of simulation jobs with full control.

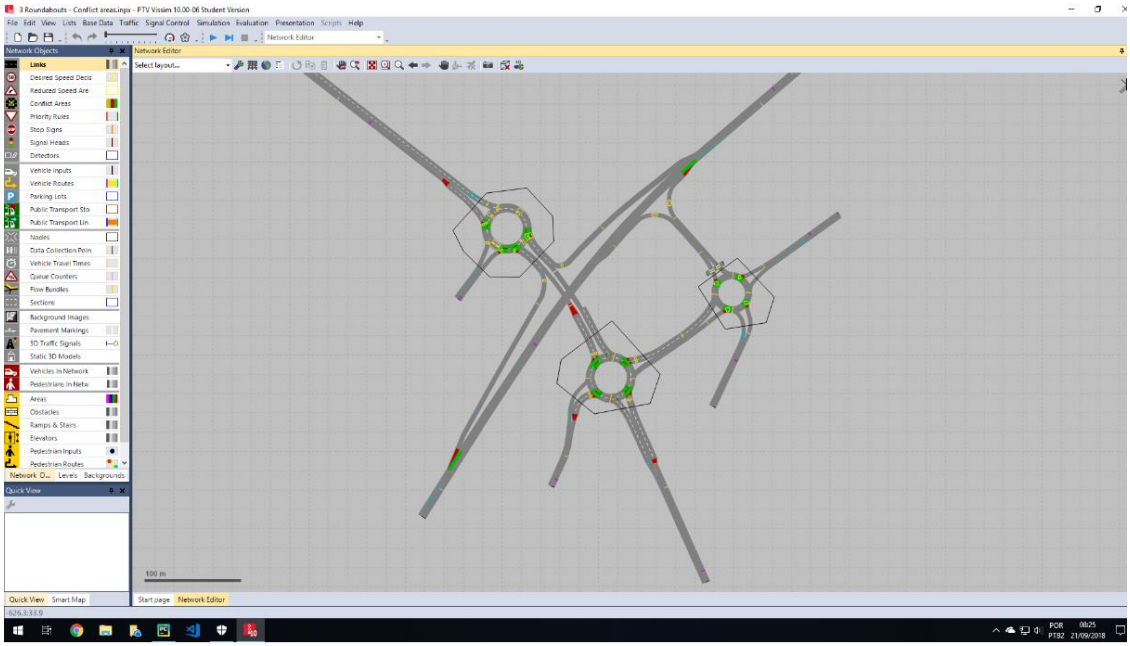


Figure 9 – Overview screen of PTV Vissim.

Source: Author.

Vissim is used to run a large number of simulations, in order to use their results to build a dataset that represents the calibration function, and to do so a list of input and output variables of the simulator must be defined. As described in Figure 8, the calibration function's inputs and outputs, in the dataset built, are respectively the simulator outputs and inputs of normal simulation runs. Furthermore, a well-defined list of input and output variables of the simulator determines all that is taken into consideration during calibration.

As discussed by Punzo, Montanino and Ciuffo (2014), not all variables are equally important for calibration. The gradient descent is the principle underlying the learning methods for the Neural Networks, as it will be further explained, and it emphasizes learning from the strongest input-output relationships, i.e. those with highest correlations. Therefore, the Neural Networks offer flexibility to the listing of inputs and outputs from Vissim. An insufficient number of listed variables may result

in unsuccessful calibration attempts, whereas an excessive number of variables adds computational complexity to the Neural Network training phase and may add the noise of input-output relationships that are weakly correlated.

The simulation inputs and outputs vary according to the experiment and network map. From the performed experiments, the variables that belong to each group are listed in the following, even though not all variables were necessarily used in all experiments.

- **Vissim inputs**: Vehicle volumes, vehicle volume distributions, vehicle compositions, traffic light timing settings, static route decisions and driving behavior parameters.
- **Vissim outputs**: Vehicle volume counts, vehicle average speeds, vehicle travel times and vehicle queue counters.
- **Vissim parameters kept constant**: Simulation duration, which depends on the scenario of interest; simulation step, which is kept on default unless there is the suspicion that it interferes with the performance of the calibration function modelling. Also the distribution of the generation of vehicles and the choice of algorithm for routing allocation are kept at the default options and values.

All the listed Vissim inputs and outputs parameters are accessible through the COM interface. It should be noted that they are variables of interest to the traffic engineer, as discussed in the literature review.

It has been empirically observed on a default installation on the Windows Operating System, the computational processing of the simulator models run mostly on the Central Processing Unit (CPU). Whereas Xu et al. (2014) and Vu and Tan (2017) make considerations about parallel processing and the use of a Graphics Processing Unit (GPU) to improve performance. Therefore, parallelization is a direction of future research once the proposed methodology is validated, since a GPU should be already available to optimize the training of the machine learning model.

Additionally, the number of simulations that were run in each experiment in order to build the training dataset was determined empirically and is another topic for research and experimentation. The lower and upper boundaries for the number of simulations are the values that suffice to represent: the behavior of the simulator with generalization, for the former; and the number of simulations that takes a time slot that is manageable in practical terms, for the latter.

5.3. Dataset built

The dataset of the calibration function has the format described in Figure 10, which highlights that the input and output roles are reversed depending on whether the simulation or calibration function is being performed. The columns list the input and output variables, and the rows list the simulation runs. The simulation input variables (referred to as parameters) are set to a variety of randomly generated values within a defined range that is appropriate to the application (e.g. vehicle volumes are non-negative integers that range from 0 vehicles/hour to the maximum theoretical capacity of the road, according to a uniform distribution).

In the proposed methodology, the user defines the value ranges and number of simulation runs after building the infrastructure map and choosing the variables of interest. To keep the scripts within the same programming environment, although not necessary, the tables can be managed with Python libraries such as NumPy or Pandas (2019), which have optimized functions for operations of large tables and random-value generation. Spreadsheet software like Microsoft Excel (MICROSOFT CORPORATION, 2019) or other open-source alternatives can be used in order to quickly generate the variable values and confirm their statistical distributions.

Example of dataset format

Vehicle volume 1 (1/h)	Vehicle volume 2 (1/h)	Route Decision 2	Travel Time 1 (s)	Average speed 1 (km/h)	Vehicle count 1
600	120	0.4	45	26.55	357
550	2300	0.5	140	5.88	12
1200	100	0.1	30	34.5	689

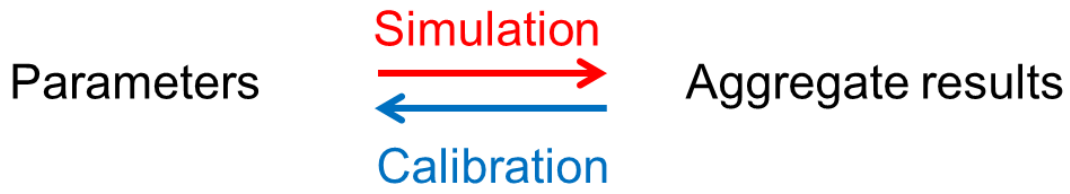


Figure 10 – Example of input-output dataset.

Source: Author.

5.4. Neural network training

The second part of the experiments uses open-source software that is trending in machine learning research to implement Artificial Neural Networks (ANNs)

Python (PYTHON SOFTWARE FOUNDATION, 2018) is a general-purpose interpreted programming language that is used both for the COM interface scripts in Vissim and the development of the machine learning models. TensorFlow (ABADI et al., 2016) is a library created to implement Neural Networks in a scalable manner and is GPU-enabled to improve performance through computing parallelism. Keras (CHOLLET, 2018) is a library that works as the frontend of TensorFlow as is used for its simplicity to create feedforward Neural Network models. It also has the optimized implementation of various data preprocessing methods, optimizer methods for the network's weight updates, evaluation methods and other training and validation adjustments.

The main approach for modelling the defined calibration function is with an ANN of the type Multilayer Perceptron (MLP) (HAYKIN, 2009). The perceptron is a computational model inspired by the biological neuron cells, presented in Figure 11. The artificial neuron computes the weighted sum of the inputs and delivers the result

of the activation function, which represents if the neuron was triggered or not. The MLP is the concatenation on perceptron's in order to create a larger and more abstract model, presented in Figure 12.

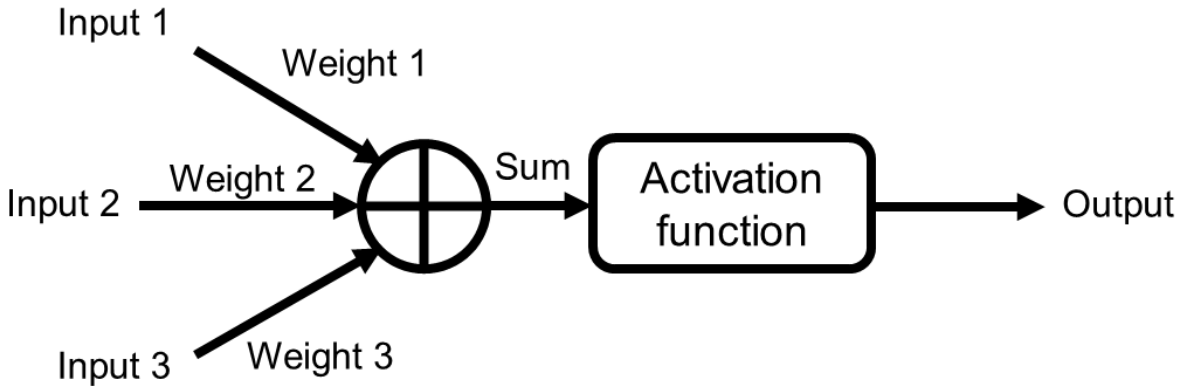


Figure 11 – The perceptron model.
Source: Author.

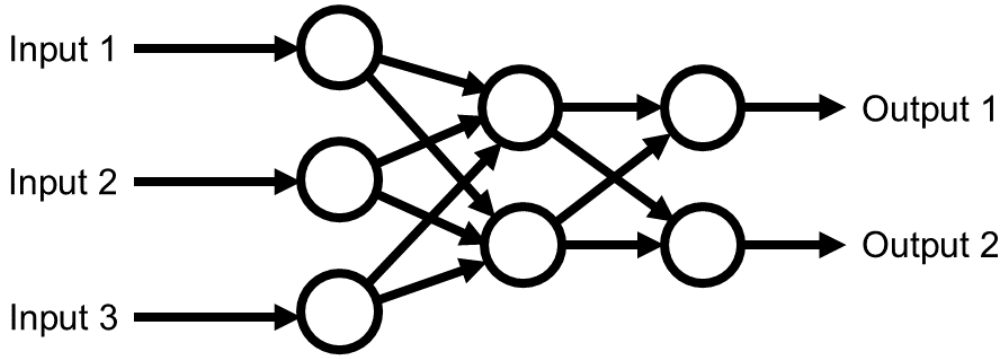


Figure 12 – The Multilayer Perceptron.
Source: Author.

The generalization capacity of MLPs makes them adequate as both regression models and pattern classifiers. The first refers to modelling multivariable continuous non-linear functions based on data samples and interpolation properties. Whereas the second refers to delivering the probability of an entity belonging to groups after training the model with previous entity examples and their features.

The configuration of the neurons and their connections in a graph is referred to as the ANN topology; whereas training is the process of adjusting the connection weights between neuron layers in order for the MLP to reproduce the input-output results of the training dataset with minimized error. The training process is iterative and in the MLP uses the Backpropagation algorithm (HAYKIN, 2009), which involves computing the contributions of each connection weight to the output error by means of the gradient function of the error function.

One of the simplest strategies for training and evaluating an Artificial Neural Network is to divide the dataset of examples for learning into a training dataset and a testing dataset. The training phase of the Neural Network is then conducted using only the training samples and, once the training is finished and there are no more adjustments to the network, the testing samples are used to evaluate the performance of the Neural Network (e.g. mean quadratic error in regression problems and accuracy in classification problems) in a deployment scenario. It is important that the training phase be blind to the examples that are used for testing, thus eliminating bias from the network. The traditional dataset split is presented in Figure 13, and Figure 14 presents the block diagram of the training process.

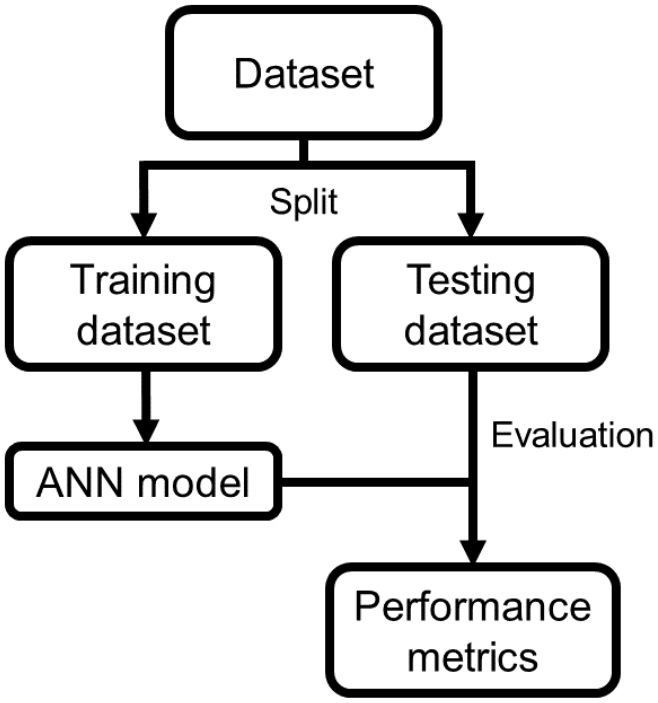


Figure 13 – The training and testing dataset split.

Source: Author.

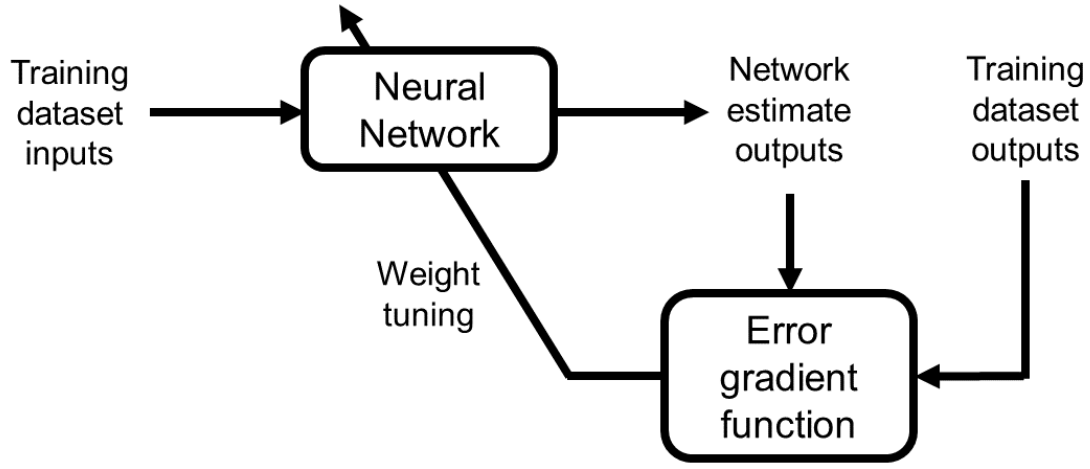


Figure 14 – Block diagram of the training process.

Source: Author.

To perform the iteration steps for the weight tuning of the neurons, the error function (between the outputs of Neural Network and the desired outputs from the training dataset samples) is calculated and can be visualized as a multi-variable surface, as presented in Figure 15. Furthermore, the set of coordinates of the lowest

point on the surface represents the set of weights that minimizes the error function and thus a good-performing Neural Network. The gradient vector of the error function surface is calculated to indicate the slopes of the function, and the training phase is the iterative navigation through the error function surface to search for the point of lowest error (by following the opposite direction of the gradient vector). Finally, the method is heuristic and randomization of the starting point is used to prevent the recurring trapping of this optimization method in local minima while searching for the global minimum.

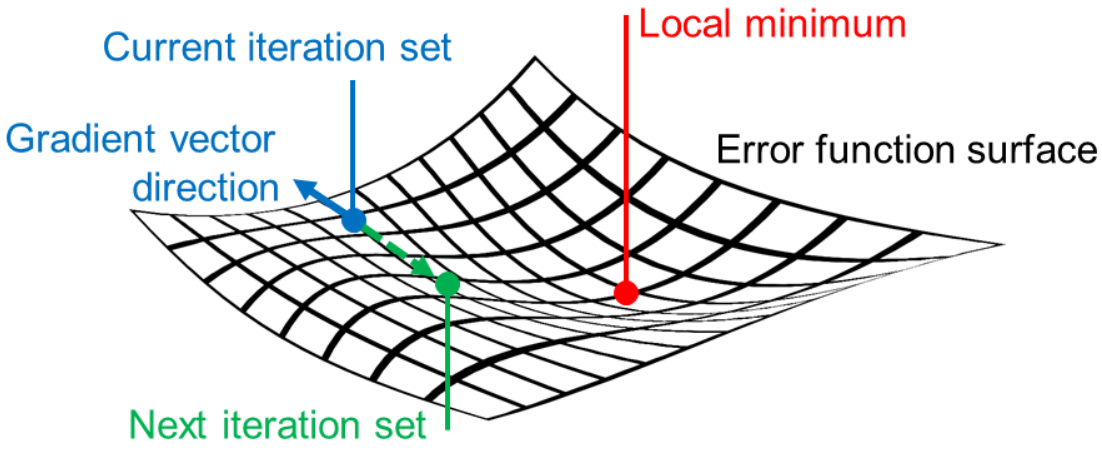


Figure 15 – Visualization of training iterations on the error function surface.

Source: Author.

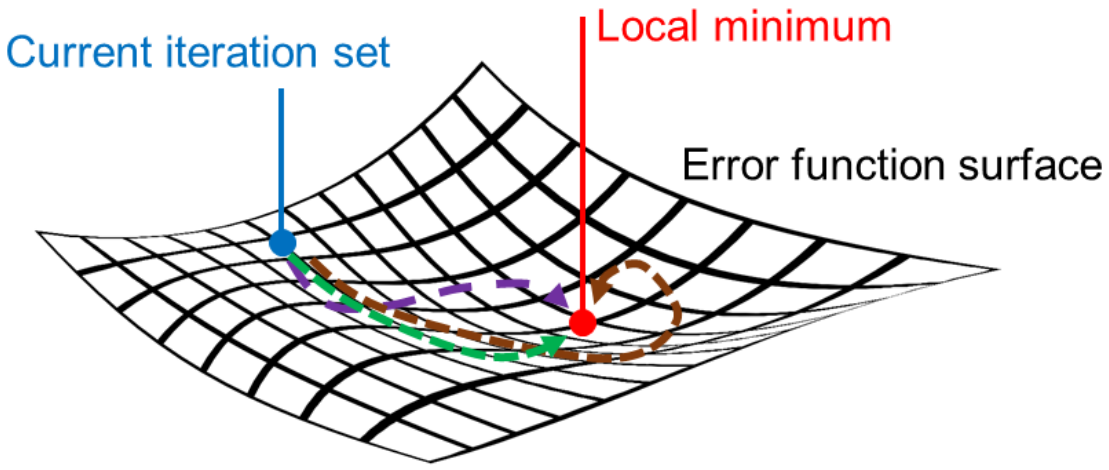


Figure 16 – Paths to the local minimum with different hypothetical optimizers.

Source: Author.

Apart from the randomization of the starting conditions, the training iteration steps can follow a variety of strategies to compute the gradients efficiently and navigate the error function quickly and accurately, as presented in Figure 16. The strategies involve the computation of adaptive iteration step sizes, momentum corrections to prevent stagnation on function plateaus or local minima; the selection of activation functions in the neurons with derivatives that are fast to compute (the derivatives are necessary to compute the error function gradient) or the division of the training samples into batches according to a computation advantage. These strategies depend on the application and are usually chosen through experimentation during the development of the Neural Network, as discussed by Ruder (2016).

The simple dataset split into training and testing from Figure 13 can be used in an initial approach to train Neural Networks. However, there are disadvantages in using only training samples to evaluate the network performance during the training phase and to determine when the iterations should stop. In this approach, there are two options for the stopping criteria.

One alternative is using the training error as a reference for the stopping threshold. The Neural Network is a computing system that learns the general relationships between inputs and outputs in the dataset that it is learning, as opposed to strictly memorizing the input-output pairs, and therefore evaluating the network with the same training samples introduces an undesirable bias, because the network is being evaluated solely on examples that it knows from training.

The second alternative is to stop training at a fixed amount of training epochs, which are the full rounds of iterations across the entire training dataset. Determining the best number of training epochs is necessary to avoid both underfitting and overfitting of the Neural Network. Underfitting means that there have not been enough training epochs for the network to reach the surroundings of a local minimum on the error function surface, and thus the training error is high. Overfitting, on the other hand, refers to the network exhaustively learning the regression model from the training data and instead memorizing the training samples.

Figure 17 shows the difference between a correctly fitted and an overfitted regression model for a set of training data. The black dots on the XY plane represent

samples from function to be modeled by the Neural Network as a regression model. The blue curve approximates the general trend of the data and should model accurately the underlying phenomenon of the function. By contrast, the red curve represents an overfitted regression model. In this case the exhaustive training of the Neural Network, with the goal of minimizing the training error, has led the regression model to also approximate the statistical noise of the data points. The consequence is that for a new (blind) set of testing points, the red overfitted curve has a higher regression error than the blue curve.

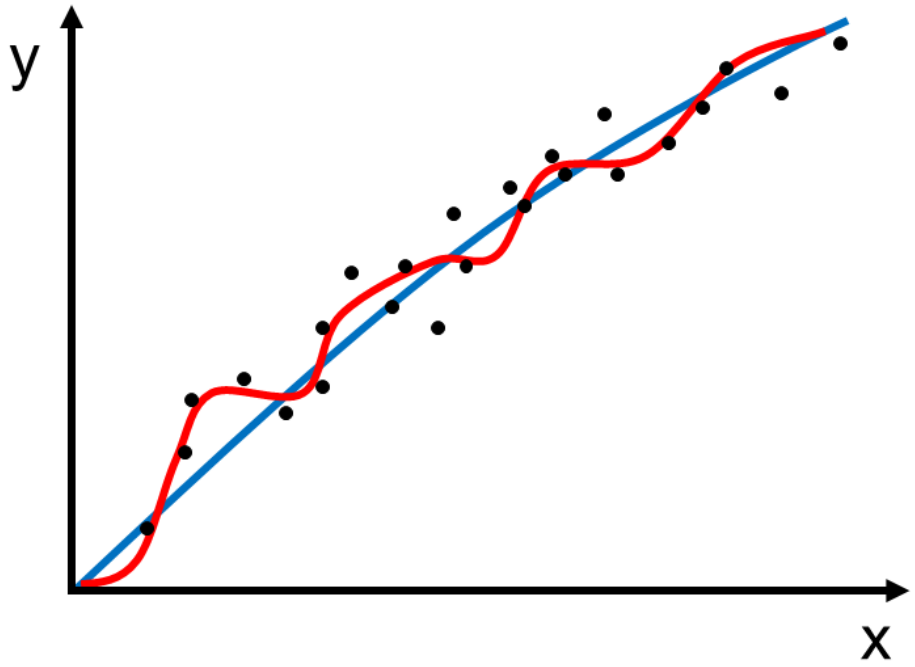


Figure 17 – Visualization of a fitted (blue) and an overfitted (red) regression model for the training data.

Source: Author.

One strategy to avoid overfitting of the regression model is called early-stopping of the training (CARUANA; LAWRENCE; GILES, 2001). It is worth mention that for an unbiased evaluation of the Neural Network in a deployment scenario, the testing dataset is not used at any stage of the network development and no adjustments can be made to the network after the testing performance has been evaluated.

Since an overfitted model performs better than a model with fewer epochs for the training data, but performs worse for a new set of testing data, a part of the training dataset is separated for blind validation at each training epoch, as represented in Figure 18.

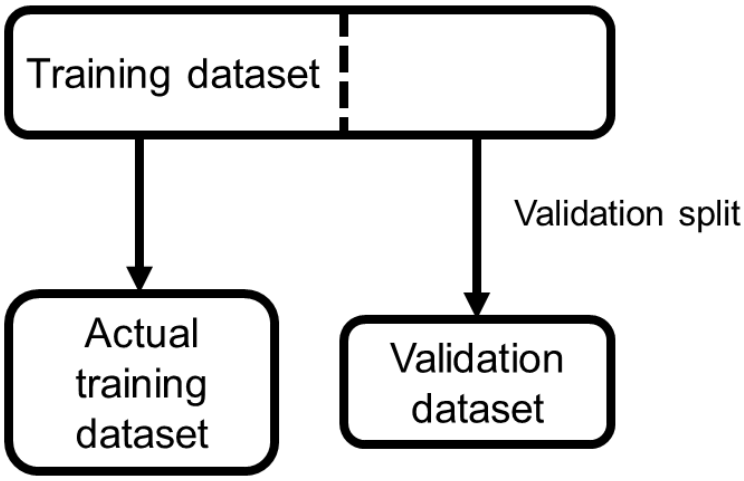


Figure 18 – Separation of a part of the training dataset for validation.
Source: Author.

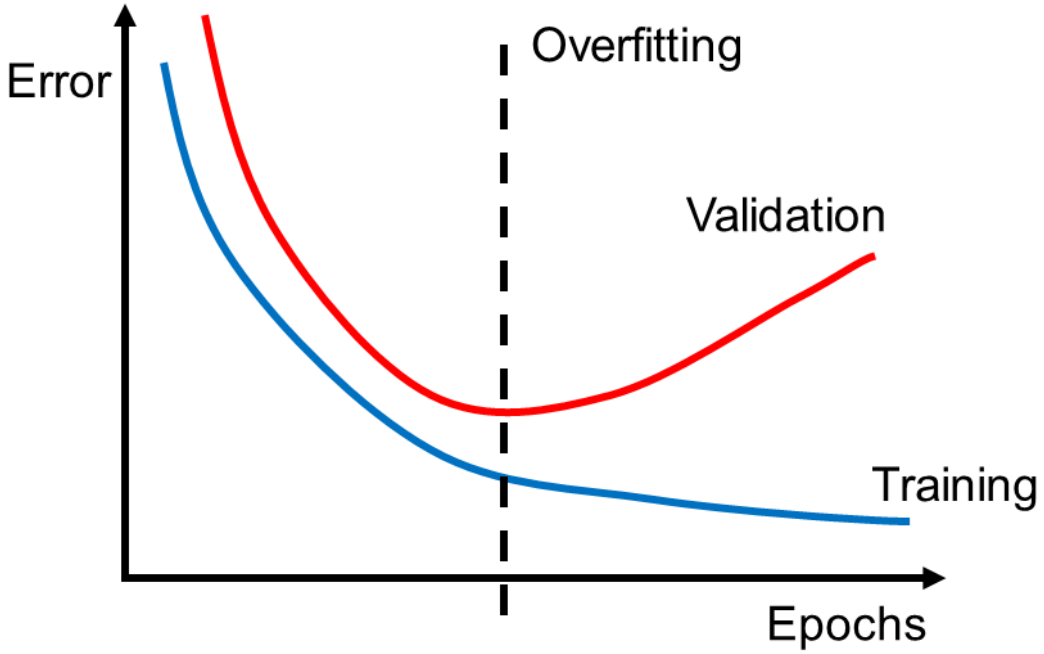


Figure 19 – Comparison between the training (blue) and validation (red) errors to determine the early-stopping of the training.

Source: Author.

The fraction of the validation split is determined empirically and reduces the size of the training dataset because the validation samples are not used for computing the error gradient function. After each training epoch, the performance is evaluated on basis of the validation samples to determine the early-stopping of the training.

Figure 19 presents the decay of the training and the validation errors of a hypothetical Neural Network undergoing training. Both training and validation errors decay at the beginning of the training phase, since the regression model is being developed from the training dataset. Exhaustive training leads to the decaying of the training error. However, overfitting can be detected as the performance error for the blind validation samples increases after an inflexion point. In the early-stopping strategy, the epoch in which the validation error starts increasing is when the training stops.

In the proposed methodology of this research, the dataset is generated from a large number of traffic simulations. In particular, the performed experiments were conducted with data obtained from between 2000 and 10000 simulations, a number

that was determined empirically and feasible with the available computing power in the laboratory.

However, a limited number of dataset samples can be a problem in the training of Neural Networks, and a strategy called cross-validation can be used to reduce the effects of sampling variation and deliver performance metrics that are more unbiased across the available dataset, in comparison to the simple training-testing split from Figure 13. As an example, from the literature, Singh and Panda (2011) used a 10-fold cross-validation technique to minimize training bias in their development of a Neural Network. Figure 20 presents a simplification of their strategy with 4 folds.

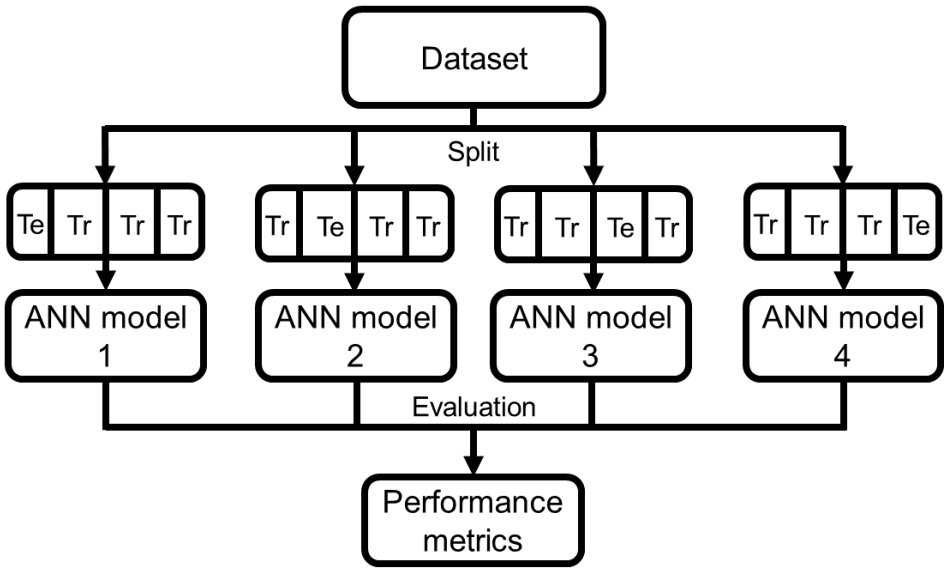


Figure 20 – Evaluation with 4-fold cross-validation of training (Tr) and testing (Te) dataset sections.

Source: Author.

In the example of the 4-fold cross-validation strategy, the dataset is divided into 4 equal parts and, for each part, a different permutation of the training-testing splits is used in the training of a Neural Network model. Finally, the combination of the performance metrics of the 4 individual networks represents statistically the performance metrics of a hypothetical Neural Network that is trained using the entire available dataset. The cross-validation strategy was not used in the experiments

because there were no constraints for obtaining more simulation samples if necessary.

5.5. Neural Network prototyping software

Multiple Back-Propagation (LOPES; RIBEIRO, 2001, 2003, 2009, 2010, 2011) is an alternative software worth of mention for fast and simple prototyping, as well as teaching and visualization because of its simple interface shown in Figure 21. It was used in the early stages of this research for quick prototyping of feedforward Neural Networks before switching to the TensorFlow environment due to limited settings for training the Neural Networks. There is no choice of optimization method and the choices for stopping criteria of the training are either upon reaching a fixed number of training epochs, or by verifying that the training error has reached a determined threshold.

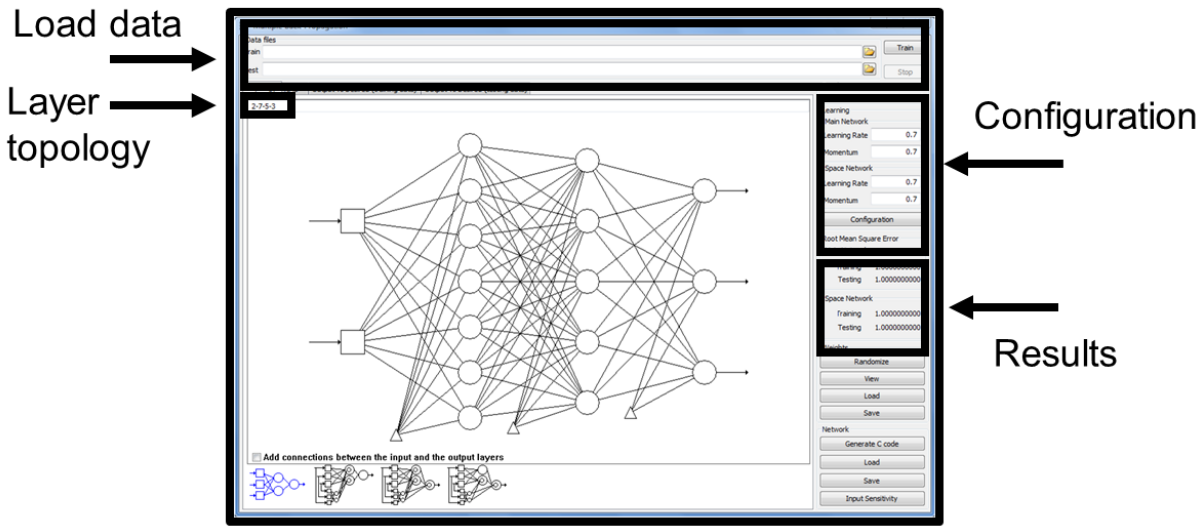


Figure 21 – Interface of the Multiple Back-Propagation software.
Source: Author.

Switching to the TensorFlow and ultimately Keras environment was necessary because of flexibility in the experimentation with Neural Networks. One of the reasons for using Keras in Python scripts is the ability to vary the sizes of the Neural

Network layers in an automated way. Once the dataset is prepared for training, a collection of networks with different depth and width configurations is trained and evaluated. As shown in Figure 22, a network is built by organizing the neuron nodes in a topology.

This research utilizes feedforward networks and the layer configuration refers to the hidden layers, since the input and output layers are determined by the number of variables in the dataset. Therefore, the notation used in the results is that, for example, a 50-50-50 topology represents a Neural Network with three layers, each containing 50 neurons in width.

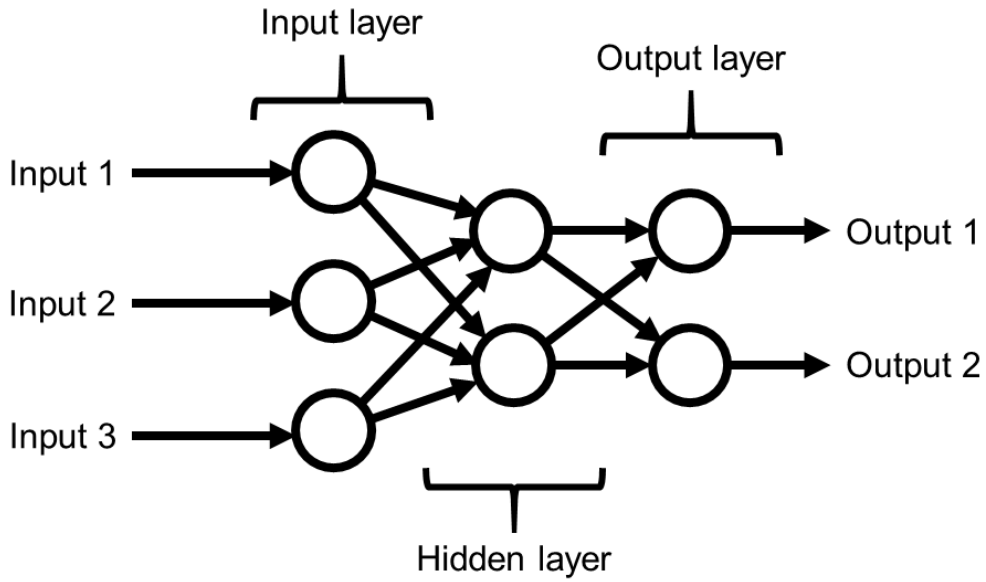


Figure 22 – Layers of a feedforward Neural Network.
Source: Author.

A second reason for using Keras is that the strategy of early-stopping the training is already implemented, and also are various commonly used activation functions and optimizer algorithms. Variation of these settings is done to verify the consistency of the results that validate the proposed methodology. Despite the wide range of activation functions, eleven at the time of writing, the developed Neural Networks were kept with the default sigmoid function, one of the non-linear

continuous activation functions that are adequate for regression, as recommended in the Keras documentation.

However, the Keras library includes the implementation of various optimizer algorithms for the network training and they had effect in the training times and number of epochs before the early-stopping trigger. From those recommended in the documentation for use in regression problems, the six tested optimizers are listed below; all kept with their default internal settings.

- Stochastic Gradient Descent (SGD), as discussed by Ruder (2016).
- Adagrad (DUCHI; HAZAN; SINGER, 2011).
- Adadelta (ZEILER, 2012).
- Adam (KINGMA; BA, 2014).
- Adamax (KINGMA; BA, 2014).
- Nadam (SUTSKEVER et al, 2013).

Additionally, to improve the performance of the optimizer algorithms, the dataset samples can be normalized before the network training. The reason for normalization of the input samples is that different input variables can vary in numerical range (e.g. vehicle counts range from zero to thousands of vehicles/hour, while average speeds have a smaller range from zero to highway speeds of around 100 km/h at maximum). Therefore, some direction axis of the error function surface can be stretched with orders of magnitude in comparison to others, resulting in more computationally expensive iterations because of the numerical disparity.

Normalization of the inputs (subtraction of the variable’s mean and division by its standard deviation) is therefore a step that reduces the computational complexity of the iterations. On the other hand, outputs with larger ranges (e.g. vehicle volume inputs ranging from zero to thousands of vehicles/hour, while route decision variables are fractions between zero and one) can bias the training against fitting the outputs with smaller ranges. Consequently, normalization of the output samples is also desirable in order to keep all the outputs equally important in the modelling of the calibration function.

Once a Neural Network has been trained and is deployed, new inputs are normalized before being processed and the outputs are later denormalized for use.

5.6. Result evaluation

Training the MLP is an iterative optimization problem of searching for the minimum of the error function. The selected error function is customizable and computed between the model outputs at a given iteration and the desired outputs listed in the training dataset. There are different training strategies that range from one training dataset sample being computed by iteration, to the division of the training samples in batches for group computation, and finally the use of the entire training dataset to compute a more generalized version of the error gradient to update the connection weights. In Keras, the default batch size is 32 and this value was kept throughout the experiments.

According to the survey of Hollander and Liu (2008), the literature has a variety of error functions that are used to evaluate the calibration and convergence of an ANN model. For the specific application of traffic microsimulations, they argue that the stochastic nature of traffic makes appropriate the use of quadratic errors, mean quadratic errors and root mean quadratic errors, with some authors using them interchangeably. The adequate error functions are presented in Table 1, from which the Mean Squared error was selected to be used in the experiments, since it was one the default alternatives in Keras.

Table 1 – Adequate error functions for training and evaluation.

Name	Formula
Squared error	$\sum_{i=0}^{N}(x_i - y_i)^2$
Mean squared error	$\frac{1}{N}\sum_{i=0}^{N}(x_i - y_i)^2$
Root mean squared error	$\sqrt{\frac{1}{N}\sum_{i=0}^{N}(x_i - y_i)^2}$
Root mean squared normalized error	$\sqrt{\frac{1}{N}\sum_{i=0}^{N}\frac{(x_i - y_i)^2}{y_i}}$

Source: Author.

However, it is proposed in the methodology that the user (e.g. a traffic engineer) have information to evaluate the trained Neural Network for each individual calibration variable. Furthermore, the user can verify the variables that were modeled with satisfactory error and use them, while ignoring the suggestion from the Neural Network for the variables that were modeled poorly.

The evaluation of performance for each variable is done with the testing dataset. The test inputs are processed by the Neural Network, and the outputs are compared with the desired outputs from the dataset. Additionally, the selected metric to measure performance in a normalized way is the Pearson correlation between the network's predictions and the desired test outputs. The correlation functions as a score of the quality of the calibration for each variable and it is calculated with Equation 1, where x and y are the predicted and desired values of the outputs.

Equation 1 – Pearson correlation formula.

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}}, -1 \leq r_{xy} \leq 1$$

Variables with negative and small correlation values should be ignored by the user, as those represent that the network's predictions do not match the desired values with accuracy, and therefore the accuracy of the network upon deployment is questionable. By contrast, variables with correlation values close to 1 will be possibly calibrated with accuracy by the network upon deployment.

Graphically, a plot of the x and y variables on the XY plane indicates a weak correlation when the points are randomly distributed, as opposed to a strong correlation when the points are close to the x=y line, as presented in Figure 23(a) and (b).

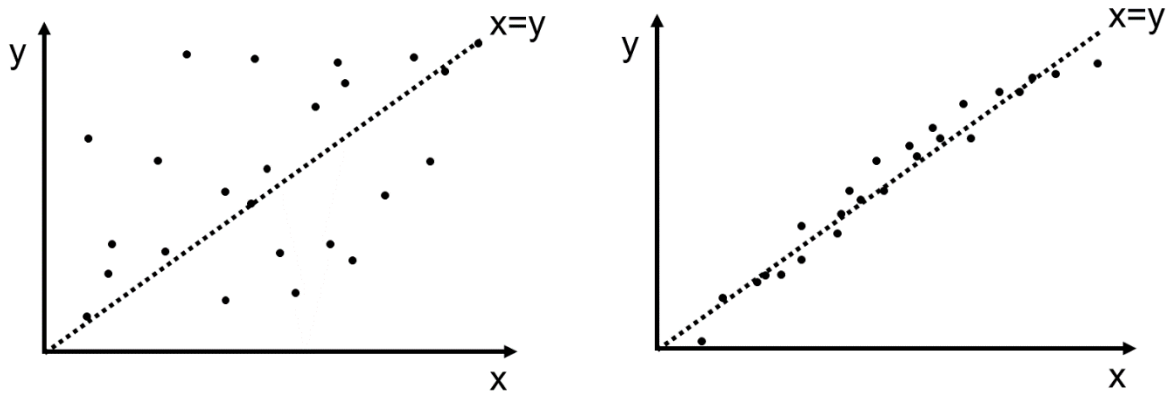


Figure 23 – Example of x and y variables with weak (a) and strong (b) correlations.

Source: Author.

6. Experimental results

6.1. Overview of the experiments

Over twenty experiments were progressively conducted in this research. This project involved learning phases with the traffic simulator, the automation steps and the training of the Neural Networks, and the history of experiments is incremental with iterations of fine-tuning and correction of errors.

Therefore, this dissertation presents four major experiments that were conducted to validate the methodology proposed to automatically calibrate traffic microsimulations. The traffic simulations were run in the PTV Vissim software and the Neural Network models were developed in the Python and Keras environment.

The four experiments were performed with two road networks of interest due to their complexity and frequent traffic congestions. The road network used in Experiment 1 is shown in Figure 24, the access roundabouts between the avenues Radial Leste-Oeste and 23 de Maio in the city center of São Paulo, Brazil. In this network, Neural Networks were used to calibrate vehicle volumes entering the edges of the map and the proportion of cars that follow each direction in every road fork. All experiments use the equivalent vehicle model to compute volumes, i.e. the simulation only presents car entities, under the assumption that motorcycles are implicitly represented as equivalent to less than one car, and buses and trucks are equivalent to more than one car (MARTE et al., 2017b).

Experiments 2, 3 and 4 were performed on the road network shown in Figure 25, a return access on the Raposo Tavares highway in São Paulo, Brazil. In this network, Experiment 2 is similar to Experiment 1 and calibrates the vehicle volumes entering the edges of the map and their routing decisions.

The road network from Figure 25 is the same that was calibrated by Klapper et al. (2017) in their work. By using their report as reference, the vehicle volumes and routings were kept at fixed values and Experiments 3 and 4 use Neural Networks to calibrate the driving behavior parameters, respectively from the Wiedemann 74 (1974) and Wiedemann 99 car-following models (abbreviated to W74 and W99), which are the options available in the PTV Vissim simulator.

Figure 24 – Access roundabouts from Avenida Radial Leste-Oeste to Avenida 23 de Maio (red avenues).

Source: OpenStreetMap (2019).

Figure 25 – Return access (yellow) at km 23 on the Raposo Tavares highway (red).

Source: OpenStreetMap (2019).

6.2. Running multiple simulations

The first part of the proposed methodology is the creation of the dataset by running a large number of traffic simulations, with different input parameters in order to map the transfer function of the simulator. Table 2 presents an example of input table that can be automatically read by a Python script and loaded into PTV Vissim.

The tables with the input data can be created with either Microsoft Excel or the Pandas library from Python and saved into a CSV file. Each variable should follow a statistical distribution that suits the application. The ranges of values used for each experiment is further explained in their setups.

Table 2 – Examples of inputs for multiple simulations.

Simulation*	Vehicle volume 1 (vehs/h)	Vehicle volume 2 (vehs/h)	Vehicle volume 3 (vehs/h)	Route decision 1a	Route decision 1b**
1	629	2003	304	0.15	0.85
2	321	3544	221	0.91	0.09
3	404	120	50	0.35	0.65
4	20	2	1020	0.56	0.44

*** The simulation index is removed from the table before the Python script is run to read the data from the table.**

**** For simplicity, route decisions are modeled as always pairs and the sum of the routing options (a) and (b) always equals to 1. Therefore, the Neural Network only has to estimate the value of option (a) for each route decision.**

Source: Author.

A Python script example to run multiple simulations is presented in Figure 26. The script loop reads and loads the data from each row individually and then launches a simulation run. The loop counter, the command to load the data and the launch of the simulation are highlighted in red. Minor changes are necessary for each different experiment and the full scripts are in the Appendices.

After all the simulation runs, PTV Vissim generates a table with the outputs for each simulation listed in the same format as in Table 2. Finally, the dataset for training the Neural Network is created from the use of Vissim input parameters as the Neural Network outputs, and the Vissim output results as the Neural Network inputs. The script that concatenates the tables is also in the Appendices. This script also removes the rows of data that are incomplete and appear, for example, when not a single car has been able to pass a travel-time segment in very congested traffic simulations. In all the experiments, these scenarios were observed in less than 3% of the simulations.

```
# import format: 3 vehicle volumes + 14 static route RelFlows
# Vol1, Vol2, ..., Rout1_opt1, Rout1_opt2, ..., Rout2_opt1,...
import csv

with open('inputs.csv', 'rb') as csvfile:
    all_flows = Vissim.Net.VehicleInputs.GetAll()
    all_routes = Vissim.Net.VehicleRoutingDecisionsStatic.GetAll()
    myfile = csv.reader(csvfile, delimiter=';')

    # first row is the variable names
    input_variable_names = myfile.next()

    for k in range(4000):
        this_line = myfile.next()
        index = 0
        for t in range(len(all_flows)):
            all_flows[t].SetAttValue("Volume(1)",
this_line[index])
            index+=1
        for m in range(len(all_routes)):
            options = all_routes[m].VehRoutSta
            for n in range(len(options)):
                options[n].SetAttValue("RelFlow(1)",
this_line[index])
                index+=1

        Vissim.Simulation.RunContinuous()
```

Figure 26 – Example script for automated 4000 simulation runs.

Source: Author.

6.3. First experiment

The first experiment uses the road network presented in Figure 27 and Figure 28. The desired inputs for calibration are the 5 vehicle volumes that enter the edges of the map, and the ratios of the 5 route decisions that are part of the road network.

The output performance metrics from PTV Vissim that are used by a Neural Network to calibrate simulations on this map are 13 travel times with also their vehicle counts.

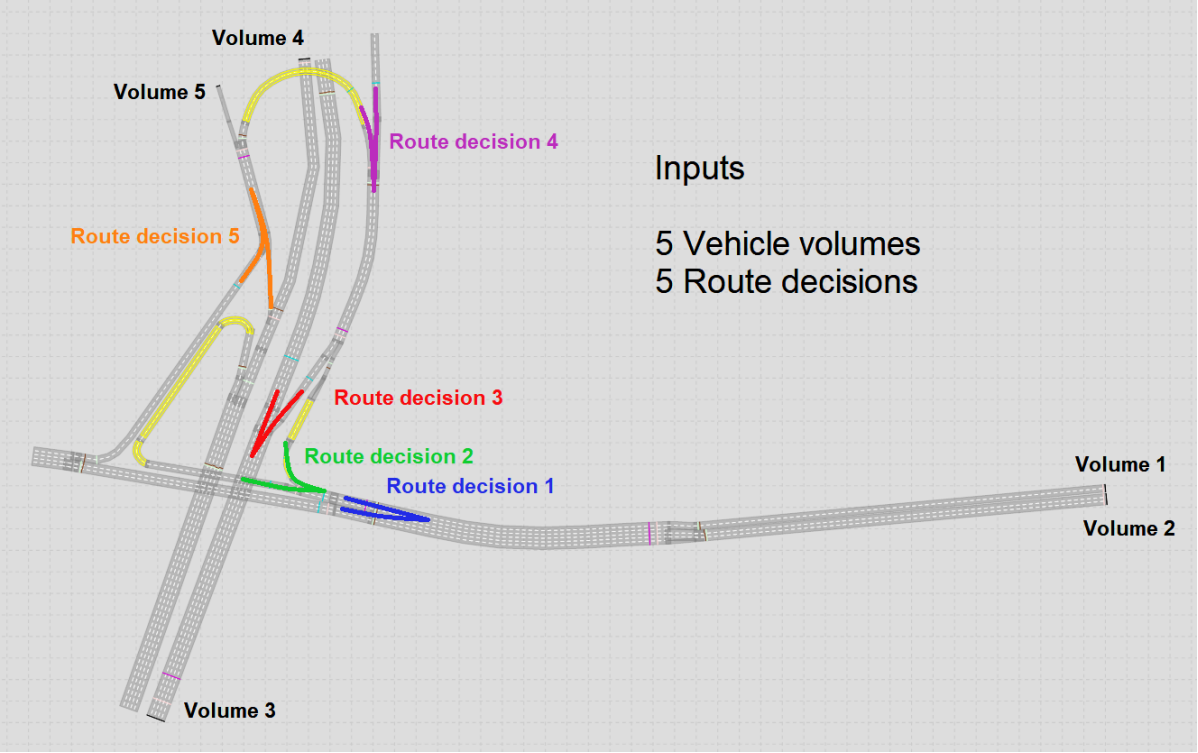


Figure 27 – Inputs for calibration of the road network in the first experiment.
Source: Author.

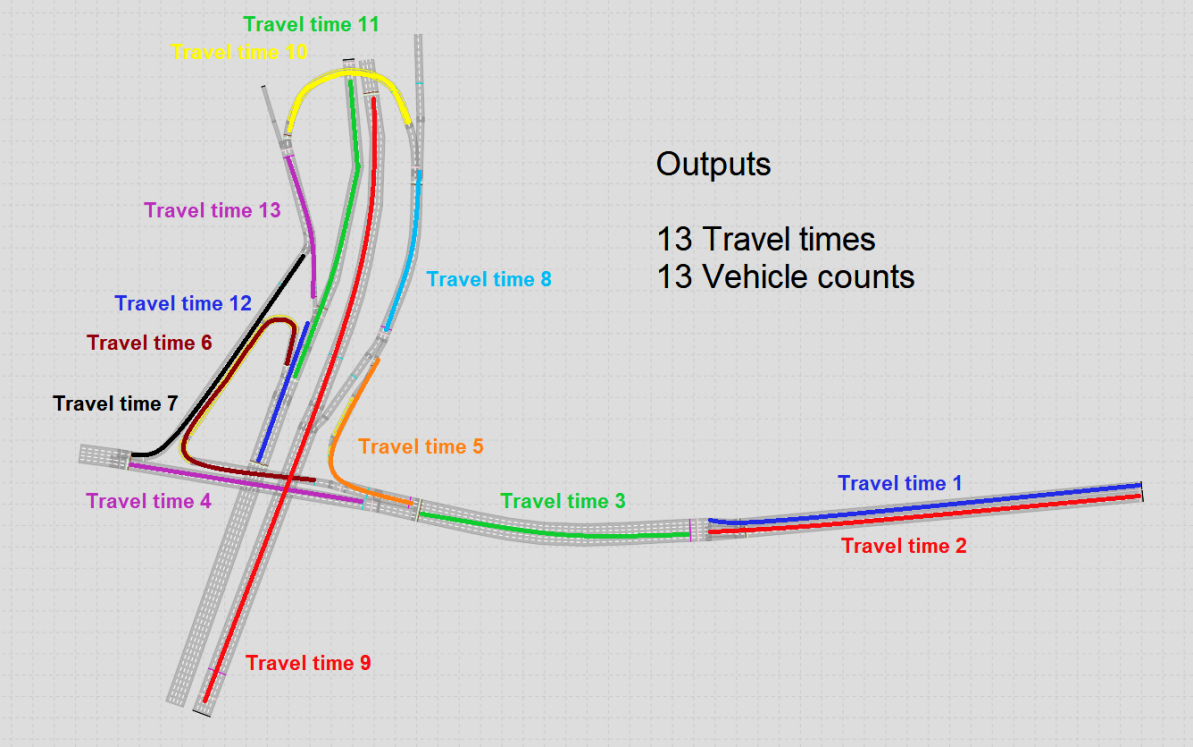


Figure 28 – Output results used as inputs of the Neural Network in the first experiment.
Source: Author.

The preparation of the simulation runs followed the setup listed below.

- **Number of simulations:** 4000. The number was chosen empirically. The ratio between training/testing is 80/20, and the validation split within the training dataset is 20%.
- **Inputs**
  - 5 vehicle volumes, following a uniform distribution between 0 and 1800 vehicles per hour per lane, the maximum theoretical capacity according to Dhamaniya and Chandra (2014), and Spack (2011). The objective is to allow a variety of combinations of free-flow and congested traffics (BETHONICO, 2016).
  - 5 Route decisions, following a uniform distribution between zero and one.
- **Simulation duration:** 3600 seconds, chosen empirically, ignoring the initial 600 seconds for transient behavior in the road network, which starts empty (MARTE et al., 2017a).

- **Desired speeds:** Set to the regulatory speed limits of the road segments.
- **Outputs:** 13 travel times for the segments shown in Figure 28. The travel time entities also count the vehicles that passed through the segment for the calculation; hence there are 13 additional vehicle counts.
- **Driving behaviors:** All parameters for the Wiedemann 74 car-following model, lateral behavior and lane-change behavior were kept in their default values (FRANSSON, 2018; MIRANDA et al., 2018).

After the simulations and the creation of the training dataset, a collection of Neural Networks was trained and tested, using the correlation between the desired values for calibrating the simulation and the Neural Network's estimates as the performance metric. For all trained networks in all the experiments, the training/testing dataset split was empirically determined as 80/20 from the common practice in the reviewed literature. Within the training dataset, 80/20 was also the ratio of the validation split.

Due to the number of input and output variables, the topology of 2 hidden layers with 50 neurons each (the notation is a 50-50 configuration) was fixed, as an initial attempt, for experimentation with 6 optimizer algorithms recommended for regression in the Keras documentation. Figure 29 shows how many epochs were necessary for training the Neural Networks with each optimizer, before the early-stopping callback.

It can be observed that the Stochastic Gradient Descent (SGD) optimizer required more epochs in two orders of magnitude, being the possible reason that this optimizer uses a fixed step size for navigation on the error function surface. By contrast, the other 5 optimizer alternatives implement adaptive-sized steps and required around 100 epochs before possibly overfitting the models.

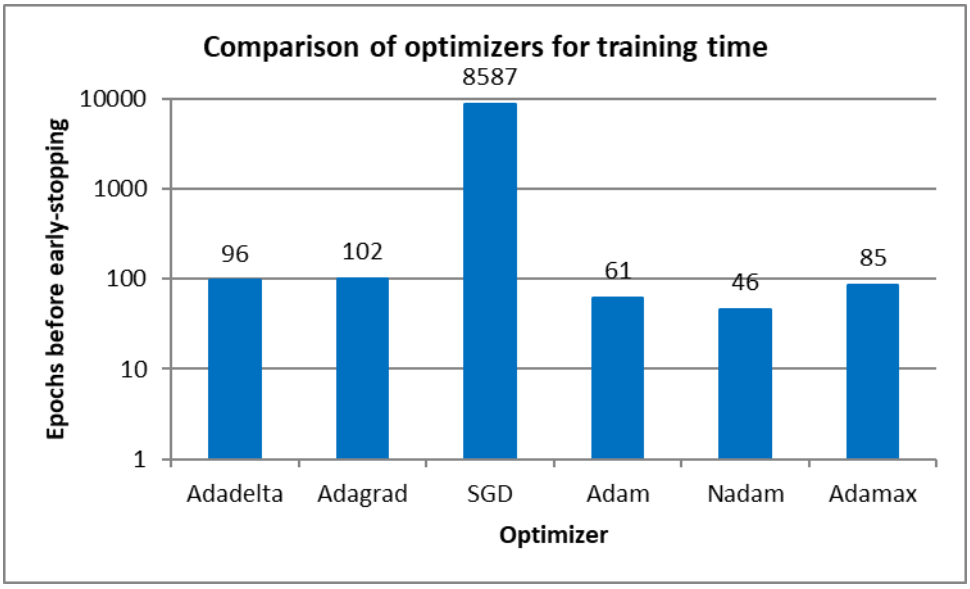


Figure 29 – Comparison of training epochs in the first experiment.

Source: Author.

Figure 30 shows the comparison between the average of the correlation metrics for the 6 tested optimizers. The variation of performance is less than 1% and therefore it has been concluded that the optimizers affect the training time of the Neural Networks, but not their performance in this particular experiment.

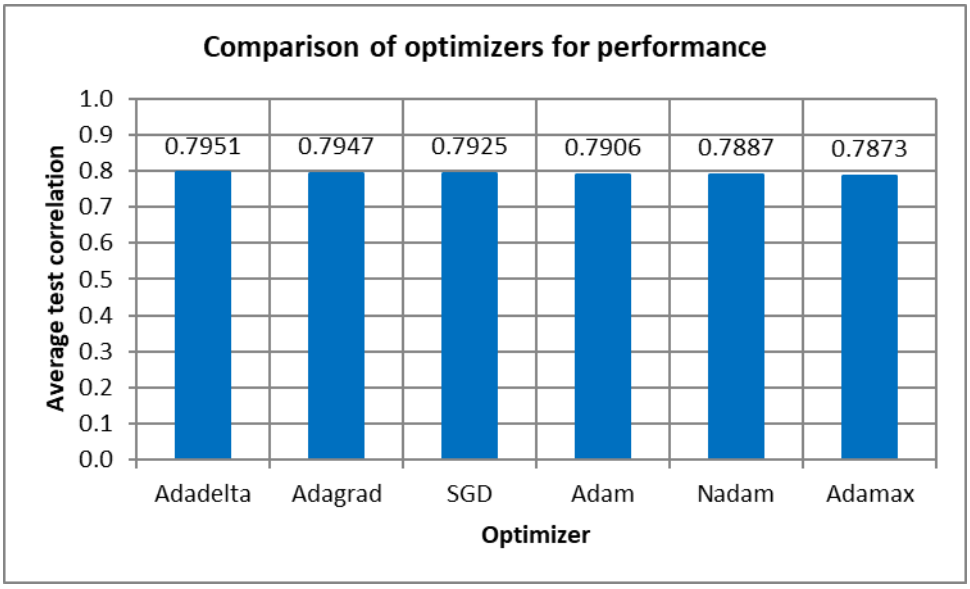


Figure 30 – Comparison of correlations for each optimizer in the first experiment.

Source: Author.

The next step of experimentation is the effect of the hidden-layer topology in the performance of the model. To isolate the effects of the topology, the optimizer was kept with the Adadelta alternative, the best-performing for the previous step, and the activation functions of the neurons were kept at the default sigmoid option for simplicity. Figure 31 shows a comparison between the average correlation metrics of 21 trained Neural Networks, with topologies that are the combinations of 1, 2 and 3 layers in depth, with 2, 3, 5, 10, 15, 20 and 50 neurons in width.

The overall best performance is that of networks with 1 and 2 hidden layers, as opposed to 3 layers. One possible explanation is the vanishing gradient problem, as explained by Glorot and Bengio (2010). The transfer function representing the Neural Network is a concatenation of the transfer functions of its layers. Since the recommended activation functions for non-linear regression are normalized or limited between -1 and 1 (e.g. sigmoid and tahn), the gradient error function vanishes (in other words, the error function surface flattens) if it is computed over networks with many concatenated layers.

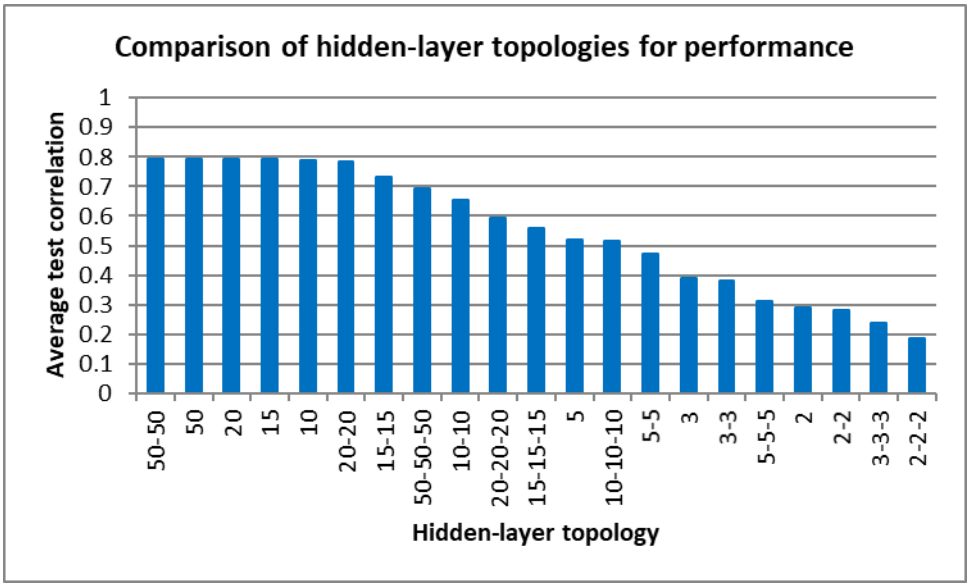


Figure 31 – Comparison of correlations for each topology in the first experiment.

Source: Author.

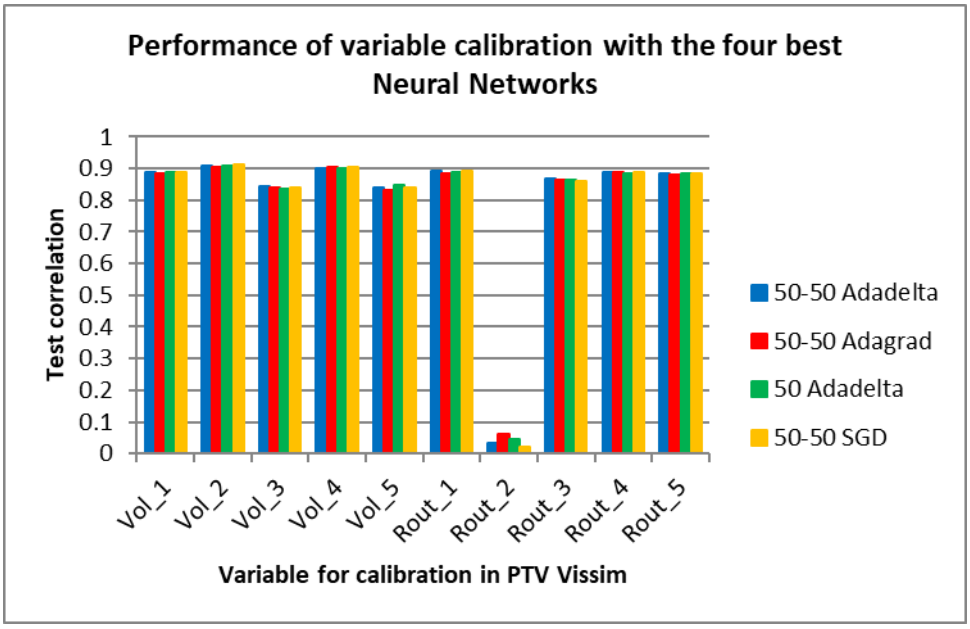


Figure 32 – Comparison of correlations of each variable in the first experiment.
Source: Author.

Figure 32 compares the performance of calibration for each variable in PTV Vissim, in the case of the four best-performing Neural Networks. The choice of network in these four cases has small impact on the performance and the measured correlation was between 0.8 and 0.9 (which are desirable values) for all variables except for Rout_2. The latter has a correlation close to zero and the conclusion that the Neural Networks estimates random values for its calibration.

The results for the variables where high correlation was measured are positive to validate the proposed methodology. However, the non-existent correlation measured for Rout_2 should be examined. From the map in Figure 27, route decision 2 is a corner access between the avenues with only one lane. Additionally, cars can take both alternatives on route decision 2 to the same destination on Avenida 23 de Maio.

Punzo, Montanino and Ciuffo (2014) discuss in their work that a subset of variables may be sufficient for calibration, and that a sensitivity analysis can be used to determine what variables to ignore. It is possible that the Neural Network's learning process is unable to capture the relationship between the Rout_2 variable

and the output metrics due to complexity limitations, but it is also possible that the contribution of this variable to the metrics is too small to be considered for calibration.

At the end of the Neural Network development, the correlations of the individual variables are delivered. In cases such as that of this first experiment, the user can verify the metrics and choose or not to use the network's suggestions as the values of calibration. Variables such as Rout_2 can then be either calibrated manually or set at a default value and ignored if the overall result satisfies the requirements of the traffic engineer, for example.

Since the training dataset has been normalized, all the inputs and outputs from the Neural Network are equally balanced. Even though there are measured vehicle counts in the edge segments of the road network, and the Neural Network attempts to calibrate the vehicle inputs to those segments, the regression model also takes into consideration the error of the inner measurements of the map.

Except for Rout_2, the volume variables did not show significantly higher correlations than the route variables because the Neural Network did not exactly match the vehicle volume inputs to the vehicle counts of the edge segments, for the benefit of reducing also the error of the inner segments.

6.4. Second experiment

The second experiment uses the road network presented in Figure 33, Figure 34 and Figure 35. It follows the format from the first experiment and the desired inputs for calibration are the 3 vehicle volumes on the edges of the map and 7 route-decision ratios.

The output performance metrics from PTV Vissim that are used for calibrating simulations on this map are 12 travel times, 14 vehicle counts, and 14 harmonic average speeds computed on those counters throughout the simulation period.

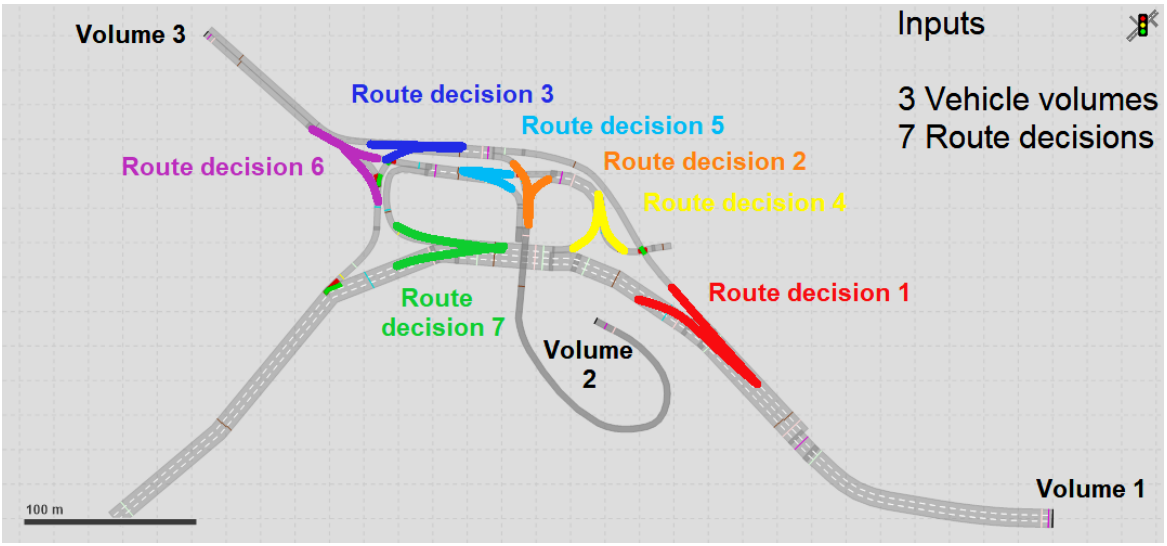


Figure 33 – Inputs for calibration of the road network in the second experiment.

Source: Author.

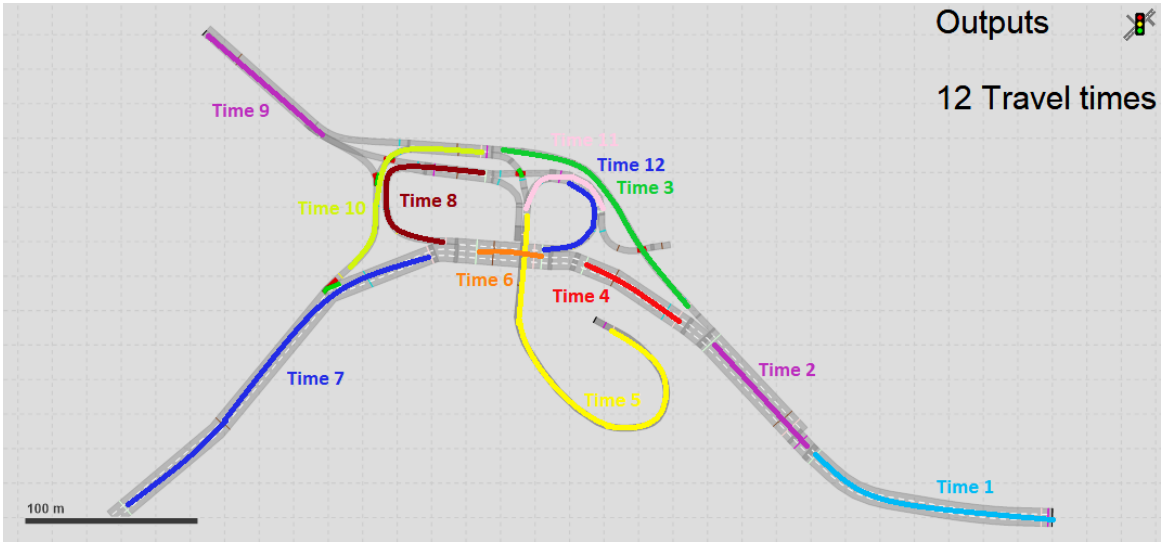


Figure 34 – Output results from the travel time entities, used as inputs of the Neural Network in the second experiment.

Source: Author.

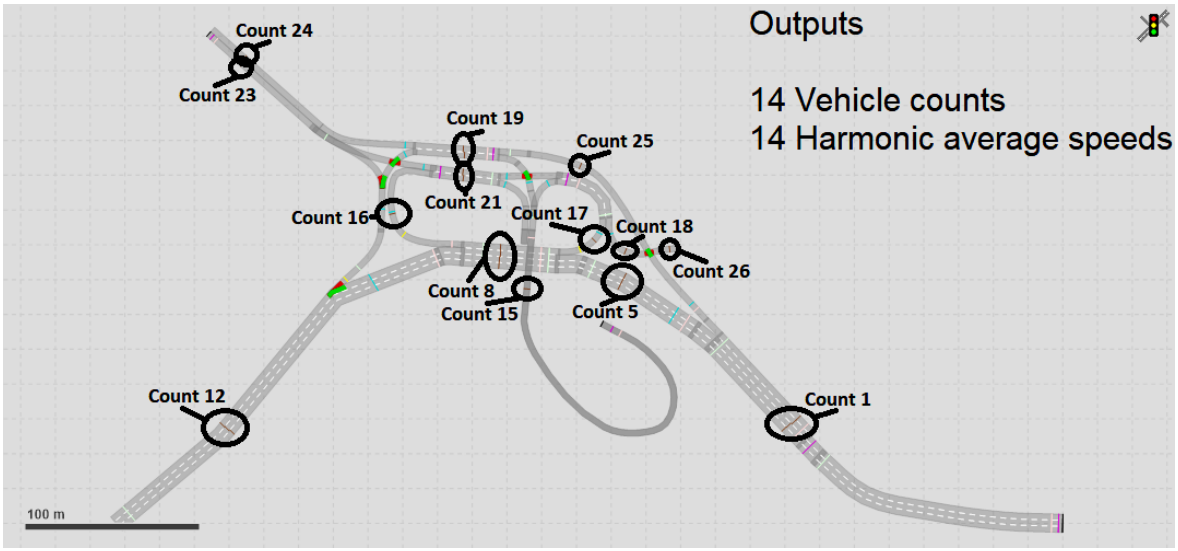


Figure 35 – Output results from the vehicle counter entities, used as inputs of the Neural Network in the second experiment.

Source: Author.

The preparation of the simulation runs followed the setup listed below.

- **Number of simulations:** 4000, chosen empirically. The ratio between training/testing is 80/20, and the validation split within the training dataset is 20%.
- **Inputs**
  - 3 vehicle volumes, following a uniform distribution between 0 and 1800 vehicles per hour per lane (DHAMANIYA; CHANDRA, 2014; SPACK, 2011).
  - 7 Route decisions, following a uniform distribution between zero and one.
- **Simulation duration:** 1800 seconds, chosen empirically, ignoring the initial 300 seconds for transient behavior (MARTE et al., 2017a).
- **Desired speeds:** Set to the regulatory speed limits of the road segments.
- **Outputs:** 12 travel times, 14 vehicle counts and 14 harmonic speed averages.

- **Driving behaviors:** All parameters for the Wiedemann 74 car-following model, lateral behavior and lane-change behavior were kept in their default values (FRANSSON, 2018; MIRANDA et al., 2018).

After the simulations and the creation of the training dataset, the Neural Networks were trained and tested using the same methods from the first experiment of measuring correlations between estimated and desired Neural Network outputs.

Similarly to the first experiment, the topology of 2 hidden layers with 50 neurons each was fixed for experimentation with the 6 recommended optimizer algorithms in Keras. Figure 36 shows the number of epochs necessary to reach a possible overfitting of the Neural Networks. In a similar result from the previous experiment, there was a disparity of order of magnitude between the optimizers.

Figure 37 reproduces the result from the first experiment that the optimizers affect the training time more than the correlation measurements. Following the reference from the previous experiment and due to the small variation in results, the Adadelta optimizer was kept fixed in order to experiment with the hidden layer topologies. The activation function remained the default sigmoid for simplicity.

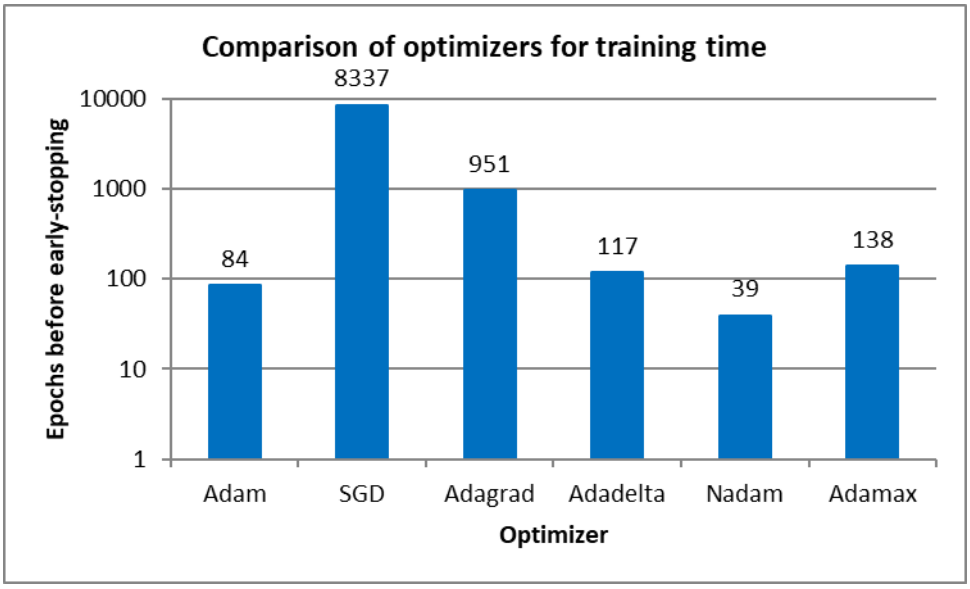


Figure 36 – Comparison of training epochs in the second experiment.

Source: Author.

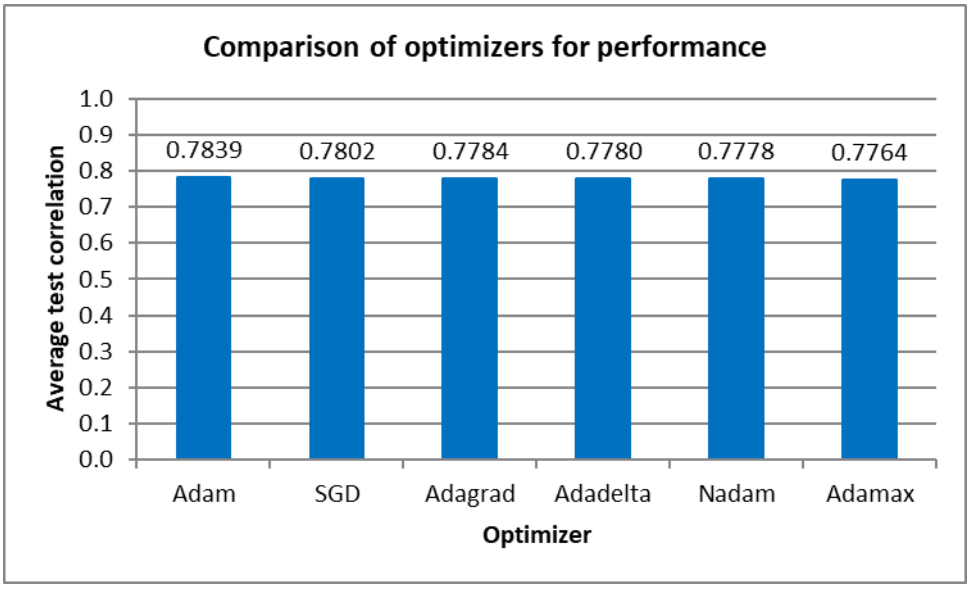


Figure 37 – Comparison of correlations for each optimizer in the second experiment.
Source: Author.

The combinations of depths and layer widths used to vary the hidden-layer topologies were those of 1, 2 and 3 layers with 3, 5, 10, 15, 20 or 50 neurons each. The average correlations for each of these networks are compared in Figure 38. Finally, the measured correlations at individual variables are compared for the overall best-performing networks in Figure 39.

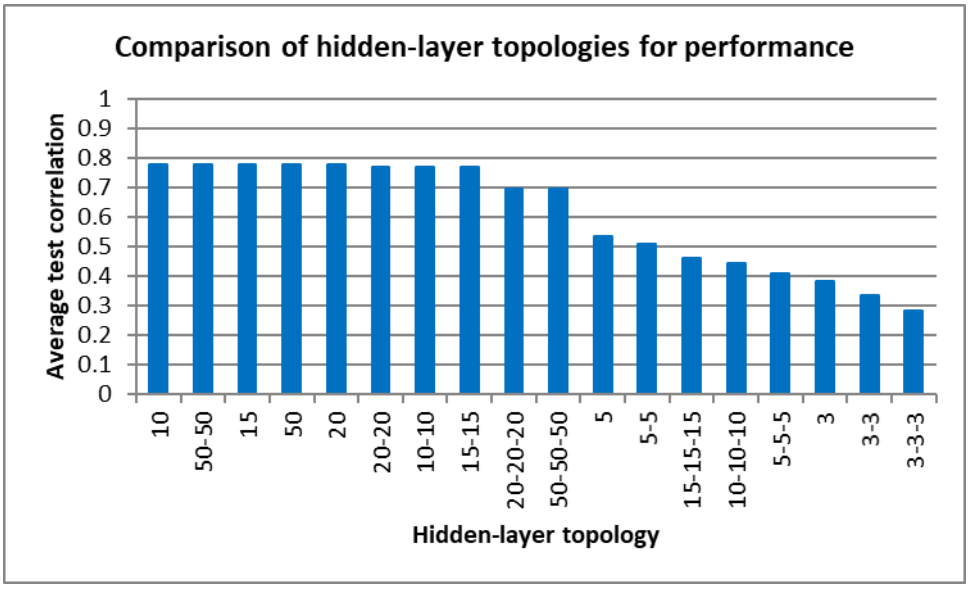


Figure 38 – Comparison of correlations for each topology in the second experiment.
Source: Author.

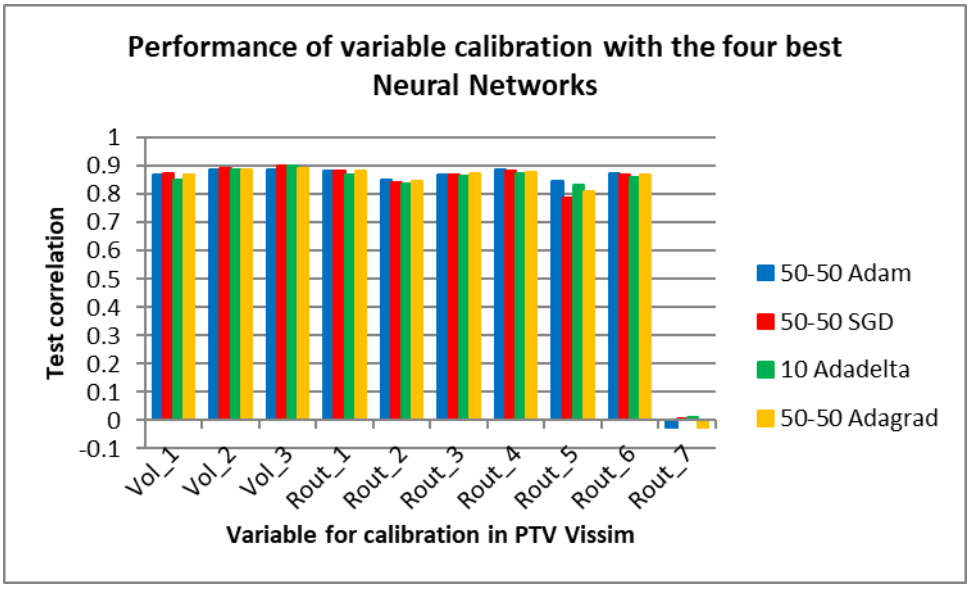


Figure 39 – Comparison of correlations of each variable in the second experiment.
Source: Author.

The best Neural Network in this experiment had the 50-50 configuration with the Adam optimizer. All variables had a correlation between 0.8 and 0.9, except for Rout_7, which is a roundabout corner access. Following the justification from the first experiment, it is possible that the relationships between this variable and the PTV Vissim metrics are too complex for the Neural Networks to absorb during training.

The user has access to the individual metrics of the calibration variables after the Neural Network development. A traffic engineer has the possibility to separately calibrate this variable manually, or to ignore it at a default value if the overall results of the estimated calibration values are adequate to the project's requirements.

6.5. Third experiment

The third experiment used the same road network from the second experiment and has the goal of calibrating another group of variables of interest: the driving behavior parameters. More specifically, the car-following and lane-change models are abstract and used to compute the interactions between the drivers.

Aghabayk et al. (2013), Llanque Ayala (2013), Otkovic, Tommazzi and Sraml (2013), Miller (2009) and Vilarinho (2008) are all authors who conducted research on the calibration of driving behaviors, with focus on the Wiedemann 74 and Wiedemann 99 car-following models. These models are equations to describe the positions and speeds of the vehicles that follow the same lane. Even though these parameters can be accessed and edited in PTV Vissim, their meanings are abstract and often a topic of discussion in the reviewed literature.

Klapper et al. (2017) have calibrated a PTV Vissim simulation of the map from the second experiment with data from the real-world, and the simulation file was available to use in this research. The route decisions were kept fixed at the values from their calibration and the vehicle volumes were kept fixed at 80% of each road's capacity for simplicity. This value was chosen empirically because a low occupancy rate minimizes the interactions between drivers, which would reduce the sensitivity of the simulation output metrics to the variation of the driving parameters.

By contrast, a very congested road network would not allow the vehicles to reach higher speeds and display behaviors related, for example, to overtaking and aggressive braking. The work from Miranda et al. (2018) and Fransson (2018) were used as reference to select 4 variables of interest, related to the Wiedemann 74 car-following model and the lane-change model, which affects the aggressiveness in overtaking cars. The PTV Vissim outputs used for calibration were identical to those of the second experiment: travel times, vehicle counts and harmonic average speeds. The preparation of the simulation runs followed the setup listed below.

- **Number of simulations:** 3000, chosen empirically. The ratio between training/testing is 80/20, and the validation split within the training dataset is 20%.
- **Inputs**
    - W74ax, uniformly distributed between 0.5m and 3m
    - W74bxAdd, uniformly distributed between 0.5m and 4m.
    - W74bxMult, uniformly distributed between 0.5m and 6m.
    - MinHdwy, the variable that corresponds to the minimum headway distance with the vehicle at an adjacent lane, used as a threshold

for changing lanes. This variable was uniformly distributed between 0.5m and 7m.

    - The above ranges for the variables were determined from the works from Miranda et al. (2018), Fransson (2018) and Sukennik and Kautzsch (2018).
- **Simulation duration:** 1800 seconds, chosen empirically, ignoring the initial 300 seconds for transient behavior (MARTE et al., 2017a).
- **Desired speeds:** Set to the regulatory speed limits of the road segments.
- **Outputs:** 12 travel times, 14 vehicle counts and 14 harmonic speed averages.
- **Vehicle volumes:** Fixed at 80% of the capacity of the edge segments.
- **Route decisions:** Fixed at the values from the calibration work from Klapper et al. (2017).

After the simulations and the creation of the training dataset, the Neural Networks were trained and tested using the same methods from the previous experiments. Following the same procedures of experimenting with the 6 Keras optimizers in a 50-50 topology, the results are presented in Figure 40 and Figure 41. The results are consistent with the conclusions from the previous two experiments that the choice of optimizer affects the training time but has a small effect on the correlation results of the Neural Networks.

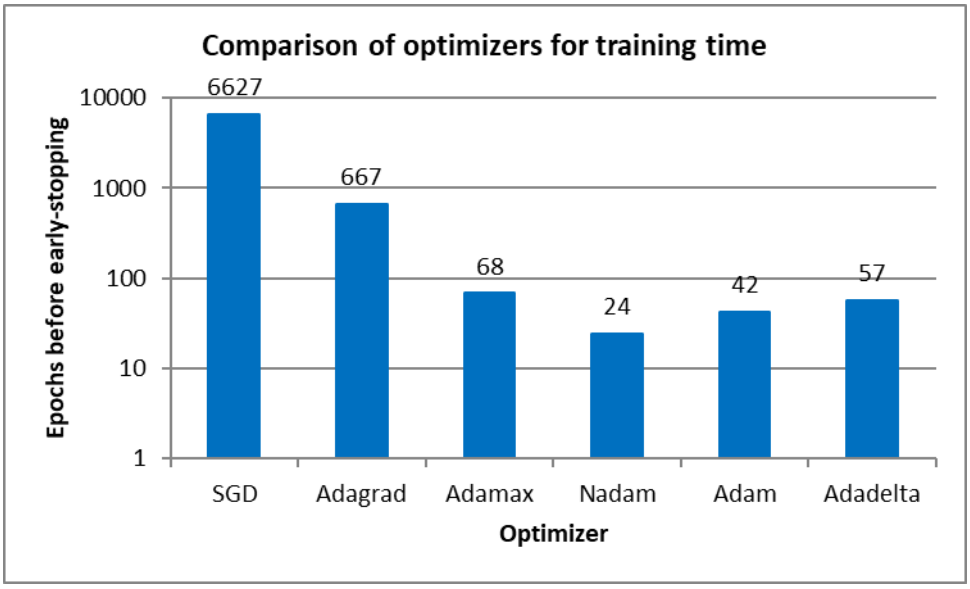


Figure 40 – Comparison of training epochs in the third experiment.

Source: Author.

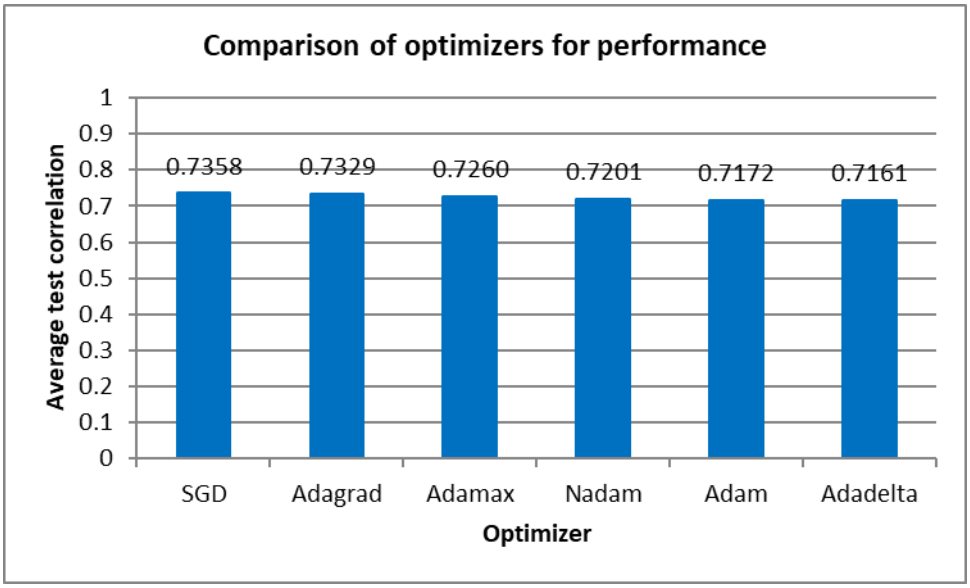


Figure 41 – Comparison of correlations for each optimizer in the third experiment.

Source: Author.

For consistency and simplicity, the Adadelta optimizer and the sigmoid activation function were kept fixed for the experimentation with topologies. The combinations of depths and layer widths used to vary the hidden layers were those of 1, 2 and 3 layers with 1, 2, 3, 5, 10 or 50. The results are presented in Figure 42 and Figure 43.

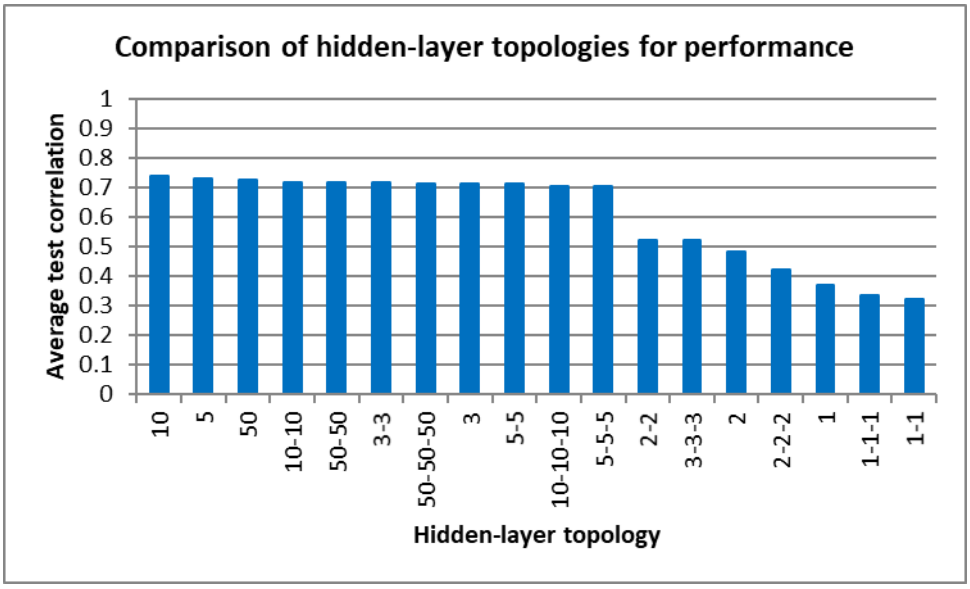


Figure 42 – Comparison of correlations for each topology in the third experiment.

Source: Author.

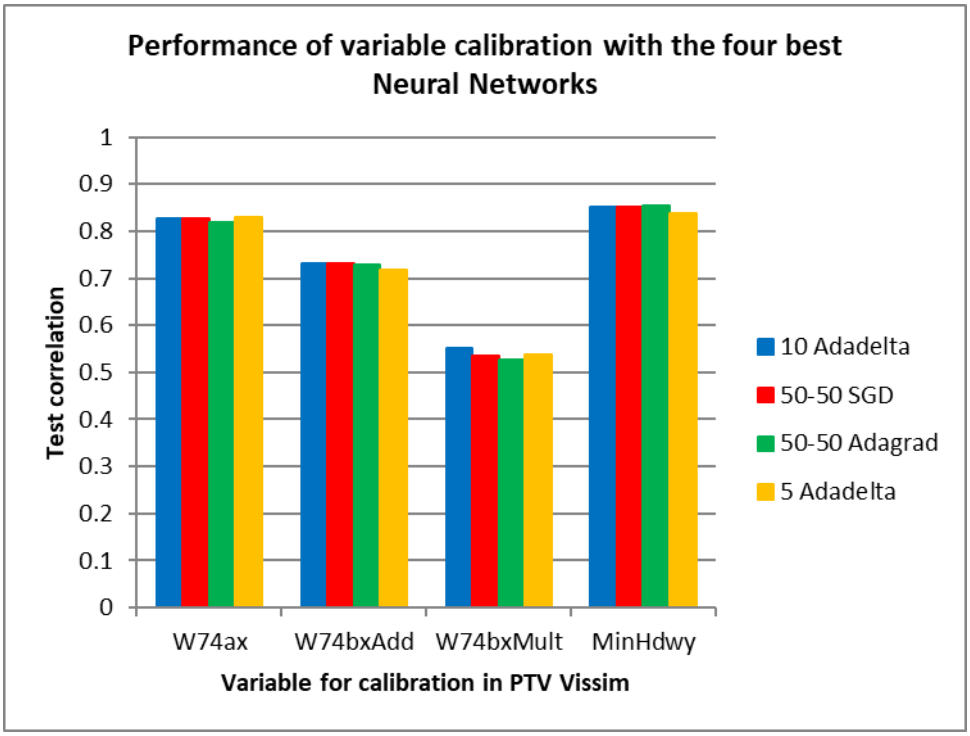


Figure 43 – Comparison of correlations of each variable in the third experiment.

Source: Author.

The number of variables for calibration was smaller in the third experiment than in the previous two, and a smaller Neural Network with a single layer of 10 neurons

was the best-performing for the collection. W74ax and MinHdwy are variables with consistent correlations above 0.8, which is a desirable value.

However, W74bxMult has a weaker correlation below 0.6 and poses a situation for the evaluation of the user. The level of abstraction of the driving behavior models makes it more difficult to explain a poorer performance of calibration than variables with more visible effects on the road network. A traffic engineer that uses the proposed methodology can choose to ignore this variable or use the Neural Network's estimate as a starting point for further fine-tuning.

6.6. Fourth experiment

The fourth experiment is an adaptation of the third experiment with the only change of the car-following model. The lane-change MinHdwy was kept fixed at its default value of 0.5, and the Wiedemann 74 car-following model was replaced by Wiedemann 99, a more complete model that was designed for use in highways. Wiedemann 99 has 10 parameters that relate to acceleration, braking, reaction times and safe distances, among others. Calibration of these parameters is a subject of interest for research and discussion because of their abstract meanings and because their tuning has been suggested as means to model the behavior of autonomous vehicles (SUKENNIK; KAUTZSCH, 2018).

However, the work from Punzo, Montanino and Ciuffo (2014) discusses the effectiveness of attempting to calibrate all parameters at the expense of simplicity, especially if low sensitivity of the traffic simulations is observed to some of the parameters. The preparation of the simulation runs followed the setup listed below.

- **Number of simulations:** 2000, chosen empirically. The ratio between training/testing is 80/20, and the validation split within the training dataset is 20%.
- **Inputs:** Wiedemann 99 has 10 parameters, of which 9 can be automatically edited with scripts in the COM interface (W99cc1 is a drop-down menu with the default value 0.9 and was kept fixed for simplicity).

The ranges of the W99 parameters below were chosen to contain the suggested values in the report from Sukennik and Kautzsch (2018).

- W99cc0: Uniformly distributed between 0.5m and 2m.
- W99cc1: Fixed at 0.9s.
- W99cc2: Uniformly distributed between 0m and 5m.
- W99cc3: Uniformly distributed between -12 and -4.
- W99cc4: Uniformly distributed between -0.4 and 0.
- W99cc5: Uniformly distributed between 0 and 0.4.
- W99cc6: Uniformly distributed between 0 and 12.
- W99cc7: Uniformly distributed between 0m/s² and 0.3 m/s².
- W99cc8: Uniformly distributed between 0 m/s² and 5m/s².
- W99cc9: Uniformly distributed between 0 m/s² and 3m/s².

- **Simulation duration:** 1800 seconds, chosen empirically, ignoring the initial 300 seconds for transient behavior (MARTE et al., 2017a).
- **Desired speeds:** Set to the regulatory speed limits of the road segments.
- **Outputs:** 12 travel times, 14 vehicle counts and 14 harmonic speed averages.
- **Vehicle volumes:** Fixed at 80% of the capacity of the edge segments.
- **Route decisions:** Fixed at the values from the calibration work from Klapper et al. (2017).

Following the same procedures of experimenting with the 6 Keras optimizers in a 50-50 topology as an initial approach, the results are presented in Figure 44 and Figure 45, and are consistent with all three other experiments. However, the average correlations in all cases are below 0.5. Figure 46 compares 24 Neural Networks with the sigmoid activation functions and the Adadelta optimizer, in combinations of 1, 2 and 3 layers with 2, 3, 5, 10, 15, 20, 30 and 50 neurons per layer. In all cases, the average correlations were below 0.5.

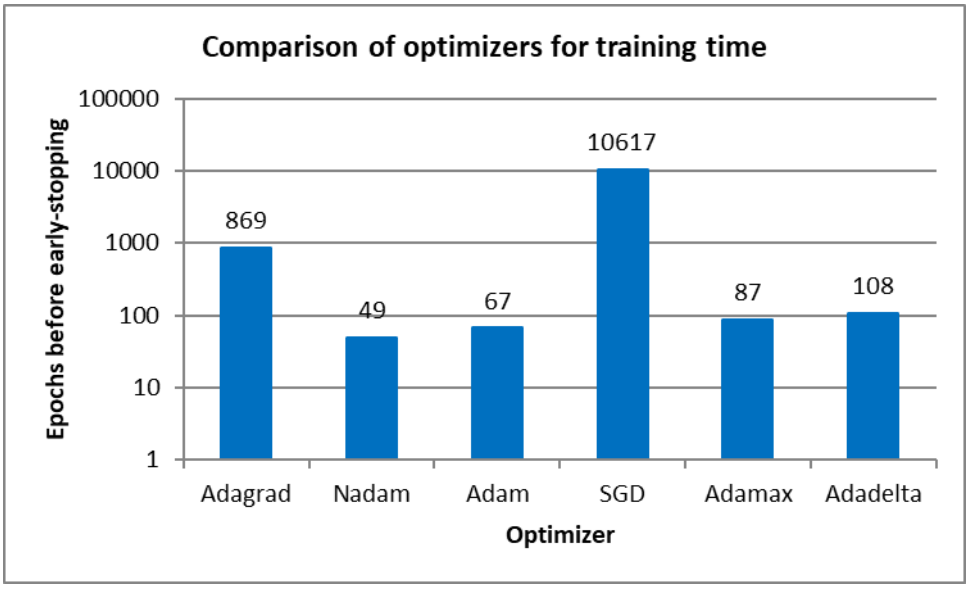


Figure 44 – Comparison of training epochs in the fourth experiment.

Source: Author.

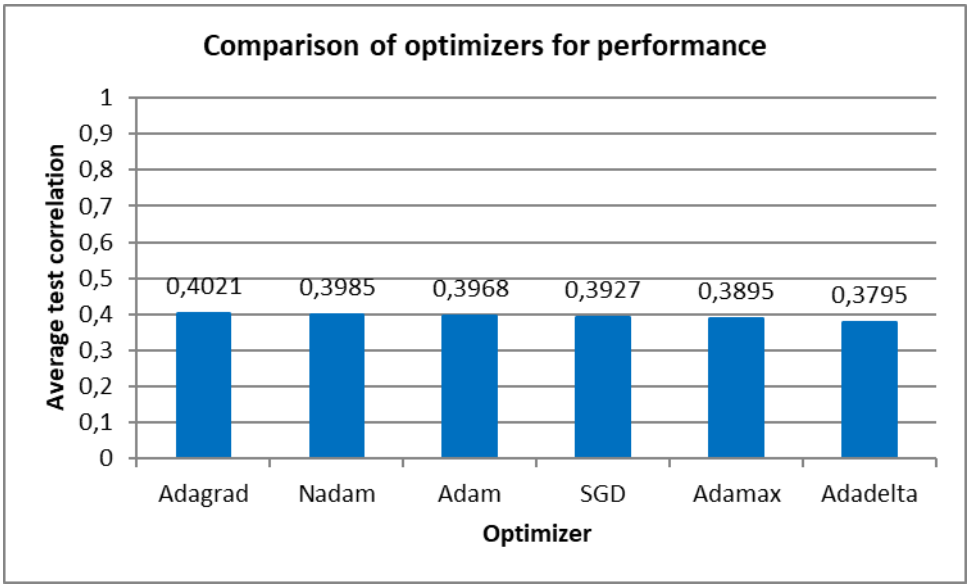


Figure 45 – Comparison of correlations for each optimizer in the fourth experiment.

Source: Author.

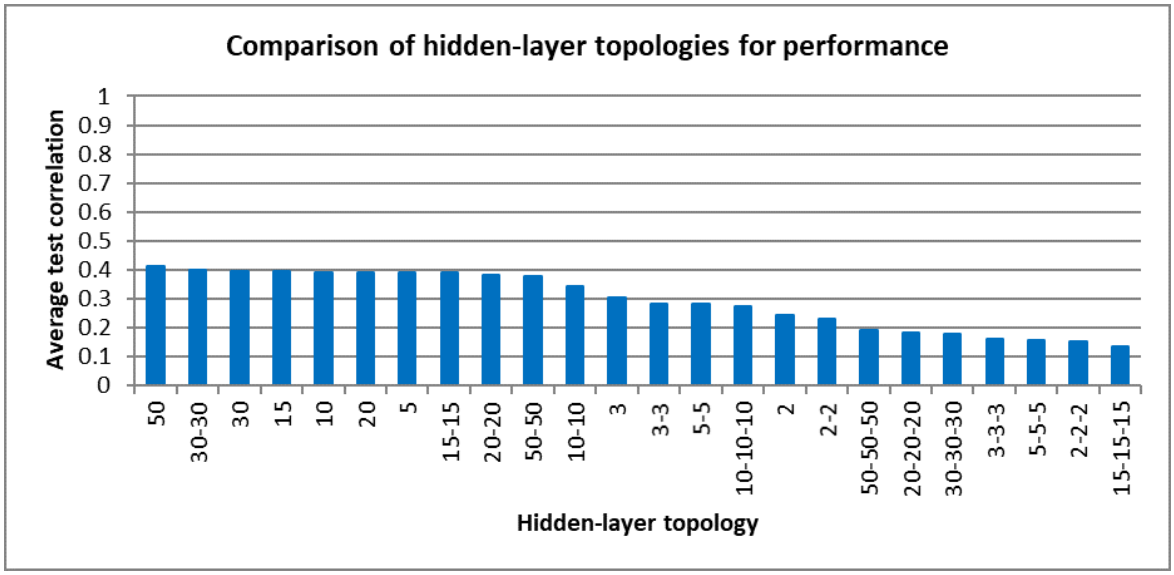


Figure 46 – Comparison of correlations for each topology in the fourth experiment.
Source: Author.

Upon investigation of the individually calibrated variables, the results in Figure 47 show that one of the W99 parameters had a correlation above 0.8, three had values between 0.4 and 0.7; and four had very weak correlation values below 0.3.

These results show that the automatic calibration with Neural Networks might not be accurate in this case for half of the variables involved. From the reviewed literature, the calibration of Wiedemann 99 parameters often involves straight road segments, roundabouts or small road networks with low complexity. It is possible that the mathematical complexity of the relationships between these inputs and the simulator output metrics is high for Neural Networks to model, due to highly stochastic nature of traffic microsimulations and the high abstraction of the Wiedemann 99 car-following model.

Finally, evaluation from the user is required. Since the calibration is often an iterative and manual process, the estimates from the Neural Network can offer starting points for refinement when convenient and reduce workload. The discussion from Punzo, Montanino and Ciuffo (2014) holds for the actual sensitivity of the simulations to all the W99 parameters, since it can be practice to calibrate only the parameters that resemble the Wiedemann 74 model, as the former is a more detailed extension of the latter.

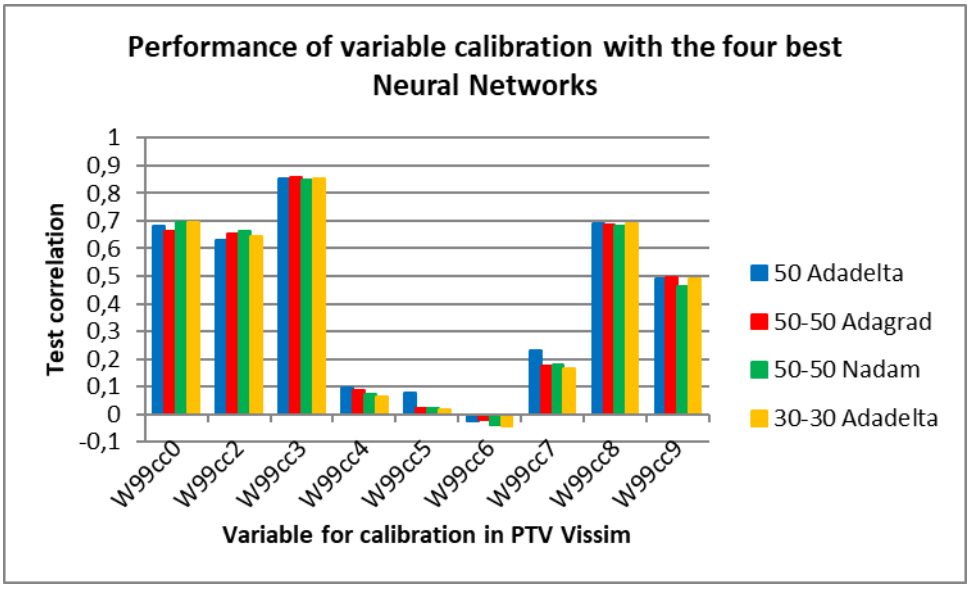


Figure 47 – Comparison of correlations of each variable in the fourth experiment.

Source: Author.

7. Conclusions

This research proposes a methodology to automatically calibrate traffic microsimulations with Neural Networks. The motivation derived from the possibility of reuse of the intermediate results from heuristic optimization algorithms, examples of which are the Genetic Algorithms that have been identified as the state-of-the-art solution for automated calibration.

The research gap was identified as the hypothesis that a calibration function could be defined as the inverse function of the direct computations of the traffic simulator. Building a regression model of such non-linear function, in the case that it is possible to obtain numerous examples of multivariate input-output pairs is a problem for which Neural Networks are among the current state-of-the-art techniques, and hence the link of both subjects, calibration of simulations and supervised learning, was created in this research.

Four major experiments were performed to validate the proposed methodology. Since this research uses techniques from Computer Science and Engineering to solve a problem in the context of Traffic Engineering, the setups of the experiments were developed from the Traffic Engineering references and the proposed methodology, despite automated, allows for feedback and evaluation of the user of the software.

One of the validated hypotheses in the proposed methodology is that the steps can automated in a future software implementation, since they were conducted by scripts through all experiments. The first two experiments calibrated one of the common subjects of interest: the routing information of the road network, represented by volumes and route decisions.

In both experiments, the measurements of correlation between desired values and their estimates by the Neural Networks were above 0.8 and considered high, with exception of one variable in each. It has been argued that the Neural Network development process delivers the metrics of the individual variables, and that traffic engineers have the chance to evaluate if results are satisfactory and should be used in their work. In those grounds the methodology has been validated.

In the last two experiments the subject of calibration were the driving behavior models, which are abstract and posed high complexity for the Neural Networks to model. There is ongoing discussion about the effects of the behavioral parameters and the sensitivity of the traffic simulations to the models as the road networks grow in size, since the opposite in scale are macrosimulations that rely heavily on statistics and have no individualized behavior for the vehicles.

The difficulties in calibrating the driving behaviors could be observed as the measured performance metrics were lower in the last two experiments. Even so, for some of the parameters of the Wiedemann 74 car-following model, the lane-change headway and half of the Wiedemann 99 parameters, the methodology provided results that can serve as a starting point for refinement, and one of the contribution of this research lies on the fact that the previous solutions did not take advantage of data reuse to increase scalability.

One direction for further research is establishing criteria to identify beforehand the variables that would have a weak correlation in the regression model, of which one possibility is to perform sensitivity analyses. Another direction for improvement is to study how many measuring entities on the road network (e.g. vehicle counters, time counters, queue counters and measurements of speeds) are sufficient for the regression models to deliver satisfactory calibration performance. Finally, a third direction for development is the expansion of the proposed methodology to macrosimulations, since the calibration of this group is also a topic of research.

REFERENCES

ABADI, Martín et al. Tensorflow: a system for large-scale machine learning. In: OSDI. **Proceedings…,** 2016. p. 265-283.

AGHABAYK, Kayvan et al. A novel methodology for evolutionary calibration of Vissim by multi-threading. In: AUSTRALASIAN TRANSPORT RESEARCH FORUM. **Proceedings…**. 2013. p. 1-15.

ALKHEDER, Sharaf; TAAMNEH, Madhar; TAAMNEH, Salah. Severity prediction of traffic accident using an artificial neural network. **Journal of Forecasting**, v. 36, n. 1, p. 100-108, 2017.

BETHONICO, Felipe Costa. **Calibração de simuladores microscópicos de tráfego através de medidas macroscópicas**. 2016. 116 p. Dissertation (Master's degree in Transport Engineering) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Paulo, 2016.

CARUANA, Rich; LAWRENCE, Steve; GILES, C. Lee. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. **Advances in neural information processing systems**. 2001. p. 402-408.

CHEN, Chen et al. A rear-end collision prediction scheme based on deep learning in the Internet of Vehicles. **Journal of Parallel and Distributed Computing**, v. 117, p. 192-204, 2018.

CHOLLET, François et al. Keras: Deep learning library for theano and tensorflow. Available at <https://keras. io>. Accessed on 24 Oct. 2018.

CHU, Lianyu et al. A calibration procedure for microscopic traffic simulation. In: IEEE INTERNATIONAL CONFERENCE ON INTELLIGENT TRANSPORTATION SYSTEMS. **Proceedings…**, IEEE, 2003. p. 1574-1579.

CIREŞAN, Dan; MEIER, Ueli; SCHMIDHUBER, Jürgen. Multi-column deep neural networks for image classification. **IDSIA.** Manno, 04-12, Feb 2012. Available at <https://arxiv.org/abs/1202.2745>. Accessed on: 24 Oct. 2018

DEKA, Lipika; QUDDUS, Mohammed. Network-level accident-mapping: distance based pattern matching using artificial neural network. **Accident Analysis & Prevention**, v. 65, p. 105-113, 2014.

DHAMANIYA, Ashish; CHANDRA, Satish. Speed Based Capacity Model for Urban Arterial Roads. **Traffic Engineering & Control**, v. 55, n. 2, 2014.

DUCHI, John; HAZAN, Elad; SINGER, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, n. Jul, p. 2121-2159, 2011.

FANG, Fang Clara; ELEFTERIADOU, Lily. Some guidelines for selecting microsimulation models for interchange traffic operational analysis. **Journal of Transportation Engineering**, v. 131, n. 7, p. 535-543, 2005.

FRANSSON, Emelie. **Driving behavior modeling and evaluation of merging control strategies** - A microscopic simulation study on Sirat Expressway. 50 p. Dissertation (Master's degree in Transport Engineering) – Linköpings Universitet, Norrköping, 2018.

GLOROT, Xavier; BENGIO, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS. 13. **Proceedings…,** 2010. p. 249-256.

HATTAB, Nour; MOTELICA-HEINO, Mikael. Application of an inverse neural network model for the identification of optimal amendment to reduce copper toxicity in phytoremediated contaminated soils. **Journal of Geochemical Exploration**, v. 136, p. 14-23, 2014.

HAYKIN, Simon. **Neural networks and learning machines**. Upper Saddle River: Pearson, 2009. 906 p. ISBN 0131471392, 9780131471399.

HENCLEWOOD, Dwayne et al. A calibration procedure for increasing the accuracy of microscopic traffic simulation models. **Simulation**, v. 93, n. 1, p. 35-47, 2017.

HOLLANDER, Yaron; LIU, Ronghui. The principles of calibrating traffic microsimulation models. **Transportation**, v. 35, n. 3, p. 347-362, 2008.

KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KLAPPER, Amanda B. et al. **Soluções para o transporte público no retorno da Rodovia Raposo Tavares e Av. São Camilo**. 2017.14 p. Escola Politécnica, Universidade de São Paulo, São Paulo, 2017. Course report (Bachelor's Degree in Civil Engineering).

KOZA, John R. et al. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. **Artificial Intelligence in Design'96**. Springer, Dordrecht, 1996. p. 151-170.

LLANQUE AYALA, Rosemary Janneth. **Procedimento para identificação dos principais parâmetros dos microssimuladores a serem considerados no processo de calibração**. 2013. 226 p. Dissertation (Master's degree in Transport Engineering) - Universidade de Brasília, Brasília, 2013.

LOPES, N.; RIBEIRO, B. Hybrid learning in a multi-neural network architecture. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, **Proceedings…**, IEEE, 2001. p. 2788-2793.

LOPES, Noel; RIBEIRO, Bernardete. An efficient gradient-based learning algorithm applied to neural networks with selective actuation neurons. **Neural, Parallel & Scientific Computations**, v. 11, n. 3, p. 253-272, 2003.

LOPES, Noel; RIBEIRO, Bernardete. GPU implementation of the multiple back-propagation algorithm. In: INTERNATIONAL CONFERENCE ON INTELLIGENT DATA ENGINEERING AND AUTOMATED LEARNING, **Proceedings…,** Springer, Berlin, Heidelberg, 2009. p. 449-456.

LOPES, Noel; RIBEIRO, Bernardete. Stochastic GPU-based Multithread Implementation of Multiple Back-propagation. In: ICAART. **Proceedings…**, 2010. p. 271-276.

LOPES, Noel; RIBEIRO, Bernardete. An evaluation of multiple feed-forward networks on GPUs. **International journal of neural systems**, v. 21, n. 01, p. 31-47, 2011.

MA, Xiaolei et al. Large-scale transportation network congestion evolution prediction using deep learning theory. **PloS one**, v. 10, n. 3, p. e0119044, 2015.

MARTE, C. L. et al. **Atividade prática 3 (Lab_ITS_03): VISSIM (Construção de uma Rede de Simulação)**. 2017. Escola Politécnica, Universidade de São Paulo, São Paulo, 2017. Course report (Bachelor's degree in Civil Engineering).

MARTE, C. L. et al. **Atividade prática 4 (Lab_ITS_04): VISSIM (Validação de uma Rede de Simulação)**. 2017. Escola Politécnica, Universidade de São Paulo, São Paulo, 2017. Course report (Bachelor's degree in Civil Engineering).

MICROSOFT CORPORATION. Microsoft Excel. Available at <https://products.office.com/pt-br/excel>. Accessed on 19 Jul. 2019.

MILLER, David Michael. **Developing a procedure to identify parameters for calibration of a VISSIM model**. 2009. 169 p. Dissertation (Master's degree in Civil & Environmental Engineering) - Georgia Institute of Technology, Atlanta, 2009.

MIRANDA, Carolina de Medeiros et al. **Gestão semafórica em tempo real:** estudo de um trecho da avenida dos autonomistas na cidade de Osasco e comparação com o modelo de São Paulo. 2018. 83 p. Graduation project (Bachelor's degree in Civil Engineering) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2018.

MOHANTY, Saraju P.; CHOPPALI, Uma; KOUGIANOS, Elias. Everything you wanted to know about smart cities: The internet of things is the backbone. **IEEE Consumer Electronics Magazine**, v. 5, n. 3, p. 60-70, 2016.

NGWANGWA, Harry Magadhlela et al. Reconstruction of road defects and road roughness classification using vehicle responses with artificial neural networks simulation. **Journal of Terramechanics**, v. 47, n. 2, p. 97-111, 2010.

OLIVEIRA, M. L. d.; CYBIS, H. B. B. Revisão da Experiência de Calibração do Software Vissim Aplicado a um Estudo de Caso de Autoestrada Brasileira.In: SEPROSUL, 8. **Proceedings…**, Bento Gonçalves, Brasil, 2008.

OPENSTREETMAP. Available at <https://www.openstreetmap.org>. Accessed on 19 Jul. 2019.

OTKOVIĆ, Irena Ištoka; TOLLAZZI, Tomaž; ŠRAML, Matjaž. Calibration of microsimulation traffic model using neural network approach. **Expert systems with applications**, v. 40, n. 15, p. 5965-5974, 2013.

PANDAS. Available at <https://pandas.pydata.org>. Accessed on 19 Jul. 2019.

PARK, Byungkyu; QI, Hongtu. Development and Evaluation of a Procedure for the Calibration of Simulation Models. **Transportation Research Record:** Journal of the Transportation Research Board, n. 1934, p. 208-217, 2005.

PEČAR, Martin; PAPA, Gregor. Transportation problems and their potential solutions in smart cities. In: INTERNATIONAL CONFERENCE ON SMART SYSTEMS AND TECHNOLOGIES. **Proceedings…**, IEEE, 2017. p. 195-199.

PLANUNG TRANSPORT VERKEHR AG. PTV Vissim. Available at <http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/>. Accessed on 24 Oct. 2018.

PUNZO, Vincenzo; MONTANINO, Marcello; CIUFFO, Biagio. Do we really need to calibrate all the parameters? Variance-based sensitivity analysis to simplify microscopic traffic flow models. **IEEE Transactions on Intelligent Transportation Systems**, v. 16, n. 1, p. 184-193, 2014.

PYTHON SOFTWARE FOUNDATION. **Python Language Reference**, version 2.7. Available at <http://www.python.org>. Accessed on 24 Oct. 2018.

RRECAJ, Arlinda Alimehaj; MBOMBOL, Kristi. Calibration and Validation of the VISSIM Parameters-State of the Art. **TEM Journal**, v. 4, n. 3, p. 255, 2015.

RUDER, Sebastian. An overview of gradient descent optimization algorithms. **arXiv preprint arXiv:1609.04747**, 2016.

SAMUEL, Arthur L. Some studies in machine learning using the game of checkers. **IBM Journal of research and development**, v. 3, n. 3, p. 210-229, 1959.

SCHMIDHUBER, Jürgen. Deep learning in neural networks: An overview. **Neural networks**, v. 61, p. 85-117, 2015.

SHAFIEI, Sajjad; GU, Ziyuan; SABERI, Meead. Calibration and validation of a simulation-based dynamic traffic assignment model for a large-scale congested network. **Simulation Modelling Practice and Theory**, v. 86, p. 169-186, 2018.

SINGH, Gurjeet; PANDA, Rabindra K. Daily sediment yield modeling with artificial neural network using 10-fold cross validation method: a small agricultural watershed, Kapgari, India. **International Journal of Earth Sciences and Engineering**, v. 4, n. 6, p. 443-450, 2011.

SPACK, M. Numbers Every Traffic Engineer Should Know. Mike On Traffic. 2011. Available at <http://www.mikeontraffic.com/numbers-every-trafficengineer-should-know/>. Accessed on 29 Oct. 2018.

SUKENNIK, Peter; KAUTZSCH, Lukas. Default behavioural parameter sets for Automated Vehicles (AVs). In: CoEXist. **Proceedings…**, 2018

SUTSKEVER, Ilya et al. On the importance of initialization and momentum in deep learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING. **Proceedings…**, 2013. p. 1139-1147.

TANG, Jinjun et al. Travel time estimation using freeway point detector data based on evolving fuzzy neural inference system. **PloS one**, v. 11, n. 2, p. e0147263, 2016.

TETTAMANTI, T. et al. Iterative calibration of VISSIM simulator based on genetic algorithm. **Acta Technica Jaurinensis**, v. 8, n. 2, p. 145-152, 2015.

VAN GERVEN, Marcel; BOHTE, Sander (Ed.). **Artificial neural networks as models of neural information processing**. Frontiers Media SA, 2018. ISBN: 9782889454013.

VILARINHO, Cristina Alexandra Teixeira. **Calibração de modelos microscópicos de simulação de tráfego em redes urbanas.** Dissertation (Master's degree in Civil Engineering) - Faculdade de Engenharia, Universidade do Porto, Porto, 2008.

VU, Vinh An; TAN, Gary. High-performance mesoscopic traffic simulation with GPU for large scale networks. In: INTERNATIONAL SYMPOSIUM ON DISTRIBUTED SIMULATION AND REAL TIME APPLICATIONS, 21. **Proceedings...**, IEEE Press, 2017. p. 127-135.

WIEDEMANN, Rainer. Simulation des Strassenverkehrsflusses. **Schriftenreihe des Instituts für Verkehrswesen der Universität Karlsruhe**, Band 8, Karlsruhe, Germany. 1974.

XIONG, Zhang et al. Intelligent transportation systems for smart cities: a progress review. **Science China Information Sciences**, v. 55, n. 12, p. 2908-2914, 2012.

XU, Yan et al. Mesoscopic traffic simulation on CPU/GPU. In: ACM SIGSIM CONFERENCE ON PRINCIPLES OF ADVANCED DISCRETE SIMULATION, 2. **Proceedings…**, ACM, 2014. p. 39-50.

YIN, Derek; QIU, Tony Z. Compatibility analysis of macroscopic and microscopic traffic simulation modeling. **Canadian Journal of Civil Engineering**, v. 40, n. 7, p. 613-622, 2013.

YU, Miao; FAN, Wei David. Calibration of microscopic traffic simulation models using metaheuristic algorithms. **International Journal of Transportation Science and Technology**, v. 6, n. 1, p. 63-77, 2017.

ZEILER, Matthew D. ADADELTA: an adaptive learning rate method. **arXiv preprint arXiv:1212.5701**, 2012.

ZHANG, Jian; EL KAMEL, Abdelkader. Virtual traffic simulation with neural network learned mobility model. **Advances in Engineering Software**, v. 115, p. 103-111, 2018.

ZHOU, Mofan; QU, Xiaobo; LI, Xiaopeng. A recurrent neural network based microscopic car following model to predict traffic oscillation. **Transportation research part C: emerging technologies**, v. 84, p. 245-264, 2017.

APPENDIX A

Python scripts run in the PTV Vissim COM interface to automate simulation runs.

1. **Variation of vehicle volume inputs and route decisions (experiments 1 and 2):**

```
# import format: 3 vehicle volumes + 14 static route RelFlows
# Vol1, Vol2, ..., Rout1_opt1, Rout1_opt2, ..., Rout2_opt1,...
import csv

with open('inputs.csv', 'rb') as csvfile:
    all_flows = Vissim.Net.VehicleInputs.GetAll()
    all_routes = Vissim.Net.VehicleRoutingDecisionsStatic.GetAll()
    myfile = csv.reader(csvfile, delimiter=';')

    # first row is the variable names
    input_variable_names = myfile.next()

    # ranges up to excel row+1
    for k in range(4000):
        this_line = myfile.next()
        index = 0
        for t in range(len(all_flows)):
            all_flows[t].SetAttValue("Volume(1)",
this_line[index])
            index+=1
        for m in range(len(all_routes)):
            options = all_routes[m].VehRoutSta
            for n in range(len(options)):
                options[n].SetAttValue("RelFlow(1)",
this_line[index])
                index+=1

        Vissim.Simulation.RunContinuous()
```

2. **Variation of the driving behavior Wiedemann 74 and lane change parameters (experiment 3):**

```
# import format: W74ax, W74bxAdd, W74bxMult, MinHdwy
import csv

with open('inputs.csv', 'rb') as csvfile:
    driving_behavior_list =
Vissim.Net.DrivingBehaviors.GetAll()

    myfile = csv.reader(csvfile, delimiter=';')

    # first row is the variable names
    input_variable_names = myfile.next()

    # ranges up to excel row+1
    for k in range(3000):
        this_line = myfile.next()

        driving_behavior_list[0].SetAttValue("W74ax",
this_line[0])
        driving_behavior_list[0].SetAttValue("W74bxAdd",
this_line[1])
        driving_behavior_list[0].SetAttValue("W74bxMult",
this_line[2])
        driving_behavior_list[0].SetAttValue("MinHdwy",
this_line[3])

        Vissim.Simulation.RunContinuous()
```

3. Variation of the driving behavior Wiedemann 99 parameters (experiment 4):

```
# import format: W99 CC 1-9
import csv

with open('inputs.csv', 'rb') as csvfile:
    driving_behavior_list =
Vissim.Net.DrivingBehaviors.GetAll()

    myfile = csv.reader(csvfile, delimiter=';')

    # first row is the variable names
    input_variable_names = myfile.next()

    # ranges up to excel row+1
    for k in range(2000):
        this_line = myfile.next()

# W99cc1 is not an editable value, but drop-down menu
        driving_behavior_list[0].SetAttValue("W99cc0",
this_line[0])
        # driving_behavior_list[0].SetAttValue("W99cc1",
this_line[1])
        driving_behavior_list[0].SetAttValue("W99cc2",
this_line[1])
        driving_behavior_list[0].SetAttValue("W99cc3",
this_line[2])
        driving_behavior_list[0].SetAttValue("W99cc4",
this_line[3])
        driving_behavior_list[0].SetAttValue("W99cc5",
this_line[4])
        driving_behavior_list[0].SetAttValue("W99cc6",
this_line[5])
        driving_behavior_list[0].SetAttValue("W99cc7",
this_line[6])
        driving_behavior_list[0].SetAttValue("W99cc8",
this_line[7])
        driving_behavior_list[0].SetAttValue("W99cc9",
this_line[8])

        Vissim.Simulation.RunContinuous()
```

APPENDIX B

Python scripts run in the Python environment to create the training datasets from the concatenation of the PTV Vissim outputs and inputs, in inverse order.

Experiment 1:

```
import pandas as pd

inputs = pd.read_csv('inputs8.csv', decimal=',', sep=';')
times = pd.read_excel('results8.xlsx', decimal=',', sep=';',
sheet_name='times')

inputs = inputs.drop(columns=['Rout1b', 'Rout2b', 'Rout3b',
'Rout4b', 'Rout5b'])
times = times.pivot(index='SimRun',
columns='VehicleTravelTimeMeasurement')
times = times.reset_index()

dataset = pd.concat([times, inputs], axis=1)

dataset.columns = dataset.columns.map(str)
dataset.rename(columns='_'.join, inplace=True)
dataset.columns = dataset.columns.str.replace('[^a-zA-Z0-9]', '')

dataset = dataset.drop(columns='SimRun')
dataset = dataset.dropna()

dataset.to_csv('dataset8.csv', decimal='.', sep=',', index=False)
```

Experiment 2:

```
import pandas as pd

inputs = pd.read_csv('inputs5.csv', decimal=',', sep=';')
times = pd.read_excel('results5.xlsx', decimal=',', sep=';',
sheet_name='times')
counts = pd.read_excel('results5.xlsx', decimal=',', sep=';',
sheet_name='counts')

inputs = inputs.drop(columns=['Rout_1b', 'Rout_2b', 'Rout_3b',
'Rout_4b', 'Rout_5b', 'Rout_6b', 'Rout_7b'])
times = times.pivot(index='SimRun',
columns='VehicleTravelTimeMeasurement')
times = times.reset_index()
counts = counts.pivot(index='SimRun',
columns='DataCollectionMeasurement')
counts = counts.reset_index()

dataset = pd.concat([times, counts, inputs], axis=1)

# for multi-level indexing only
dataset.columns = dataset.columns.map(str)
dataset.rename(columns='_'.join, inplace=True)
dataset.columns = dataset.columns.str.replace('[^a-zA-Z0-9]', '')

dataset = dataset.drop(columns='SimRun')
dataset = dataset.dropna()

dataset.to_csv('dataset5.csv', decimal='.', sep=',', index=False)
```

Experiment 3:

```
import pandas as pd

inputs = pd.read_csv('inputs6.csv', decimal=',', sep=';')
times = pd.read_excel('results6.xlsx', decimal=',', sep=';',
sheet_name='times')
counts = pd.read_excel('results6.xlsx', decimal=',', sep=';',
sheet_name='counts')

times = times.pivot(index='SimRun',
columns='VehicleTravelTimeMeasurement')
times = times.reset_index()
counts = counts.pivot(index='SimRun',
columns='DataCollectionMeasurement')
counts = counts.reset_index()

dataset = pd.concat([times, counts, inputs], axis=1)

# for multi-level indexing only
dataset.columns = dataset.columns.map(str)
dataset.rename(columns='_'.join, inplace=True)
dataset.columns = dataset.columns.str.replace('[^a-zA-Z0-9]', '')

dataset = dataset.drop(columns='SimRun')
dataset = dataset.dropna()

dataset.to_csv('dataset6.csv', decimal='.', sep=',', index=False)
```

Experiment 4:

```
import pandas as pd

inputs = pd.read_csv('inputs7.csv', decimal=',', sep=';')
times = pd.read_excel('results7.xlsx', decimal=',', sep=';',
sheet_name='times')
counts = pd.read_excel('results7.xlsx', decimal=',', sep=';',
sheet_name='counts')

times = times.pivot(index='SimRun',
columns='VehicleTravelTimeMeasurement')
times = times.reset_index()
counts = counts.pivot(index='SimRun',
columns='DataCollectionMeasurement')
counts = counts.reset_index()

dataset = pd.concat([times, counts, inputs], axis=1)

# for multi-level indexing only
dataset.columns = dataset.columns.map(str)
dataset.rename(columns='_'.join, inplace=True)
dataset.columns = dataset.columns.str.replace('[^a-zA-Z0-9]', '')

dataset = dataset.drop(columns='SimRun')
dataset = dataset.dropna()

dataset.to_csv('dataset7.csv', decimal='.', sep=',', index=False)
```

APPENDIX C

Python scripts using the Keras library to train and evaluate the Neural Network models.

Experiment 1:

```
import matplotlib.pyplot as plt
import keras
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.models import load_model
import pickle

layer_width = [50, 20, 15, 10, 5, 3, 2]

n_inputs = 26
n_outputs = 10

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
keras.backend.set_session(tf.Session(config=config))

dataset = pd.read_csv('dataset8.csv')

train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)

train_stats = train_dataset.describe()
train_stats = train_stats.transpose()


def norm(x):
    return (x - train_stats['mean']) / train_stats['std']


normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)

train_data_as_numpy = normed_train_data.values
test_data_as_numpy = normed_test_data.values
x_train = train_data_as_numpy[:, 0:n_inputs]
y_train = train_data_as_numpy[:, n_inputs:]
x_test = test_data_as_numpy[:, 0:n_inputs]
y_test = test_data_as_numpy[:, n_inputs:]

for width in layer_width:
    model = Sequential()
    model.add(Dense(units=width, activation='sigmoid',
input_dim=n_inputs))
```

```
    model.add(Dense(units=width, activation='sigmoid'))
    # model.add(Dense(units=width, activation='sigmoid'))
    model.add(Dense(units=n_outputs, activation='sigmoid'))
    model.compile(loss='mean_squared_error',
                  optimizer='Nadam', metrics=['mean_squared_error'])

    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',
min_delta=0, patience=2)

    history = model.fit(x_train, y_train, validation_split=0.2,
epochs=50000, callbacks=[early_stop, keras.callbacks.TensorBoard()],
verbose=1)


    def plot_history(history):
        hist = pd.DataFrame(history.history)
        hist['epoch'] = history.epoch

        plt.figure()
        plt.xlabel('Epoch')
        plt.ylabel('Mean Square Error')
        plt.plot(hist['epoch'], hist['mean_squared_error'],
                 label='Train Error')
        plt.plot(hist['epoch'], hist['val_mean_squared_error'],
                 label='Val Error')
        plt.ylim([0, 1])
        plt.legend()
        plt.show()


    # plot_history(history)

    # hist = pd.DataFrame(history.history)

    # with open('history.txt', 'wb') as file:
    #     pickle.dump(history.history, file)

    loss, mse = model.evaluate(x_test, y_test, verbose=0)

    outputs = model.predict(x_test, verbose=0)

    correlations = np.zeros(outputs.shape[1])

    for i in range(len(correlations)):
        correlations[i] = np.corrcoef(y_test[:, i], outputs[:, i])[0, 1]

    print(normed_test_data.columns.values[26:])
    print(correlations)

       # model.save('calibration_nn.h5')
    del model
```

Experiment 2:

```
import matplotlib.pyplot as plt
import keras
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
import pickle

layer_width = [50, 20, 15, 10, 5, 3]

n_inputs = 40
n_outputs = 10

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
keras.backend.set_session(tf.Session(config=config))

dataset = pd.read_csv('dataset5.csv')

train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)

train_stats = train_dataset.describe()
train_stats = train_stats.transpose()
# train_stats.to_csv('train_stats.csv')


def norm(x):
    return (x - train_stats['mean']) / train_stats['std']


normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)

train_data_as_numpy = normed_train_data.values
test_data_as_numpy = normed_test_data.values
x_train = train_data_as_numpy[:, 0:n_inputs]
y_train = train_data_as_numpy[:, n_inputs:]
x_test = test_data_as_numpy[:, 0:n_inputs]
y_test = test_data_as_numpy[:, n_inputs:]

for width in layer_width:

    model = Sequential()
    model.add(Dense(units=width, activation='sigmoid',
input_dim=n_inputs))
    model.add(Dense(units=width, activation='sigmoid'))
    # model.add(Dense(units=width, activation='sigmoid'))
    model.add(Dense(units=n_outputs, activation='sigmoid'))
```

```
    model.compile(loss='mean_squared_error', optimizer='Nadam',
metrics=['mean_squared_error'])

    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',
min_delta=0, patience=2)

    history = model.fit(x_train, y_train, validation_split=0.2,
epochs=50000, callbacks=[early_stop], verbose=1)


    def plot_history(history):
        hist = pd.DataFrame(history.history)
        hist['epoch'] = history.epoch
        epoch = history.epoch
        print("Epochs: " + epoch[-1])

        plt.figure()
        plt.xlabel('Epoch')
        plt.ylabel('Mean Square Error')
        plt.plot(hist['epoch'], hist['mean_squared_error'],
                 label='Train Error')
        plt.plot(hist['epoch'], hist['val_mean_squared_error'],
                 label='Val Error')
        plt.ylim([0, 1])
        plt.legend()
        plt.show()


    # plot_history(history)

    # hist = pd.DataFrame(history.history)

    # with open('history.txt', 'wb') as file:
    #     pickle.dump(history.history, file)

    loss, mse = model.evaluate(x_test, y_test, verbose=0)

    outputs = model.predict(x_test, verbose=0)

    correlations = np.zeros(outputs.shape[1])

    for i in range(len(correlations)):
        correlations[i] = np.corrcoef(y_test[:, i], outputs[:, i])[0, 1]

    print(normed_test_data.columns.values[n_inputs:])
    print(correlations)
    del model
```

Experiment 3:

```
import matplotlib.pyplot as plt
import keras
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
import pickle

layer_width = [50, 10, 5, 3, 2, 1]

n_inputs = 40
n_outputs = 4

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
keras.backend.set_session(tf.Session(config=config))

dataset = pd.read_csv('dataset6.csv')

train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)

train_stats = train_dataset.describe()
train_stats = train_stats.transpose()
# train_stats.to_csv('train_stats.csv')


def norm(x):
    return (x - train_stats['mean']) / train_stats['std']


normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)

train_data_as_numpy = normed_train_data.values
test_data_as_numpy = normed_test_data.values
x_train = train_data_as_numpy[:, 0:n_inputs]
y_train = train_data_as_numpy[:, n_inputs:]
x_test = test_data_as_numpy[:, 0:n_inputs]
y_test = test_data_as_numpy[:, n_inputs:]

for width in layer_width:
    model = Sequential()
    model.add(Dense(units=width, activation='sigmoid',
input_dim=n_inputs))
    model.add(Dense(units=width, activation='sigmoid'))
    # model.add(Dense(units=width, activation='sigmoid'))
    model.add(Dense(units=n_outputs, activation='sigmoid'))

    model.compile(loss='mean_squared_error',
                  optimizer='Nadam', metrics=['mean_squared_error'])
```

```
    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',
min_delta=0, patience=2)

    history = model.fit(x_train, y_train, validation_split=0.2,
epochs=50000, callbacks=[early_stop], verbose=1)


    def plot_history(history):
        hist = pd.DataFrame(history.history)
        hist['epoch'] = history.epoch

        plt.figure()
        plt.xlabel('Epoch')
        plt.ylabel('Mean Square Error')
        plt.plot(hist['epoch'], hist['mean_squared_error'],
                 label='Train Error')
        plt.plot(hist['epoch'], hist['val_mean_squared_error'],
                 label='Val Error')
        plt.ylim([0, 1])
        plt.legend()
        plt.show()


    # plot_history(history)

    # hist = pd.DataFrame(history.history)

    # with open('history.txt', 'wb') as file:
    #     pickle.dump(history.history, file)

    loss, mse = model.evaluate(x_test, y_test, verbose=0)

    outputs = model.predict(x_test, verbose=0)

    correlations = np.zeros(outputs.shape[1])

    for i in range(len(correlations)):
        correlations[i] = np.corrcoef(y_test[:, i], outputs[:, i])[0, 1]

    print(normed_test_data.columns.values[n_inputs:])
    print(correlations)
    del model
```

Experiment 4:

```
import matplotlib.pyplot as plt
import keras
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
import pickle

layer_width = [50, 30, 20, 15, 10, 5, 3, 2]

n_inputs = 40
n_outputs = 9

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
keras.backend.set_session(tf.Session(config=config))

dataset = pd.read_csv('dataset7.csv')

train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)

train_stats = train_dataset.describe()
train_stats = train_stats.transpose()
# train_stats.to_csv('train_stats.csv')


def norm(x):
    return (x - train_stats['mean']) / train_stats['std']


normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)

train_data_as_numpy = normed_train_data.values
test_data_as_numpy = normed_test_data.values
x_train = train_data_as_numpy[:, 0:n_inputs]
y_train = train_data_as_numpy[:, n_inputs:]
x_test = test_data_as_numpy[:, 0:n_inputs]
y_test = test_data_as_numpy[:, n_inputs:]

for width in layer_width:
    model = Sequential()
    model.add(Dense(units=width, activation='sigmoid',
input_dim=n_inputs))
    model.add(Dense(units=width, activation='sigmoid'))
    # model.add(Dense(units=width, activation='sigmoid'))
    model.add(Dense(units=n_outputs, activation='sigmoid'))

    model.compile(loss='mean_squared_error',
                  optimizer='Nadam', metrics=['mean_squared_error'])
```

```
    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',
min_delta=0, patience=2)

    history = model.fit(x_train, y_train, validation_split=0.2,
epochs=50000, callbacks=[early_stop], verbose=1)


    def plot_history(history):
        hist = pd.DataFrame(history.history)
        hist['epoch'] = history.epoch

        plt.figure()
        plt.xlabel('Epoch')
        plt.ylabel('Mean Square Error')
        plt.plot(hist['epoch'], hist['mean_squared_error'],
                 label='Train Error')
        plt.plot(hist['epoch'], hist['val_mean_squared_error'],
                 label='Val Error')
        plt.ylim([0, 1])
        plt.legend()
        plt.show()


    # plot_history(history)

    # hist = pd.DataFrame(history.history)

    # with open('history.txt', 'wb') as file:
    #     pickle.dump(history.history, file)

    loss, mse = model.evaluate(x_test, y_test, verbose=0)

    outputs = model.predict(x_test, verbose=0)

    correlations = np.zeros(outputs.shape[1])

    for i in range(len(correlations)):
        correlations[i] = np.corrcoef(y_test[:, i], outputs[:, i])[0, 1]

    print(normed_test_data.columns.values[n_inputs:])
    print(correlations)
    del model
```

APPENDIX D

Python script to load a previously saved Neural Network file and run it over a new array of inputs.

```
from keras.models import load_model
import keras
import pandas as pd
import numpy as np

new_inputs = pd.read_csv('new_inputs.csv', sep=',', decimal='.')

model = load_model('calibration_nn.h5')

train_stats = pd.read_csv('train_stats.csv', index_col=0, sep=',',
decimal='.')

input_stats = train_stats.drop(index=['Vol1', 'Vol2', 'Vol3',
'Vol4', 'Vol5', 'Rout1a', 'Rout2a', 'Rout3a', 'Rout4a', 'Rout5a'])


def norm(x):
    return (x - input_stats['mean']) / input_stats['std']


normed_new_inputs = norm(new_inputs)

x_numpy = normed_new_inputs.values

y_numpy = model.predict(x_numpy)

new_outputs = pd.DataFrame(y_numpy, columns=['Vol1', 'Vol2', 'Vol3',
'Vol4', 'Vol5', 'Rout1a', 'Rout2a', 'Rout3a', 'Rout4a', 'Rout5a'])

output_stats = train_stats.loc[['Vol1', 'Vol2', 'Vol3', 'Vol4',
'Vol5', 'Rout1a', 'Rout2a', 'Rout3a', 'Rout4a', 'Rout5a']]


def denorm(y):
    return (y * output_stats['std']) + output_stats['mean']


new_outputs = denorm(new_outputs)

new_outputs.to_csv("new_outputs.csv", index=False)
```

APPENDIX E

Tables with the correlations of the individual variables, for all trained Neural Networks in all experiments.

Experiment 1:

		Correlation										
NN settings	Epochs	Vol_1	Vol_2	Vol_3	Vol_4	Vol_5	Rout_1	Rout_2	Rout_3	Rout_4	Rout_5	Average
Keras 50-50 Adadelta	96	0.88963988	0.9080342	0.84240521	0.90000808	0.8411398	0.89287077	0.0327211	0.86927261	0.89022485	0.88494244	0.795125894
Keras 50-50 Adagrad	102	0.88488808	0.90464712	0.83944082	0.90253992	0.83286746	0.88555843	0.06165541	0.86391774	0.89003549	0.88122656	0.794677703
Keras 50 Adadelta		0.88697601	0.90696226	0.83468251	0.90181605	0.84620575	0.88838025	0.04494046	0.86191733	0.88301628	0.88379402	0.793869092
Keras 50-50 SGD	8587	0.88983084	0.91068358	0.83884225	0.90585291	0.83777133	0.89160328	0.01926797	0.8610858	0.88782832	0.88248197	0.792524825
Keras 20 Adadelta		0.88830913	0.91012925	0.83246672	0.89611138	0.82394402	0.88439628	0.05231575	0.86335468	0.88599901	0.87811016	0.791513638
Keras 15 Adadelta		0.88870515	0.90872832	0.82827505	0.89989369	0.82619677	0.89380482	0.04108194	0.86321475	0.88085613	0.87876057	0.790951719
Keras 50-50 Adam	61	0.88083658	0.8983201	0.82158176	0.89952595	0.82898581	0.88881008	0.0517023	0.86665454	0.8876959	0.88140551	0.790551853
Keras 50-50 Nadam	46	0.8744322	0.90107812	0.82350363	0.89445449	0.84030539	0.88905288	0.03797661	0.86700083	0.87654292	0.88241721	0.788676428
Keras 10 Adadelta		0.88686681	0.90125948	0.83575012	0.9054052	0.80761003	0.88632929	0.02767602	0.85212742	0.88690689	0.88978267	0.787971393
Keras 50-50 Adamax	85	0.88175128	0.90244251	0.83154758	0.89726514	0.83817128	0.88945704	0.01050369	0.86357294	0.88298871	0.87512619	0.787282636
Keras 20-20 Adadelta		0.8774082	0.89803578	0.83491244	0.89370652	0.8375738	0.88032964	0.03620772	0.85429025	0.87631716	0.86792493	0.785670644
Keras 15-15 Adadelta		0.87133868	0.89668157	0.83176985	0.89702269	0.32987754	0.88218906	0.04576841	0.84409315	0.8728679	0.86467373	0.733628258
Keras 50-50-50 Adadelta		0.47009117	0.88904958	0.81396858	0.89880622	0.32376578	0.89242187	0.03025602	0.86276048	0.88473767	0.85371259	0.691956996
Keras 10-10 Adadelta		0.86477899	0.89628773	0.83176352	0.89844724	0.04230324	0.87981527	0.01935995	0.85054223	0.39935953	0.8399518	0.65226095
Keras 20-20-20 Adadelta		0.46851041	0.88806292	0.81900235	0.89072557	0.341365	0.89217294	0.01525279	0.83763831	-0.06237872	0.84602485	0.593637642
Keras 15-15-15 Adadelta		0.84639485	0.88982932	0.29398777	0.89245513	0.12174517	0.88595978	0.03748187	0.36408368	0.43052599	0.83505308	0.559751664
Keras 5 Adadelta		0.84838774	0.8680583	0.20944833	0.89757085	0.03842087	0.8860757	0.02609467	0.33461573	0.22584583	0.84986553	0.518438355
Keras 10-10-10 Adadelta		0.46558426	0.8845495	0.31300337	0.88897372	0.04639141	0.87455742	0.00408925	0.00127703	0.81516882	0.86460498	0.515819976
Keras 5-5 Adadelta		0.83220747	0.81754859	0.21163671	0.86482375	0.05941734	0.8709682	0.02057282	0.40298266	0.40751528	0.25129944	0.473897226
Keras 3 Adadelta		0.45227976	0.89525298	0.07043554	0.52889844	0.1518815	0.87731313	0.02073553	0.02679383	0.04012395	0.81166424	0.38753789
Keras 3-3 Adadelta		0.38025469	0.84476504	0.10983211	0.82698518	0.08731916	0.46801468	0.00207892	-0.01560337	0.24959713	0.83770678	0.379095032
Keras 5-5-5 Adadelta		0.46474954	0.88478534	0.07043914	0.51052344	0.08178535	0.85625132	-0.01401901	0.01782141	0.02816933	0.22471892	0.312522478
Keras 2 Adadelta		0.28994843	0.46069404	0.0331598	0.81525085	0.0672782	0.78784426	0.02428283	0.40633669	-0.0450958	0.05807877	0.289777807
Keras 2-2 Adadelta		0.44861718	0.87852906	0.08033388	0.76521064	0.06463259	0.43610536	-0.00524497	0.02297877	0.01304983	0.09322453	0.279743687
Keras 3-3-3 Adadelta		0.86982634	0.83781475	0.00226637	0.07038211	0.04009756	0.49172343	-0.02333576	0.05323867	0.01960109	0.007043	0.236865755
Keras 2-2-2 Adadelta		0.21899124	0.85391627	0.03741041	0.09652482	0.05164143	0.49903866	-0.03149741	0.09565261	0.02979064	0.01771064	0.186917931

Experiment 2:

							Correlation						
NN settings	Epochs	Vol_1	Vol_2	Vol_3	Rout_1	Rout_2	Rout_3	Rout_4	Rout_5	Rout_6	Rout_7	Average
Keras 50-50 Adam	84	0.86940132	0.88685541	0.88875575	0.88530436	0.85118906	0.86996409	0.88791071	0.84876074	0.87633715	-0.02568743	0.783879116
Keras 50-50 SGD	8337	0.87427934	0.89035043	0.90113538	0.88112686	0.84233216	0.86898835	0.88253301	0.78970863	0.86919387	0.00275544	0.780240347
Keras 10 Adadelta		0.85293152	0.88595659	0.90099175	0.86987851	0.83827249	0.86693192	0.87479635	0.83266396	0.85947074	0.00907403	0.779096786
Keras 50-50 Adagrad	951	0.86793183	0.88738953	0.89086801	0.88186743	0.84654058	0.87441656	0.88072984	0.81185964	0.86958212	-0.02700678	0.778417876
Keras 50-50 Adadelta	117	0.87295967	0.88519572	0.89138043	0.87998744	0.8497705	0.87686189	0.88508411	0.81042067	0.85806912	-0.02962927	0.778010028
Keras 15 Adadelta		0.86063315	0.88349582	0.89287821	0.8895902	0.83934493	0.86949919	0.87359663	0.83158109	0.8609776	-0.02192558	0.777967561
Keras 50-50 Nadam	39	0.86602235	0.88016191	0.8873788	0.88218083	0.84733749	0.85815391	0.88573032	0.83496771	0.87255119	-0.03665971	0.77778248
Keras 50 Adadelta		0.85758406	0.88490672	0.89465038	0.88396346	0.84222839	0.87029718	0.88847174	0.81091763	0.8638303	-0.03105193	0.776579793
Keras 50-50 Adamax	138	0.847661	0.88017	0.88086796	0.88138655	0.84593902	0.87339742	0.88176203	0.82000879	0.87208814	-0.01963495	0.776364596
Keras 20 Adadelta		0.85988598	0.8921356	0.88883518	0.89061585	0.85052693	0.86391357	0.87704758	0.80419462	0.87181106	-0.04120679	0.775775958
Keras 20-20 Adadelta		0.8595386	0.86978455	0.88817519	0.87086614	0.83865199	0.86638203	0.86165975	0.83780872	0.87418009	-0.06450687	0.770254019
Keras 10-10 Adadelta		0.86245391	0.88640982	0.87201825	0.86574411	0.84963641	0.85754151	0.88015723	0.84777441	0.85452697	-0.07814808	0.769811454
Keras 15-15 Adadelta		0.85300409	0.86848289	0.88622447	0.87063081	0.84135337	0.85712656	0.86826463	0.8085464	0.85886301	-0.03760305	0.767489318
Keras 20-20-20 Adadelta		0.491438	0.86975844	0.8529429	0.86596322	0.82061446	0.85696369	0.8659067	0.48271119	0.8551431	0.00524813	0.696668983
Keras 50-50-50 Adadelta		0.81459565	0.88514776	0.88194535	0.79639706	0.78470426	0.61891518	0.87251287	0.49326662	0.8565597	-0.03918089	0.696486356
Keras 5 Adadelta		0.61842603	0.82863693	0.86465783	0.69655294	0.6141229	0.06029195	0.8711338	0.35131454	0.49994978	-0.05494548	0.535014122
Keras 5-5 Adadelta		0.75463627	0.83681682	0.86496341	0.85901458	0.77055383	0.24636019	0.382561	0.26086829	0.1180169	-0.00587791	0.508791338
Keras 15-15-15 Adadelta		0.83609238	0.81493179	0.87844028	0.87969323	0.4502598	0.20332781	0.05845353	0.22463864	0.26556061	-0.02011117	0.45912869
Keras 10-10-10 Adadelta		0.84616978	0.86265688	0.85386225	0.85860808	0.17499031	0.10204119	0.03446398	0.22303416	0.48394968	0.01350761	0.445328392
Keras 5-5-5 Adadelta		0.47828774	0.81777466	0.79220122	0.84288859	0.473648	0.09243419	0.02192594	0.17659722	0.42361915	-0.01248044	0.410689627
Keras 3 Adadelta		0.45259669	0.73965612	0.03127356	0.78786345	0.29018226	0.51080897	0.83767472	0.0807908	0.10296154	0.01325247	0.384706058
Keras 3-3 Adadelta		0.03597093	0.86415853	0.80622372	0.85339531	0.02161914	0.17700293	0.0487495	0.05459658	0.43001449	0.04015707	0.33318882
Keras 3-3-3 Adadelta		0.06982726	0.81612118	0.80235026	0.03959677	0.07333056	0.42931384	0.1179581	0.02293118	0.4306383	0.02621425	0.28282817

Experiment 3:

		Correlation				
NN settings	**Epochs**	**W74ax**	**W74bxAdd**	**W74bxMult**	**HeadwayLaneChange**	**Average**
Keras 10 Adadelta		0.82744514	0.73205179	0.55050382	0.85052307	0.740130955
Keras 50-50 SGD	6627	0.82597493	0.73045219	0.53542305	0.85138868	0.735809713
Keras 50-50 Adagrad	667	0.8202803	0.72979363	0.52675644	0.85463448	0.732866213
Keras 5 Adadelta		0.8284987	0.71767662	0.53825702	0.83722585	0.730414548
Keras 50-50 Adamax	68	0.81567189	0.71467057	0.52402788	0.84958111	0.725987863
Keras 50 Adadelta		0.81290327	0.71088586	0.53574365	0.84162438	0.72528929
Keras 50-50 Nadam	24	0.81184628	0.71192222	0.51417265	0.84251744	0.720114648
Keras 10-10 Adadelta		0.8191621	0.69921326	0.51644338	0.84126279	0.719020383
Keras 50-50 Adam	42	0.80466827	0.69862179	0.51876204	0.84667394	0.71718151
Keras 50-50 Adadelta	57	0.81245311	0.69095628	0.52019235	0.84098844	0.716147545
Keras 3-3 Adadelta		0.827451	0.6875826	0.51390996	0.83240693	0.715337623
Keras 50-50-50 Adadelta		0.79805387	0.69143609	0.52351269	0.84441539	0.71435451
Keras 3 Adadelta		0.82407297	0.68676344	0.50548182	0.83993494	0.714063293
Keras 5-5 Adadelta		0.81837327	0.68390253	0.50741743	0.83719595	0.711722295
Keras 10-10-10 Adadelta		0.8080715	0.66276677	0.51046118	0.83020229	0.702875435
Keras 5-5-5 Adadelta		0.78954623	0.67856075	0.49995989	0.8398604	0.701981818
Keras 2-2 Adadelta		0.80170527	0.69105007	0.50244555	0.09169528	0.521724043
Keras 3-3-3 Adadelta		0.80164769	0.68338834	0.50144345	0.09400452	0.520121
Keras 2 Adadelta		0.81157527	0.688282	0.50815715	-0.07037097	0.484410863
Keras 2-2-2 Adadelta		0.3875158	0.66173387	0.45636839	0.17634942	0.42049187
Keras 1 Adadelta		0.68719056	0.41248284	0.25418703	0.12818089	0.37051033
Keras 1-1-1 Adadelta		0.64846014	0.42085591	0.29885588	-0.01968541	0.33712163
Keras 1-1 Adadelta		0.32774311	0.61120822	0.42859119	-0.07568311	0.322964853

Experiment 4:

			Correlation								
NN settings	Epochs	W99cc0	W99cc2	W99cc3	W99cc4	W99cc5	W99cc6	W99cc7	W99cc8	W99cc9	Average
Keras 50 Adadelta		0.67994408	0.63161451	0.85082674	0.09465286	0.07881449	-0.02622938	0.2325094	0.68998811	0.49151402	0.413737203
Keras 50-50 Adagrad	869	0.661642	0.65286355	0.8583192	0.08438413	0.02353456	-0.02218539	0.17691857	0.6872705	0.49659844	0.402149507
Keras 50-50 Nadam	49	0.69286431	0.6638401	0.85028254	0.07189521	0.02312333	-0.03678529	0.17805327	0.68161318	0.4616457	0.398503594
Keras 30-30 Adadelta		0.69700445	0.64575143	0.85378669	0.06546901	0.01671138	-0.04490702	0.16416489	0.69187558	0.4909388	0.397866134
Keras 30 Adadelta		0.69337075	0.63036248	0.85053633	0.08960549	-0.05202433	-0.04102024	0.20406419	0.69320283	0.5039621	0.396895511
Keras 50-50 Adam	67	0.69296348	0.6469937	0.84672171	0.07923129	-0.01812136	-0.02537193	0.16586795	0.68268525	0.50041188	0.396820219
Keras 15 Adadelta		0.66946464	0.6363558	0.85084036	0.07672927	-0.03334974	-0.03676572	0.20797288	0.67569586	0.49432869	0.393474671
Keras 50-50 SGD	10617	0.69174245	0.64449313	0.84750817	0.01894159	0.0057077	-0.03271713	0.17167728	0.68194536	0.50482259	0.392680127
Keras 10 Adadelta		0.678980644	0.638928442	0.858109856	0.07382374	-0.000188441	-0.035253134	0.168653323	0.64613575	0.502010332	0.392355612
Keras 20 Adadelta		0.69428683	0.62383891	0.85641758	0.06119287	-0.00986928	-0.05621054	0.17903332	0.67575059	0.50605754	0.392277536
Keras 5 Adadelta		0.683456643	0.603099724	0.858073754	-0.000153343	0.027163253	-0.025698775	0.18239566	0.700758247	0.492811849	0.391323001
Keras 50-50 Adamax	87	0.68218321	0.6141295	0.85227262	0.02143897	0.02808606	-0.02279812	0.19549253	0.65720475	0.4775153	0.389511228
Keras 15-15 Adadelta		0.70276432	0.63138907	0.85030263	0.06205447	-0.02337197	-0.02435553	0.13319913	0.64272605	0.52958121	0.389365487
Keras 20-20 Adadelta		0.67715177	0.62657883	0.85275077	0.08412757	0.01761778	-0.01376741	0.10371261	0.59823965	0.49583254	0.382471568
Keras 50-50 Adadelta	108	0.68133201	0.55844438	0.85245764	0.04039904	0.00225218	-0.03657933	0.15277527	0.66902236	0.49580126	0.379544979
Keras 10-10 Adadelta		0.6703881	0.63858164	0.84881545	0.03434192	0.02151154	-0.0250605	0.06284365	0.598968	0.23614931	0.34294879
Keras 3 Adadelta		0.5466117	0.41986064	0.84465102	0.01061538	0.02528067	-0.02667717	0.07477505	0.55256585	0.27910035	0.302975943
Keras 3-3 Adadelta		0.51949242	0.45349873	0.81284761	0.05036858	0.0493917	0.05434997	-0.02875468	0.4602885	0.17327009	0.282750324
Keras 5-5 Adadelta		0.48897463	0.51267985	0.82563819	-0.0394154	0.0265089	-0.01614763	0.02498356	0.50754028	0.20693451	0.281966321
Keras 10-10-10 Adadelta		0.53711961	0.4888509	0.8119646	-0.04804926	0.06476478	-0.00768442	0.00774752	0.46960796	0.1311136	0.272826143
Keras 2 Adadelta		0.40576461	0.52294548	0.82677568	-0.06529561	0.01817634	-0.03315162	0.01075275	0.40236297	0.10907841	0.244156557
Keras 2-2 Adadelta		0.29476613	0.39325114	0.81189575	-0.07438982	0.04113553	0.01217221	-0.00415795	0.4696831	0.13560101	0.231106344
Keras 50-50-50 Adadelta		0.14498806	-0.0476313	0.42579882	0.00196682	-0.00205149	0.02855089	0.010717	0.70535247	0.44439191	0.190231464
Keras 20-20-20 Adadelta		0.11049254	-0.00252296	0.81916172	-0.06714494	0.03296045	-0.02261005	0.02238805	0.60949017	0.14873865	0.183439292
Keras 30-30-30 Adadelta		0.07520939	0.20274562	0.59830301	-0.0944344	-0.05022409	-0.07906546	0.08632062	0.53678372	0.31314259	0.176531222
Keras 3-3-3 Adadelta		0.46628541	0.42177838	0.2106333	0.03521348	0.01339501	-0.00498095	-0.01659692	-0.00204688	0.3328805	0.161840148
Keras 5-5-5 Adadelta		-0.086383124	-0.155407142	0.827573469	-0.012423455	0.012759751	-0.000759891	0.006371354	0.675912821	0.136232471	0.155986251
Keras 2-2-2 Adadelta		0.07328759	0.25624415	0.46148535	-0.03133619	-0.02305439	-0.04396055	0.05810235	0.18829887	0.41269476	0.150195771
Keras 15-15-15 Adadelta		0.15887339	0.26706057	0.52339799	-0.05062216	-0.06277622	-0.03276501	0.05585237	0.00383042	0.3582226	0.135674883